Studies in Program Obfuscation

by

Mayank Varia

B.S.E., Duke University, 2005

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Mathematics
August 6, 2010

Certified by .....
Ran Canetti
Associate Professor, School of Computer Science, Tel Aviv University
Thesis Supervisor

Accepted by .....
Michel Goemans
Chairman, Applied Mathematics Committee

Studies in Program Obfuscation

by

Mayank Varia

Submitted to the Department of Mathematics
on August 6, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Program obfuscation is the software analog to the problem of tamper-proofing hardware. The goal of program obfuscation is to construct a compiler, called an "obfuscator," that garbles the code of a computer program while maintaining its functionality.

Commercial products exist to perform this procedure, but they do not provide a rigorous security guarantee. Over the past decade, program obfuscation has been studied by the theoretical cryptography community, where rigorous definitions of security have been proposed and obfuscators have been constructed for some families of programs.

This thesis presents three contributions based on the virtual black-box security definition of Barak *et al* [10].

First, we show tight connections between obfuscation and symmetric-key encryption. Specifically, obfuscation can be used to construct an encryption scheme with strong leakage resilience and key-dependent message security. The converse is also true, and these connections scale with the level of security desired. As a result, the known constructions and impossibility results for each primitive carry over to the other.

Second, we present two new security definitions that augment the virtual black-box property to incorporate non-malleability. The virtual black-box definition does not prevent an adversary from modifying an obfuscated program intelligently. By contrast, our new definitions provide software with the same security guarantees as tamper-proof and tamper-evident hardware, respectively. The first definition prohibits tampering, and the second definition requires that tampering is detectable after the fact. We construct non-malleable obfuscators of both flavors for some program families of interest.

Third, we present an obfuscator for programs that test for membership in a hyperplane. This generalizes prior works that obfuscate equality testing. We prove the security of the obfuscator under a new strong variant of the Decisional Diffie-Hellman assumption that holds in the generic group model. Additionally, we show a cryptographic application of the new obfuscator to leakage-resilient one-time digital signatures.

The thesis also includes a survey of the prior results in the field.

Thesis Supervisor: Ran Canetti
Title: Associate Professor, School of Computer Science, Tel Aviv University

Acknowledgments

I am very grateful to many colleagues, friends, and family for their help and support throughout my graduate studies. First and foremost, I am indebted to my advisor Ran Canetti for his guidance for the past four years. As a mathematician working with a computer scientist, we approached most problems from different angles, and understanding his perspective often resulted in solutions and always contributed to my growth as a scientist. Even if we were not able to solve a problem, I knew that Ran would at least provide a list of topical, relevant papers to examine, and the accuracy rate of his suggestions amazes me to this day. Perhaps the highest compliment I can pay him is that he is more helpful from halfway across the world than most people are from halfway across the building.

I am also grateful to my co-authors Yael Kalai, Guy Rothblum, and Daniel Wichs for introducing me to their fields of study and making sense of my rambling thoughts. They were a pleasure to work with, both during the times of intensive study and leisurely banter. I also want to thank Nir Bitanski and Ronny Dakdouk for finding bugs, suggesting improvements, and proving new results in the non-malleable obfuscation and hyperplane membership testing problems. Additionally, I thank Michael Sipser and Peter Shor for serving on my thesis committee. As my internal advisor, Mike also provided valuable advice on my graduate research and job hunt.

Additionally, I want to acknowledge the applied math department for their flexibility in allowing me to find the proper field of research, and the NDSEG fellowship for supporting me for three years, making the transition much easier. I thank Ron Rivest, Eran Tromer, and all of the graduate students in the cryptography reading group for providing an excellent learning environment and cementing my decision to study cryptography.

During the past five years, I have been fortunate to have an outstanding group of officemates, including Victor Chen, Andy Drucker, Matthew Gelvin, Dah-Yoh Lim, and Rotem Oshman. They occasionally helped me with my work and often provided humorous anecdotes and informative conversations. (I promise that the relative frequency of these two events is a reflection on me, not them.) I also thank my friends in the poker group for providing me with a weekly connection to the developments in the math community.

Finally, I thank my parents for their love and support. Without their active interest in my education and well-being, I would never have been in a position to start graduate school, let alone finish it.

Contents

Chapter 1

Introduction to Obfuscation

Cryptography examines how information can be protected while still being used in a meaningful way. The latter part of the goal is particularly important, but often forgotten. Many people view cryptography simply as the art of keeping secrets, but this is not the case. In fact, it is very simple to keep something secret if you never need to recall or use it: simply forget it and continue on as if you had never learned it. The diverse set of problems that cryptographers study all balance the need to use information (a *functionality* guarantee) with the desire to keep something hidden (a *security* guarantee). For instance, classical cryptographic problems such as encryption or digital signatures provide the functionality of being able to send a message to one party with security guarantees against malice by others.

In this work, we consider the problem of *program obfuscation*, in which we wish to hide the details of how a computer program works while preserving its functionality. We illustrate the concept with an example involving two people named Alice and Bob.

Suppose Bob discovers an algorithm that factors integers quickly. He wants to show the algorithm to Alice (along with the rest of the world), but in a responsible manner that does not allow her to break most of the public-key cryptography standards. As a compromise, he decides to write a computer program that inverts the RSA family of one-way functions [82]. It is not known how to factor integers from a generic RSA-breaking program, so Bob's hope is that other trapdoor families like the Rabin function [79] could still remain secure. However, the natural way for him to write the RSA-breaking program uses the factoring algorithm as a subroutine, so Alice could learn how to factor integers anyway. Instead, Bob wants to write the RSA-breaking program in such a way that Alice cannot use it to factor integers, even if she reads the code of his program or monitors the state of her computer while the program runs. Program obfuscation achieves this goal: giving Bob the functionality of writing an RSA-breaking program with the security guarantee that Alice cannot reverse engineer it to learn how it works.

In general, the procedure of *obfuscating* a computer program should garble the program's code enough that it is difficult for an adversarial Alice to "learn" any useful information from it. The extent of the garbling is limited by the fact that the garbled program must perform the same task that the original program does (in Bob's case, breaking RSA). Hence, at the very least Alice can run the program and observe its input-output behavior, and indeed learning of this type is desirable. However, obfuscation should ensure that Alice cannot learn anything else from the program.

As we will see, this is a powerful tool, but a difficult one to construct. In fact, it is even a hard concept to define, as it is unclear what it means to "learn" useful information from the code of a computer program. We will delve into these details later, but first we look at the historical motivations of the problem.

1.1 Obfuscation in applied cryptography

Program obfuscation arose in applied cryptography with the goal of providing a software equivalent to tamper-proof hardware. "Tamper-proofing" encases a computer chip with a protective device that allows the chip to be used but prevents an adversarial user from opening the chip to observe its operation. As a result, the chip designer can protect proprietary trade secrets behind the design. Alternatively, the chip designer can hide a cryptographic key that can be used by the chip but not viewed by an adversary.

The goal of obfuscation is to provide a similar security guarantee in software. That is, we wish to write programs that can be executed, but whose code cannot be examined to learn any additional information beyond simply what can be learned by executing the program. The motivations for this problem are similar to those behind tamper-proofing: preventing hackers from reverse engineering the program, protecting proprietary information, or hiding cryptographic keys inside the program.

At a first glance, this goal seems impossible to achieve. After all, at the end of the day, a program is just machine code that an adversarial user Alice can read. The machine code cannot be unintelligible because the computer must be able to process it. Furthermore, Alice can run the program and observe its execution flow, as well as the intermediate information that is stored in CPU registers or main memory. Thus, Alice can learn a lot of information about the program. Obfuscation cannot prevent this, but instead ensures that this information is "meaningless" to Alice, in the sense that it does not help her understand anything about the program's behavior other than what she could have learned simply by running it.

Nowadays, there exist several commercial *obfuscators* that convert a program that you write into a new program with equivalent functionality but with garbled-looking, undecipherable code. Typically, these programs operate by performing a series of small, local "tweaks" to the code. They do not provide any rigorous security guarantee, but instead provide an informal "security by obscurity" promise to make the code as convoluted as they can. We illustrate these ideas with an example shown in Figure 1-1.

```
void primes(int cap) {
  int i, j, composite;
  for(i = 2; i < cap; ++i) {
    composite = 0;
    for(j = 2; j * j <= i; ++j)
      composite += !(i % j);
    if(!composite)
      printf("%d\t", i);
  }
}
int main(void) {
  primes(100);
}
```

```
_(__,___,____,_____){___/__<=_____?
_(__,___+_____,____,_____):!(___%__)?
_(__,___+_____,___%__,_____):___%__==
___/__&&!____?(printf("%d\t",___/__),
_(__,___+_____,____,_____)):(___%__>
_____&&___%__<___/__)?_(__,___+_____,
____+!(___/__%(___%__)),_____):___<__*
__?_(__,___+_____,____,_____):0;}
main(void){_(100, 0, 0, 1);}
```

Figure 1-1: Security by obscurity example [34]

This example is not from a commercial obfuscator, but illustrates the concept of security by obscurity. The program on the left is a valid C program that outputs the primes less than 100, as can easily be verified by reading the code. The program on the right is also a valid C program with the same functionality, but this is tougher to verify. It would probably take several minutes

to make sense of the code. In fact, the quickest way to check this program's behavior is to compile it, execute it, and observe the pattern in its output. Hence, the code is "meaningless" in the sense that it does not help an adversarial user to learn how the program operates.

Furthermore, the left program can easily be transformed into the right program using a few simple tricks such as variable renaming, using the alternate form of the `if` statement, and changing `for` loops to recursion [34]. The transformations increase the running time of the program but do not change its asymptotic complexity. Commercial obfuscators use a similar, but more complex, series of tweaks to achieve security by obscurity, either at the machine code level or directly on the higher-level language code. Their main objective is to fool commercial reverse-engineering software, with the side-effect of fooling humans as well.

1.2 Obfuscation in theoretical cryptography

In the past decade, the theory community picked up the problem of program obfuscation and formalized rigorous security guarantees that an obfuscator must achieve. There are several definitions in the literature that attempt to tackle this question, and they each have their merits. Their goal is not just to prevent current reverse-engineering software from succeeding, but rather to stop any possible adversary, even ones that have not been designed yet.

In addition to its benefits for the computer science community at large, general-purpose obfuscation has the ability to solve many problems within the field of cryptography. Obfuscation provides a framework to solve cryptographic problems using a simple process: first write the simplest program that solves the desired task if security is not a concern, and then obfuscate this simple program to provide the required security.

If we knew how to perform general purpose obfuscation, we could use this framework to:

- Transform a symmetric key encryption scheme into a public key one by garbling the code of the encryption routine, which generally stores the secret key in plain view [53]
- Construct a fully homomorphic encryption scheme [41, 89] by garbling the code of "simple" addition and multiplication programs that decrypt the ciphertexts (using the decryption key), add or multiply the underlying messages, and re-encrypt the result
- Make a cryptographic hash function by obfuscating the code of a pseudorandom function [47]

In this section, we give an overview of the different definitions and present the known positive and negative results in the field, focusing on their motivations and applications. (Technical details are deferred to Chapter 3.) First though, we compare obfuscation to the "classical" problem of encryption in order to illustrate the difficulty of the problem.

1.2.1 Comparison with encryption

One of the first problems studied in cryptography is encryption, which allows Alice and Bob to send each other messages over the Internet that cannot be read by an eavesdropper. *Symmetric key encryption* schemes require that Alice and Bob share a secret "key" beforehand. However, in a large, connected system like the Internet, this assumption may be impractical. By contrast, *public key encryption* allows Alice to create a secret key that she stores locally, and then she can publish a public key on the Internet that allows anyone (including Bob) to send her hidden messages that only she can read [44, 79, 82].

In program obfuscation, rather than hiding a message, the goal is to hide the runtime operation of a computer program. Intuitively, the garbling procedure "encrypts" a program, so perhaps

techniques from encryption can be used to create a program obfuscator. However, straightforward encryption does not suffice as the program needs to remain functional, and in fact the problems turn out to be very different. There are three major differences between obfuscation and encryption.

Honest and adversarial users

With encryption, Bob's goal is to send a message to Alice, and he wants to protect the message from being viewed by a dishonest eavesdropper. However, with obfuscation, Bob sells a computer program to Alice, but then wants to protect the program from reverse engineering by Alice herself!

In encryption, the dichotomy between honest and adversarial users creates an "information gap": honest users know secret key information that adversarial users do not. This information gap is instrumental in the proof of security. In contrast, there is no distinction between honest users and adversarial users in the setting of program obfuscation. In particular, all users have the same information: the code of the garbled program. The lack of an information gap makes the problem of program obfuscation significantly more difficult to solve than encryption.

Awareness and connectivity

Suppose Bob wishes to send a message to Alice (whether encrypted or not). He will typically know that his message is intended for Alice *before* he writes it, since the contents of the message usually depend on the recipient. Also, people rarely send messages in only one direction, but rather want a two-way communication channel like the Internet so they can send messages back and forth. Because these assumptions are required even to send insecure messages, encryption schemes can take advantage of them.

For example, since Bob knows in advance that his message is intended for Alice, it is reasonable for the encryption protocol to require that he access the Internet and obtain her key information before encrypting the message. Note that Bob needs a two-way communication channel here in order to receive Alice's public key and send her an encrypted message. Similarly, Alice needs to transmit her public key and receive the encrypted message.

By contrast, if Bob is a computer programmer, he typically does not know who will use the programs he creates. He may have a target audience in mind, but he doesn't have the ability (or the time) to tailor the program to each individual user. Similarly, if Alice purchases Bob's program, she should be able to run it without knowing Bob or having an Internet connection.

We want program obfuscation to be able to succeed in this model, in spite of the fewer resources available to Alice and Bob. Note that their only "communication" is through a one-time, one-way channel in which Bob sells the program to Alice. Hence, we cannot utilize public key infrastructure techniques to protect Alice and Bob because:

1. They may not be aware of each other.
2. Even if they were, they don't have a two-way communication channel to exchange keys.
3. Even if they did, it wouldn't be of much help because Bob doesn't want to tailor his computer program for Alice, but rather wants to make one program that works for everybody.

Randomness

In encryption, Alice and Bob choose a secret key at random, and the security guarantee only has to hold with high probability over this random choice. By contrast, in program obfuscation, Bob writes an arbitrary computer program and wishes to garble its code.

It is reasonable to allow the obfuscation process to be randomized (and in fact, this turns out to be necessary [23]), but it is nonsensical to have a security guarantee that only holds for a "random" program. Indeed, it is not clear what the concept of a random program would even mean. Instead, the security guarantee should hold for *every* program that Bob could write.

This distinction may seem trivial, as encryption requires the security guarantee to hold for almost every secret key anyway. However, this small change nullifies most of the common proof arguments used in cryptography. Furthermore, it makes program obfuscation a much more powerful tool.

As we will see later, obfuscation is a very general concept, and it provides a framework in which to solve other cryptographic problems. One of the major contributions of this thesis will be to connect encryption to program obfuscation. These connections improve our understanding of both primitives.

1.2.2 Possible definitions

Recall that the informal goal of program obfuscation is to ensure that a user "learns nothing" by reading obfuscated code. One big reason that this problem only recently moved into the theory world is that it is not immediately clear how to codify what it means to "learn" something. Instead, Barak *et al.* [10] approach the problem from a different angle: obfuscators produce garbled code, and the definition states when the code is "sufficiently garbled."

Consider an imaginary world in which people can give others access to oracles[1] at will. In this imaginary world, we can easily perform perfect obfuscation by giving users oracle access to a computer program. The oracle allows them to learn the program's input-output functionality, but any other aspect of the program's behavior is hidden from the users. Unfortunately, in the real world we cannot hand out oracles to other people. Instead, we want obfuscators to be able to replicate the power of oracles in the imaginary world.

The definition of obfuscation provided by Barak *et al.* [10], called *virtual black-box obfuscation*, is aimed at achieving this goal. The security guarantee uses a simulator-based definition that considers two different worlds. In the real world, Alice receives obfuscated program code and learns a predicate about the underlying program. The code is sufficiently obfuscated if there exists an efficient[2] *simulator* in the imaginary world that is given an oracle to the program and also learns the predicate. Hence, anything that Alice learns by deciphering the garbled code, she could have learned simply by observing the input-output behavior of the program.

More precisely, the virtual black-box definition requires that for every adversary, there exists a simulator such that for *all* programs P and predicates π, the simulator can learn $\pi(P)$ in the imaginary world as well as the adversary can in the real world. We emphasize two technical features of this definition.

1. The universal quantifier over programs is stronger than the requirement in most cryptographic definitions, which only require the simulator to succeed on a random instance. However, the concept of a "random program" usually does not make any sense, so we impose the stronger requirement.

2. The definition, as stated so far, is unachievable for some predicates. Informally, the issue is that Alice can *copy* the obfuscated code that she receives, even if she doesn't understand it, whereas the simulator cannot copy an oracle.

[1] Informally, an "oracle" is a black box with input and output wires. When an input is fed to the box, it instantly provides an output. However, it is impossible to peer inside the black box and observe its behavior.

[2] Throughout this work, an "efficient" algorithm is one that runs in probabilistic polynomial time.

To sidestep this problem, the virtual black-box property only considers binary predicates (i.e., predicates with one bit of output). With this restriction in place, the definition is achievable but the security guarantee is weakened. One of the main contributions of this thesis is to address this limitation. We will return to this issue in Section 1.3.2.

The virtual black-box property has several variants that may be more useful in certain circumstances. Goldwasser and Kalai [43] add auxiliary input to the definition. Bitanski and Canetti [12] allow the simulator to be inefficient. Several works [39, 53, 55] add a notion of randomness that makes sense in some applications, eliminating the universal quantifier mentioned above.

Finally, Goldwasser and Rothblum [46] form a definition that is not simulator-based. Rather than trying to explain when a program is sufficiently garbled, their definition of *best-possible obfuscation* simply aims to garble the program "as much as possible." This notion is weaker than the standard virtual black-box property [46], even if the simulator is allowed to be inefficient [12], so it may be easier to satisfy.

The results in this work all use the virtual black-box definition (possibly with auxiliary information), but in this section (and in Chapter 3) we survey the prior results known under all of the definitions.

1.2.3 Negative results

Now that we have codified what it means for a program to be sufficiently garbled, all that remains is to find a way to perform the garbling procedure. Here we hit the second big reason that the problem of program obfuscation has not been studied much by the theory community: it appears to be too difficult. The early works that codified the problem also proved that it is impossible to achieve in general [10, 43, 46].

In fact, these papers construct "unobfuscatable" programs with the property that no garbling of these programs can hide "all information." Intuitively, the negative results exploit the fact that even if you do not understand garbled code, you may feed it to another program that can use it as a subroutine in performing a meaningful task.

This is disappointing, but not necessarily dire. Our dream goal of constructing a general-purpose obfuscator may have vanished, but on a positive note, many of the "unobfuscatable" programs in the impossibility proofs are rather contrived, and are not likely to be programs that anyone would want to obfuscate anyway.

That is, despite the general impossibility results, we may still hope to find specific obfuscators for every program of interest. More precisely, perhaps we can find an obfuscator for a restricted family $\mathcal{C}$ of programs. In this setting, we make two relaxations:

1. The obfuscator only needs to garble programs in $\mathcal{C}$. (Its behavior on programs outside $\mathcal{C}$ is irrelevant.)

2. The garbled code may reveal that the program belongs to $\mathcal{C}$. We only wish to hide *which* program in $\mathcal{C}$ it is.

Ideally, we would like the family $\mathcal{C}$ to be as large as possible. In an attempt to maintain some semblance of generality, one might hope to obfuscate programs in a (low) complexity class.[3]

Sadly, even this task appears too difficult to achieve. The unobfuscatable programs, while perhaps contrived, are remarkably simple programs. Using the virtual black-box definition, it turns

[3]When obfuscating a low complexity class, the second relaxation means that obfuscated programs are allowed to have low complexity as well. Without this relaxation in place, the garbled code would have to hide the running time of the original program, which is very difficult.

out to be impossible even to obfuscate all constant-depth threshold circuits (TC^0) under common cryptographic assumptions such as the hardness of the Decisional Diffie-Hellman problem or factoring Blum integers [10]. The situation is even worse when using the best-possible obfuscator definition, where it is impossible to obfuscate 3-CNF circuits (a subset of AC^0) unless the polynomial hierarchy collapses to the second level [46]. Hence, our best hope of a general construction is to garble very simple programs like constant-depth circuits, and even a seemingly small task such as this would be a major breakthrough in the field!

Finally, we note that the unobfuscatable programs are not all contrived and irrelevant; on the contrary, they can implement productive tasks. For instance, there exist unobfuscatable encryption schemes, digital signature schemes, and pseudorandom functions [10]. Hence, even obfuscating every program of interest to cryptographers, rather than a complexity class, is still not possible.

1.2.4 Positive results

It may seem at this point as though we have no hope of finding any positive results in program obfuscation, but thankfully this is not the case! The general impossibility results indicate that we should consider targeted families of simple programs as candidates for obfuscation, and indeed this path has been successful.

The trivial case: learnable programs

A family of programs $\mathcal{C}$ is *learnable* if, given an oracle to any program in $\mathcal{C}$, one can uniquely determine which program the oracle represents by making a small (i.e., polynomial) number of queries to the oracle. A simple example is the family of "constant programs" that disregard their input and always output a string that is stored in its memory. (Different programs in the family output different strings.) Given an oracle to a constant program, it is easy to determine which constant program it is (i.e., which string it stores) by making a single oracle query with the all 0s input and noting the response.

It seems as though learnable programs should be unobfuscatable, since no matter how garbled one makes a learnable program, it is always possible to learn its behavior. Somewhat counter-intuitively, however, the virtual black-box definition considers learnable families to be trivially *obfuscatable* for this very reason: the original program is already as garbled as possible [63].[4]

While these may be positive results *per se*, they are not appealing ones, as they do not provide any insight on how the garbling procedure should be done. Hence, in the rest of this thesis, we only consider unlearnable programs. In fact, we look at the "opposite" end of the spectrum and study programs that require exponentially many queries to learn any useful information about their behavior.

The interesting case: unlearnable programs

Obfuscators have been constructed for some specific families of "extremely unlearnable" programs. In fact, some specific constructions were found even before the general definitions of obfuscation were codified! The security guarantees in these results were specific instantiations of the general virtual black-box guarantee.

[4] Returning to the question of obfuscating complexity classes, we can find incredibly small classes that are learnable, and thus trivially obfuscatable. However, the classes mentioned in the previous section, AC^0 and TC^0, are unlearnable. Moreover, they contain many cryptographically interesting functionalities, so an obfuscator for these "large" classes would have many practical uses [7].

However, all of the constructions come with a big caveat: they cannot be proved secure under "standard" cryptographic assumptions. Hoeteck Wee [93] shows this limitation is not a weakness of the particular constructions but is intrinsic to obfuscation.

In this section, we describe the results at a high level, focusing on their applications. Details on the constructions of these obfuscators can be found in Chapter 3.

Login programs

A *login program* stores a password in its memory and receives an input string from the user. The program accepts the input string only if it equals the password. The simplest functionality for a login program is a *point circuit*, which stores the password w in a clearly identifiable manner, and computes

$$I_w(x) = \begin{cases} 1 & \text{if } x = w, \\ 0 & \text{otherwise.} \end{cases}$$

Using a point circuit itself as a login program is unwise because storing the password in the clear poses a security risk. Instead, a secure login program should only store a "garbled" version of the password that can still be used to perform the equality test but cannot be used to learn the password. We can use obfuscation to achieve this goal under a variety of cryptographic assumptions [23, 28, 63, 93].

The virtual black-box definition explains the extent of the "garbling" required. The simulator is only given black-box access to the login program, so it can only find the hidden password by running a *dictionary attack*, guessing passwords until it finds the right one. Hence, the same is true of an adversary's ability to understand the code of an obfuscated login program. In particular, the adversary cannot learn anything from the garbled password to help the dictionary attack or perform a quicker attack.

In practice, most login programs garble the password using a cryptographic hash function, which would also make the password unintelligible to the adversary *if* the password were chosen uniformly at random. However, humans typically choose structured passwords like short alphanumeric strings, in which case hash functions do not provide any guarantee. It is conceivable that an adversary can break the cryptographic hash function on common passwords, even if it is hard to break in general.

By contrast, the security guarantee of program obfuscation applies even if the password is chosen from a very low entropy distribution, due to the randomness property described in Section 1.2.1. Note that if the password is chosen extremely poorly, it may be simple for a dictionary attack to succeed, so we have no hope of protecting a login program in this case. However, as long as it is difficult to guess the password, an obfuscated login program protects it.

Generalizations

There are many straightforward generalizations of the point circuit functionality. First, we can create multi-user login programs. Specifically, consider *multi-point circuits*

$$I_{\{w_1,\ldots,w_m\}}(x) = \begin{cases} 1 & \text{if } x \in \{w_1, \ldots, w_m\}, \\ 0 & \text{otherwise.} \end{cases}$$

that store a list of passwords in a readily identifiable manner, and accept if their input equals any of the stored passwords. Note that the passwords need not be unique, and in fact the set of passwords may be empty. We can obfuscate the family $\mathcal{P}^m$ of circuits that accept up to m passwords [12, 25].

Additionally, we can obfuscate "fuzzy" point circuits with proximity detection that accept their input if it is "close" to the stored password, which is useful for biometric inputs [39]. We can also create a substring matching program that accepts any input that contains the stored password, which can be used to produce an anti-virus program that checks its input for the signature of a virus without revealing the signature it is looking for.

Finally, we can consider point circuits that do not simply give a yes or no response, but rather reveal a hidden message upon receiving the correct password. Canetti and Dakdouk [25] obfuscate these *point circuits with multi-bit output*

$$I_{(k,m)}(x) = \begin{cases} m & \text{if } x = k, \\ \bot, 1^{|m|} & \text{otherwise.} \end{cases}$$

Let $\mathcal{I} = \{I_{(k,m)} : k, m \in \{0,1\}^*\}$ be the set of all these circuits. Obfuscations of circuits in $\mathcal{I}$ are called *digital lockers* because they hide the message in such a way that it can only be revealed to a user that knows the secret key k; otherwise, the user only learns the length of m. This functionality is very similar to symmetric key encryption, with the added benefit that the secret key does not have to be chosen uniformly, but can be a human-chosen string with poor entropy. We will pursue the connection between digital lockers and symmetric key encryption much further in Section 1.3.1.

Cryptographic applications

The obfuscators for all of the circuit families described above use the virtual black-box definition, except for the obfuscator for point circuits with proximity detection, which incorporates randomness to describe the level of "fuzziness" tolerated [39]. As described in Section 1.2.1, in general it does not make any sense to obfuscate a "random" program, but in specific circumstances it may. In particular, cryptographic applications lend themselves well to a randomized definition.

For example, the encryption routine in most symmetric key encryption schemes stores the secret key in plain view. If the routine is obfuscated under such a definition, the resulting program can still be used to encrypt messages but not to determine the secret key, thereby creating a public key encryption scheme [53].

Additionally, the randomized definition of obfuscation can be used to construct a "proxy re-encryption scheme," which allows a server to forward encrypted email from one client to another client without learning any secret keys [55].

Similarly, we can construct an "encrypted signature" program, which can take a message, digitally sign it, and output an encryption of the signature without revealing the signing key [48]. This functionality can be used to implement a signcryption scheme.

Finally, we can "shuffle ciphertexts in public" [1], which is useful in voting algorithms. With this device, one can "shuffle" a list of ciphertexts to produce a new list of ciphertexts that contain the same messages but in a randomized order. Furthermore, this can be performed in the presence of observers who can verify that the messages are properly shuffled (i.e., the output contains the same messages as the input) without learning the secret key or the permutation connecting the input to the output.

1.3 Our results

This thesis contains three separate results that improve our understanding of program obfuscation, with a focus on what program obfuscation *can* do rather than what it cannot do. We connect

the problem to symmetric key encryption, strengthen the definition to provide non-malleability guarantees, and construct an obfuscator for a new family of programs.

1.3.1 Obfuscation and symmetric encryption

Symmetric key encryption allows two people to communicate secret information over an open channel as long as they have a shared secret key. The classic view of symmetric encryption assumes that:

1. The secret key is chosen from precisely the distribution specified by the encryption scheme (typically a uniformly random string).

2. The encryption and decryption algorithms are executed in a completely sealed way, so no information about the key is leaked to the eavesdroppers.

3. The parties use the key only in the encryption and decryption routines and not for any other purpose. In particular, their messages are never directly related to the key.

In practice, however, none of the classical assumptions may hold. Even if the honest parties try to follow the specification, they may be foiled by an imprecise or defective source of randomness, or by an active attacker [51]. Additionally, in some applications (such as hard drive encryption) it may be desirable to encrypt messages that are related to the secret key. In recent years much research has been done to find encryption schemes that can tolerate (one of) these types of attacks.

Three forms of symmetric encryption

One line of research considers the case where the key is chosen using a "defective" source of randomness that does not generate uniform and independent random bits [3, 5, 38, 58, 71]. In this model, the key k is taken from a distribution that is adversarially chosen, subject to the constraint that the min-entropy of the distribution is at least some pre-specified function $\alpha(|k|)$. In this case, the scheme is said to be secure with respect to α-weak keys.

A different relaxation of the classic model considers the case where the key is chosen uniformly but some arbitrary function of the key $\ell(k)$ is *leaked* to the adversary [3, 71]. This models both direct attacks where the adversary gains access to the internal storage of the parties, such as the cold-boot attack of [51], and indirect information leakage that occurs when the shared key is derived from the communication between the parties, such as the information exchange used to agree on the key. Of course, all security is lost if the adversary learns the key in its entirety, and therefore some restriction needs to be imposed on the amount of information that the adversary can get.

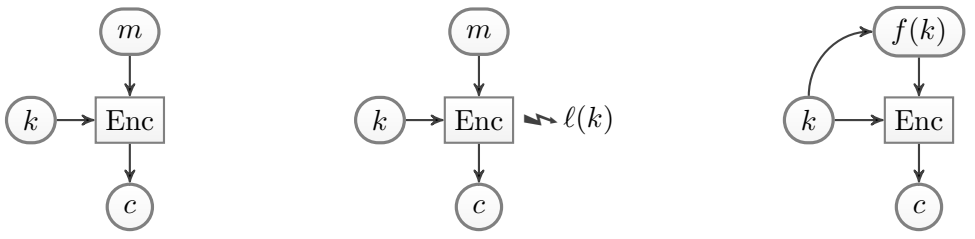

Figure 1-2: Pictorial representation of traditional (left), weak key or leakage-resilient (middle), and key-dependent message secure (right) encryption schemes.

One possibility is to require that the key has some significant statistical entropy left, even given the leakage. We call this the *entropic* setting and note that it is equivalent to the weak key setting described above [26, footnote 2]. Another, stronger, security notion only insists that it is computationally infeasible to compute the secret key from the leaked information, even if the key is information-theoretically determined from the leakage. We call this type of leakage *auxiliary input*.

Finally, a third line of research examines the case where the messages may depend on the shared key. Security in this more demanding setting was termed *key-dependent message* (KDM) security by Black, Rogaway and Shrimpton [13]. In the last few years, KDM security has been extensively studied [6, 8, 9, 18, 22, 50, 52, 54], and several positive results emerged, most notably the results of [6, 18] who showed how to obtain KDM security with respect to the class of affine functions[5] under the DDH and learning with errors assumptions, respectively.

While the constructions for KDM secure schemes and α-weak key schemes bear significant similarities to each other (such as [18, 71], [6, 38], and [3, 6]), no formal connections between the problems have been made so far.

A common ancestor

In Chapter 4, we show tight relations between symmetric key encryption and obfuscation [26]. Specifically, we show that symmetric key encryption with weak key resilience, leakage resilience, and KDM security can all be viewed as natural special cases of digital lockers. In fact, digital lockers provide weak key resilience and KDM security simultaneously. In addition to providing insight and intuition to these primitives, the connections provide new constructions and hardness results for the primitives considered.

Recall from Section 1.2.4 that a digital locker is an obfuscated point circuit with multi-bit output that reveals a message m if and only if its input equals the hidden key k. The connection between digital lockers and encryption was first pointed out by [25]. The standard virtual black-box property yields a strong security guarantee for digital lockers: for any adversary with binary output, there exists a simulator such that for *any* k and m, the output of the adversary given the digital locker is indistinguishable from the output of the simulator given oracle access to the digital locker. Due to the universal quantifier on keys, the message remains hidden even when the key is taken from a distribution which is not uniform, as long as it has sufficient min-entropy that it cannot be guessed in polynomial time.

Due to the strong security requirement, constructions of digital lockers are based on very strong and specific assumptions, such as the existence of fully-composable point circuit obfuscators [12, 25, 63]. Different constructions exist for restricted settings, such as when m is shorter than k, or when m is chosen independently from k [25, 38]. In this work, we examine these and other restricted settings for digital lockers.

Equivalence of terminology

The goal of this work is to relate the various notions of encryption to different notions of obfuscation of the family $\mathcal{I}$. To do so, we generalize (and weaken) the virtual black-box definition by relaxing the "for any" requirement on programs in the virtual black-box property described in Section 1.2.2. Instead, we merely require that k and m are sampled from a distribution with certain properties. The adversary and simulator are aware of the properties but not the particular distribution used.

[5] These functions specify the dependence between the secret key and the message that is encrypted. Restrictions on the function seem necessary because [50] shows that there is no black-box reduction from the (unrestricted) KDM security of any encryption scheme to "any standard cryptographic assumption." See Section 4.6.3 for more detail.

We show that different types of encryption correspond to obfuscation for distributions with different properties.

Obfuscation vs. weak key and leakage-resilient encryption. We say that an obfuscator for $\mathcal{I}$ is *α-entropic with independent messages* if it satisfies the above definition for product distributions on k, m where the distribution of k has min-entropy at least α. Note that the product distribution ensures that m is drawn independently of k. We impose no restriction on the entropy of m.

Our first result is that α-entropic digital lockers with independent messages are equivalent to symmetric key encryption with α-weak keys. This equivalence includes the traditional notion of encryption, in which the secret key is chosen uniformly and is not leaked, by setting $\alpha(n) = 2^n$.

We describe both directions of the equivalence. Given an obfuscator $\mathcal{O}$ for $\mathcal{I}$, we form an encryption scheme by $\mathrm{Enc}_k(m) = \mathcal{O}(I_{(k,m)})$ and $\mathrm{Dec}_k(c) = c(k)$, where the ciphertext c is interpreted as the description of a circuit. Conversely, given an encryption scheme, we form an obfuscator as follows: given a key k and message m, we form a circuit that that hard-codes the string $c = \mathrm{Enc}_k(m)$, and on input x, runs $\mathrm{Dec}_x(c)$ and outputs the result. For the correctness of obfuscation, we require that the encryption scheme can *detect* if it is decrypting a ciphertext with an incorrect secret key. We show that this property can be added generically to any semantically secure encryption scheme.

Fully entropic obfuscation and fully weak key security. An obfuscator for $\mathcal{I}$ with respect to independent messages is said to be *fully entropic* if it satisfies α-entropic security for all super-logarithmic functions α. If we start with such an obfuscator, the transformation above produces an encryption scheme with semantic security for *fully weak keys*; that is, any key distribution with super-logarithmic entropy.

To connect our new α-entropic definition to previous works, we show that any obfuscator that is fully entropic also satisfies the (standard) virtual black-box property; that is, it works for any k and m. A proof of this result is trickier than it might seem, with the main difficulty being that in the case of α-entropic security the simulator has the bound α, whereas in the virtual black-box case no such bound exists.

CPA security and self-composability. If we start with a chosen plaintext attack (CPA) secure encryption scheme, the resulting obfuscator is *self-composable* in the sense that security is preserved even if many digital lockers are produced with the same key and (possibly) different messages. This property is not, in general, implied by obfuscation alone [25]. The converse is also true.

Auxiliary input. If we start from a leakage-resilient encryption with auxiliary input, then the resulting obfuscator satisfies the virtual black-box definition with respect to auxiliary input [43]. The converse is true as well.

KDM security. The above equivalence results were stated with respect to the restricted notion of obfuscation to *independent* messages. Interestingly, the standard notion of obfuscation provides the additional (and very powerful) security guarantee for encryption with key-dependent messages. This increased level of security can be combined with leakage on the key.

We say that an obfuscator for $\mathcal{I}$ is *α-entropic with dependent messages* if it withstands any joint distribution on (k, m) where the projection distribution on k has min-entropy at least α. Note that the message m can depend on k, and in fact we typically view m as a function of k. A digital locker of this form is equivalent to an α-KDM semantically secure encryption scheme, via the same transformations as before.

Semantically secure encryption with:	Is equivalent to digital lockers with:
α-weak keys	α-entropic security for indep messages
fully weak keys	fully entropic security, indep messages
auxiliary input	auxiliary input
CPA security	self-composability
KDM security	dependent messages

Figure 1-3: Equivalence between symmetric encryption (left) and obfuscation (right) terminology.

Multiple extensions. The equivalences between obfuscation and encryption are summarized in Figure 1-3. They can be combined arbitrarily, with two caveats.

First, we do not consider KDM security with auxiliary input. Second, when combining CPA and KDM security, we require that the function connecting the message to the key be chosen non-adaptively prior to viewing any ciphertexts. This restriction still permits common uses of KDM security, such as *circular security* in which $k = m$ [12].

Implications

The connections between obfuscation and encryption allow us to translate results from one field into the other. Here, we describe three such implications, two positive and one negative. See Section 4.6 for more details.

Secure encryption with (fully) weak keys. The known constructions of α-weak key secure encryption schemes require that the bound α be chosen in advance, and then the scheme is constructed based on α. By contrast, our transformation applied to the digital locker construction in [25] yields an encryption scheme that simultaneously achieves α-weak key security for all super-logarithmic functions α, under the strong DDH assumption in [23]. The main advantage of this scheme is that the min-entropy α does not need to be chosen in advance.

We remark that the hardness assumption in [23] has a similar flavor – it explicitly makes an assumption for every distribution with super logarithmic min-entropy. The crucial point is however that the construction does *not* depend on α and so it provides a trade-off between the strength of the assumption and the strength of the obtained guarantee.

Constructing self-composable digital lockers with independent messages. We can apply our transformations to known constructions of encryption schemes that are secure with α-weak keys. This results in self-composable digital lockers for independent messages, with one caveat: the security guarantee only applies to distributions in which both k and m are efficiently sampleable.

Impossibility results for obfuscation. Using our transformations, the negative result of Haitner and Holenstein [50] implies that obfuscators for $\mathcal{I}$ cannot be proven secure via a "black-box reduction to standard cryptographic primitives." The impossibility carries over to fully composable obfuscators of point circuits [25] as well.

1.3.2 Non-malleable obfuscation

We motivated obfuscation in Section 1.1 as the software equivalent of tamper-proof hardware. However, the virtual black-box property does not quite meet this standard. Tamper-proof hardware

actually provides two separate guarantees: first, that an adversary cannot learn how the chip works, and second, that the adversary cannot tinker with the chip (say, by inserting or cutting wires).

The virtual black-box property provides an "unlearnability" guarantee, but does not limit the ability to modify an obfuscated program. In fact, many of the known constructions are malleable [12, 23, 25, 28]. In Chapter 5, we extend the virtual black-box definition to incorporate prevention and detection guarantees against modifications, and give constructions for login programs that meet the stronger definitions [32].

Malleability concerns

If an adversary has access to obfuscated code, the virtual black-box property guarantees that she cannot "understand" the underlying program. However, suppose the adversary instead uses the obfuscated code to create a new program in such a way that she controls the relationship between the input-output functionality of the two programs.

Intuitively, one might expect that virtual black-box obfuscation already prevents modifications. The simulator only has oracle access to the obfuscated code, so any program that it makes can only depend on the input-output functionality of the obfuscated code at a polynomial number of locations. Therefore, obfuscation should guarantee that the adversary is also restricted to these trivial malleability attacks. However, the virtual black-box definition does not carry this guarantee. As described in Section 1.2.2, it only considers adversaries and simulators that output a single bit, not entire programs.

A naïve solution to this problem is to extend the virtual black-box definition to hold even when the adversary and simulator output long strings. However, in this case obfuscation becomes unrealizable for any family of interest. Consider the adversary that outputs its input. Then, a corresponding simulator has oracle access to a program and needs to write the code for this program, which is impossible for any unlearnable[6] family of programs. This is why Barak *et al.* restricted predicates to binary output in the first place.

In this paper, we demonstrate two different methods to incorporate non-malleability guarantees into obfuscation. Both non-malleability definitions extend the virtual black-box definition by allowing the adversary and simulator to produce multiple bit strings, but only in a restricted manner. There are many subtleties involved in constructing a proper definition, such as deciding the appropriate restrictions to impose on the adversary and simulator, and creating relations to test the similarities between the adversary's input and output programs. We defer treatment of these important details to Section 5.2. Here, we motivate and describe the two definitions at a high level.

Tamper-proof obfuscation

Imagine that Alice, Bob, and Charles are three graduate students in an office that receives a new computer, which the department's network administrator configures to allow the graduate students root access to the computer. The administrator receives the students' desired passwords, and she writes a login program that accepts these three passwords and rejects all other inputs. The students will be able to read the program's code, so the administrator should obfuscate the login program in order to ensure that a dictionary attack is the best that the graduate students can do to learn their officemates' passwords.

However, obfuscation does not alleviate all of the administrator's fears. If the students need to have root access to the computer for their projects, then they can not only read the login program but can alter it as well. The administrator would like to prevent tampering of the login program,

[6] Recall from Section 1.2.4 that unlearnable families are the meaningful ones to obfuscate.

but the virtual black-box definition does not provide this guarantee. For instance, suppose Alice wants to remove Bob's access to the computer. There exist obfuscated login programs such that Alice can succeed in this attack with noticeable probability [12, 25], which we describe in the Constructions section below.

Recall that the goal of obfuscation is to turn a program into a "black box," so the only predicates that Alice can learn from the program are those she could learn from a black box. We extend this intuition to cover modifications as well. We say that an obfuscator is *tamper-proof* if the only programs that Alice can create given obfuscated code are the programs she could create given black-box access to the obfuscated code.

We define tamper-proof obfuscation using a simulator-based definition. A straightforward extension of the Barak *et al.* definition [10] would require that for every adversary that receives obfuscated code and uses it to create a new program, there exists a simulator that only has oracle access to the obfuscated code and produces a program that is functionally equivalent to the adversary's program. However, we saw that this definition is too strong, with the main issue being that the adversary can "pass" its obfuscated program along to its output program, whereas the simulator does not have this capability. We fix the imbalance by allowing the program that the simulator outputs to have oracle access to the obfuscated code.

This definition limits the possible attacks Alice can apply. For instance, if Alice only has black-box access to the login program, then she can only remove Bob's access to the computer with negligible probability. In this sense, tamper-proofing gives Bob more security because it protects all aspects of his access to the computer, whereas the virtual black-box property only protects his password.

Tamper-evident obfuscation

Tamper-proof hardware protects the internal circuitry of a chip, but it does not prevent all possible modifications. An adversary could insert the protected chip on a circuit board and design circuitry around it so that the overall board has a similar, but slightly different, behavior than the original chip.

Tamper-proof obfuscation is vulnerable to the same problem in software. For example, suppose that Alice wishes to play an April fools' prank on her officemates by altering the login program to accept their old passwords appended to the string "Alice is great." Alice only knows her own password, so she cannot run the obfuscator to produce this modified program. Nevertheless, she can write the following program: "on input a string s, check that s begins with 'Alice is great,' and if so, send the rest of the string to the administrator's login program." Tamper-proofing does not prevent this prank. In fact, it is impossible to prevent this prank because Alice only uses the obfuscated login program in a black-box manner.

Still, this attack is not perfect: after Alice performs this attack, the new login program "looks" very different from a program that the network administrator would create. As a result, we may not be able to prevent Alice from performing her prank, but we can *detect* Alice's modification afterward and restore the original program. Alice's job is now harder, since she has to modify obfuscated code in such a way that the change is undetectable.

We say that an obfuscator is *tamper-evident* if the only programs Alice can create that pass a verification procedure are the programs she could create given black-box access to the obfuscated code.[7] This approach gives us hope to detect attacks that we cannot prevent, although it requires

[7] A historical note: the two forms of non-malleability were originally called "functional" and "verifiable" non-malleable obfuscation [31, 32]. We change to more intuitive terminology here in order to emphasize the analogy with hardware.

a stronger model in which a verification procedure routinely audits the program.

In our example, one simple way to achieve non-malleability is for the network administrator to digitally sign every program she makes, and for the verification procedure to check the validity of the signature attached to an obfuscated program before running it. By the existential unforgeability of the signature scheme, Alice cannot make any modifications, so the non-malleability goal is achieved.

However, this solution is unsatisfactory because it requires that the verification procedure can find and store the network administrator's verification key, but as described in Section 1.2.1, we do not want to assume that users in our model have access to the Internet or are aware of each other. As a result, in this work we consider "public" verifiers that depend only on the obfuscation algorithm, and not on the party performing the obfuscation. Informally, a *verifier* algorithm V accepts programs if and only if they could have been produced by running the obfuscation algorithm. We stress that V does not receive any party-specific information (such as public keys), so it does not depend on the person that runs the obfuscator.

Verifiability has interesting applications in and of itself, as we describe in the Discussion section below, but we only use it to create a simulator-based definition of tamper-evident obfuscation. The definition guarantees that an adversary cannot modify obfuscated code into a new program that passes the verification test unless there exists a simulator that can perform the same attack given only oracle access to the obfuscated code. (Note that the adversary must create a new program and not simply output the obfuscated code it receives.) Because we hope to detect attacks that operate in a black-box manner (which we could not hope to prevent in the tamper-proof setting), we no longer give the simulator the extra help that we gave it in the definition of tamper-proof obfuscation. Instead, the simulator must output a fully-functional program that does not have an oracle.

Discussion on verifiability

Tamper-evident obfuscation incorporates a concept of verifiability that is useful even in situations where malleability is not a concern. To illustrate, we return to the example in which Bob sells an obfuscated RSA-breaking program to Alice. Obfuscation helps Bob by protecting his factoring algorithm, but it hurts Alice because she has to install and run a program without knowing what it does. For all she knows, the program could contain a virus, and because the program is obfuscated there is no hope for a virus checker to detect the presence of a virus. Hence, Bob needs a verification algorithm to prove that he is selling Alice a program from the proper family.

Despite the broader applicability, in this thesis we only use verifiability as a tool in tamper-evident obfuscation.

Comparison

We show that both forms of non-malleability imply the virtual black-box property. Intuitively, this relationship holds because an adversary that outputs programs should easily be able to encode a single bit of information in the output. As a result, all known impossibility results regarding the virtual black-box property continue to hold for both types of non-malleability [10, 43].

Additionally, we compare the two flavors of non-malleability. Tamper-proofing *prevents* as many malleability attacks as possible, whereas tamper-evidence *detects* as many attacks as possible. Intuitively, these goals are incomparable: the tamper-evident definition is stronger because we can detect more attacks than we can prevent, but on the other hand it is weaker because the model requires its participants to understand and apply the verification algorithm. We justify this

intuition by showing that in the random oracle model, the two definitions of non-malleability are indeed incomparable.

Constructions

In the random oracle model, we show that the obfuscator for point circuits in [63] satisfies both forms of non-malleability. Next, we study the family of multi-point circuits $\mathcal{P}^m$ that store a constant number of passwords. One idea to obfuscate the program that accepts passwords $w_1, \ldots, w_m$ is as follows (see Figure 1-4 for a diagram):

1. Use several instantiations of a single-point circuit obfuscator in order to create obfuscated programs $I_{w_1}, I_{w_2}, \ldots, I_{w_m}$, where each I_{w_j} accepts only the password w_j.

2. Create the program P that contains I_{w_1} through I_{w_m} as subroutines, and on input x, iteratively feeds x into the I_{w_j} and accepts if any one of these programs accept.

This methodology is known as *concatenation*, and it is shown in [12, 25] that concatenation preserves obfuscation under sufficiently strong assumptions. That is, given an obfuscator $\mathcal{O}$ for the family of single-point circuits, concatenation produces an obfuscator for the family of multi-point circuits.

However, concatenation does not preserve non-malleability. The program P stores the subroutines I_{w_1} through I_{w_m} in a readily identifiable way, so an adversary can modify one password by changing one of the subroutines. This is true even if the obfuscator for single-point circuits is non-malleable.

In the tamper-evident setting, we resolve the problem with concatenation by using a self-signing technique to ensure that the subroutines are not modified. The verification algorithm associated with this construction runs the verification algorithm for the signature scheme. This approach does not suffice in the tamper-proof setting, where the self-signing technique is useless because there is no guarantee that anybody checks the signature. Instead, we "glue" the passwords together in such a way that any attempt to change the obfuscated code destroys information on all of them simultaneously.

We also give a construction that does not use random oracles. Instead, it uses the common reference string (CRS) model, in which a sequence of bits is chosen uniformly at random and published in a public location that all participants can access. (Note that this is different from a public key infrastructure because the CRS is not tied to the specific identity of the party performing

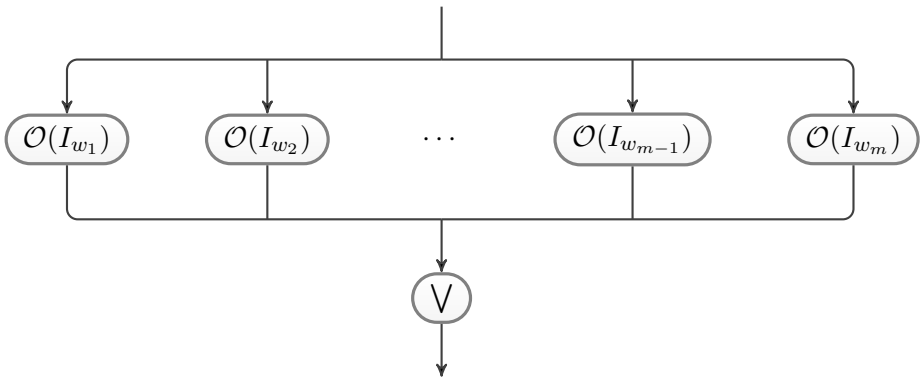

Figure 1-4: Pictorial representation of an obfuscated m-point circuit that is constructed by concatenating m obfuscated single-point circuits.

the obfuscation.) We construct a tamper-evident obfuscator for point circuits by providing any (potentially malleable) obfuscator along with a non-malleable NIZK proof of knowledge [83, 84] that the obfuscator knows the password.

Informally, a non-malleable NIZK proof of knowledge considers an adversary that can request multiple proofs for statements of its choice and then produces a new proof. The non-malleability guarantee requires that the adversary knows a witness to its constructed proof, so it cannot simply modify the old proofs to prove a new statement.

Intuitively, our construction is tamper-evident because an adversary can only make a program that passes the verification test if she knows its functionality, so she cannot modify an obfuscated login program without understanding it first, which is impossible by the virtual black-box property. However, the actual proof turns out to be delicate. Using proof techniques from [23], we achieve a somewhat weaker variant of tamper-evidence. Specifically, we show that for a large class of relations, no adversary can perform a modification that satisfies the relation with noticeable probability.

1.3.3 Obfuscation of hyperplane membership

Under the standard virtual black-box definition, obfuscators have only been constructed for very simple functionalities such as login programs and digital lockers. These programs have an "all or nothing" knowledge property: a simulator with oracle access to such a program learns (almost) nothing about its functionality until it guesses the correct password, and then it knows the complete functionality of the program.[8] There is no intermediate state in which the simulator learns some meaningful information about how the program operates but doesn't know its complete functionality. This fact is heavily exploited by the proofs of security for obfuscators of these functionalities.

In Chapter 6, we obfuscate a new functionality that performs arithmetic operations, and not just equality testing. It does *not* have the "all or nothing" property: a simulator can learn how the program behaves on exponentially many inputs without knowing its complete behavior [30].

The new functionality

The objective of this work is to obfuscate programs that perform "hyperplane membership testing." Let P be a hyperplane in a vector space, and let H_P be a program that tests whether its input is a point on the hyperplane.[9] An obfuscated version of H_P can perform the test without revealing additional information such as the distance from the input point to the hyperplane or any other points that are on the hyperplane.

More precisely, given a prime p and positive integer d, consider the family of hyperplanes through the origin in the vector space $(\frac{\mathbb{Z}}{p\mathbb{Z}})^d$ over the finite field $\frac{\mathbb{Z}}{p\mathbb{Z}}$. In this setting, a hyperplane can be defined by a vector that is orthogonal to every point in the plane. Let $\boldsymbol{a}$ be a vector in $(\frac{\mathbb{Z}}{p\mathbb{Z}})^d$ and consider the program

$$H_{\boldsymbol{a}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We construct an obfuscator for this family of programs.

[8] This is perhaps too harsh: after making an incorrect guess to an obfuscated login program, the adversary does learn one string that isn't the password. However, this information is not meaningful, in the sense that an efficient simulator cannot use it to learn the program's functionality on super-polynomially many inputs. (Note that learning information on polynomially many inputs is trivial since the simulator is allowed polynomially many queries.)

[9] Hyperplane membership testing has also been considered in the context of private predicate encryption schemes by Shen, Shi, and Waters [86], although our results are incomparable to theirs.

This primitive subsumes many of the previously-known results. In the $d = 2$ case, these "hyperplanes" are equivalent to point circuits, and our specific construction and assumption reduce to those in [23]. Furthermore, the technique of Canetti and Dakdouk [25] can be applied to our primitive as well, so we can obfuscate circuits that output a hidden message when its input is on the hyperplane (rather than simply a yes or no response).

Construction

The construction is as follows. Let G be a group of order p that satisfies a strengthened version of the Decisional Diffie-Hellman assumption, which we describe in more detail below. When the obfuscator is given a hyperplane $H_{\boldsymbol{a}}$ to obfuscate, where $\boldsymbol{a} = (a_1, \ldots, a_d)$, it chooses a random generator $g \xleftarrow{U} G$ and outputs g^{a_i} for all i. This allows the user to compute whether a given point $\boldsymbol{x} = (x_1, \ldots, x_d)$ is on the hyperplane by computing

$$(g^{a_1})^{x_1} \times \cdots \times (g^{a_d})^{x_d} = g^{\langle \boldsymbol{a}, \boldsymbol{x} \rangle},$$

and checking whether this equals G's identity element (that is, whether $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0$). This is a natural generalization of Canetti's obfuscator for point circuits described in Section 3.2.1 [23].

Our assumption

We are not able to prove the security of our construction based on the standard Decisional Diffie-Hellman assumption, which states that g^{ab} is indistinguishable from uniform, given g, g^a, and g^b for uniformly-chosen exponents a and b. We describe the difficulty with basing our scheme on DDH, as it motivates and clarifies our new assumption.

For our construction, it is crucial that the adversary not be able to compute *any* polynomial relationships in the exponent, not just quadratic ones. For example, given g, g^a, g^b, and g^c, is it possible to compute g^{abc}? How about g^{a^3}? Can elements of the form g^{abc} or g^{a^3} even be distinguished from uniform? No efficient algorithms for running these computations are known in groups where DDH is hard, but standard assumptions such as DDH do not appear to rule out the existence of such algorithms.

In general, we wish to characterize the polynomials ξ for which $g^{\xi(a,b,c)}$ is distinguishable from uniform. Clearly, this is true when ξ is linear, or closely resembles a line. Our new assumption states that these are the only such polynomials for which $g^{\xi(a,b,c)}$ can be distinguished from uniform.

We also consider the effect of choosing exponents from weak entropy distributions. This setting has been previously considered by Canetti [23], who forms a modified DDH assumption in which g^{ab} is considered to be indistinguishable from uniform given g, g^a, and g^b, where a is chosen from the uniform distribution but b is chosen from any distribution of super-logarithmic min-entropy.[10] Our assumption expands upon this idea and considers many exponents that are not only chosen from weak entropy distributions, but which may also be related.

Specifically, given a tuple of group elements $\langle g^{a_1}, g^{a_2}, \ldots, g^{a_d} \rangle$ where the a_i are chosen from some joint distribution, we ask for which polynomials ξ is $g^{\xi(a_1,\ldots,a_d)}$ still indistinguishable from uniform? If ξ is linear, or at ξ looks "close" to linear when restricted to the support of the joint distribution, then of course $g^{\xi(a_1,\ldots,a_d)}$ can be distinguished from uniform. Our new assumption states that indistinguishability holds in all other cases.[11]

[10] See Assumption 3.20 for a formal statement of Canetti's strong DDH assumption.

[11] Codifying what it means for ξ to look "close" to linear is rather delicate. To illustrate, consider the polynomial $\xi(a_1, \ldots, a_d) = a_1^p + \cdots + a_d^p$. This is a very high degree polynomial since $p \approx 2^n$, but by Fermat's little theorem it is equivalent to the linear function $a_1 + \cdots + a_d$, so our codification would have to identify ξ as close to linear.

This new assumption is stronger than the standard DDH assumption, or even the modified DDH assumption of [23], but we provide evidence of its feasibility by proving that it holds in the generic group model. Moreover, we believe that resolving the status of this new assumption would be interesting either way. If it holds, then we obtain an obfuscator for a new and interesting family of programs. Showing that the assumption does not hold would shed new light on which computations can be run efficiently in the exponent of DDH groups.

Application to digital signatures

As an example of the proposed primitive's usefulness, we demonstrate an application of our obfuscator to leakage-resilient one-time signatures. *Digital signatures* allow a user to sign messages in such a way that others can verify signatures but are unable to forge them [45, 61, 79]. We emphasize, though, that the main motivation for this work is the new obfuscator, rather than any single application.

The signature scheme is constructed as follows: the secret key is a randomly chosen plane in 3-dimensional space, and the public key is the obfuscated membership program. To sign a message m, find a point on the plane that is related to m. This signature can be verified by running the public obfuscated program.

This signature scheme remains unforgeable after receiving one message-signature pair. However, the scheme is trivially forgeable after receiving two pairs, because once an adversary knows two points on a plane, then she knows that the entire line connecting them is also on the plane.

We can only prove that the scheme satisfies a weaker form of the unforgeability game where the adversary must choose the message m to be signed before receiving the public key. Techniques from [56] allow us to transform the weak scheme into an ordinary one-time signature scheme. Additionally, this one-time signature scheme remains unforgeable even when a function of the secret key is leaked whose output length is up to half as long as the secret key. For schemes that do not use general zero-knowledge proofs, this matches the leakage bound of [58] (albeit under much stronger assumptions).

1.4 Organization

Chapter 2 gives a rigorous definition of obfuscation, and also defines the circuit families that are used throughout the thesis. Building upon this, Chapter 3 goes into far more detail on the history of program obfuscation as studied in theoretical cryptography. It is intended to be a thorough, self-contained survey on the state of the field, except for the new results presented in this work. However, it may be safely skipped by the uninterested reader.

The three main results of this thesis are described in detail in Chapters 4 through 6. The results are self-contained, so any chapter can be read without reading the others. Finally, Chapter 7 recaps the state of the field and describes several open problems.

At the end of the thesis, there is an exhaustive List of Symbols that defines every variable or symbol used in this work and provides references to more detailed explanations. There are also Lists of Algorithms and Figures.

Our actual assumption has a distributional flavor instead, while capturing the intuitive ideas discussed here. See Assumption 6.4 for details.

Chapter 2

Definitions

In this chapter, we formalize many of the concepts that were only described at a high level in Chapter 1. Specifically, we define circuit families and describe three important families that are of interest throughout this thesis. Then, we rigorously define program obfuscation, discussing some of the technicalities involved in the functionality and security guarantees.

2.1 Circuit families

Recall from Section 1.2.3 that it is impossible to build an obfuscator that simultaneously garbles every program. Instead, we can only build restricted obfuscators that are focused on a "family" of circuits. The next definition codifies this notion.

Definition 2.1. A *family of polynomial-size circuits* is an infinite sequence of sets $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where $\mathcal{C}_n$ is a set of circuits with input length n. Furthermore, there exists a polynomial s such that for every $n \in \mathbb{N}$ and every circuit $C \in \mathcal{C}_n$, the size of circuit C is at most $s(n)$. Although we do not explicitly bound the output length of circuits in $\mathcal{C}_n$, the size bound means that the output length is also polynomial in n.

There are three families of circuits that are of special interest in this thesis. The family of *point circuits* contains all circuits that accept a single input string and reject all other inputs. This family can be generalized in two different ways: first, *multi-point circuits* store a (polynomial-sized) list of strings to accept, and second, *point circuits with multi-bit output* reveal a hidden message upon receiving the accepted string. We describe these families in detail.

Point circuits

Given a string $w \in \{0,1\}^n$, let I_w be the circuit that stores w in some canonical, explicit manner, and tests whether its input is equal to w. Hence, I_w has the functionality

$$I_w(x) = \begin{cases} 1 & \text{if } x = w, \\ 0 & \text{otherwise.} \end{cases}$$

We form the family of all such circuits. For technical reasons, we also may wish to consider the circuit $I_\emptyset$ that rejects all inputs.[1]

[1] Technically speaking, there are many different circuits that reject all inputs, one for each input length. Formally, let $I_{\emptyset,n}$ be the circuit with n input bits that rejects all 2^n input strings, and whose description reveals so. We abuse notation and simply write $I_\emptyset$ to denote all of these circuits since the input length is usually clear from context.

For a given $n \in \mathbb{N}$,

$$\mathcal{P}^1_n = \{I_w : w \in \{0,1\}^n\} \cup \{I_\emptyset\}$$

is the set of all point circuits with input length n. We combine all of these sets into the *family of point circuits* $\mathcal{P}^1 = \{\mathcal{P}^1_n\}_{n\in\mathbb{N}}$. Also, let $\mathcal{P}^{1+}$ be the subfamily that does not include $I_\emptyset$.

Multi-point circuits

The above concept can be generalized to circuits that accept (polynomially) many input strings. Given m strings $w_1, w_2, \ldots, w_m \in \{0,1\}^n$, let $I_{\{w_1,\ldots,w_m\}}$ be the circuit that stores the list of strings $\{w_1, \ldots, w_m\}$ in a readily identifiable manner, and tests whether its input string x is in the list. It has the functionality

$$I_{\{w_1,\ldots,w_m\}}(x) = \begin{cases} 1 & \text{if } x \in \{w_1, \ldots, w_m\}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the w_i need not be distinct, so the circuit $I_{\{w_1,\ldots,w_m\}}$ accepts at most m inputs. Let

$$\mathcal{P}^m_n = \{I_{\{w_1,\ldots,w_m\}} : w_1, \ldots, w_m \in \{0,1\}^n\} \cup \{I_\emptyset\}$$

be the set of all circuits that accept at most m inputs, and form the *family of m-point circuits* $\mathcal{P}^m = \{\mathcal{P}^m_n\}_{n\in\mathbb{N}}$. Also, let $\mathcal{P}^{m+}$ be the subfamily that does not include $I_\emptyset$, so circuits in $\mathcal{P}^{m+}$ accept between 1 and m inputs.

Point circuits with multi-bit output

In this generalization of point circuits, we boost the output power of the circuit. Rather than simply giving a yes or no response, these circuits store a hidden string that is revealed upon receiving an accepted input.

Specifically, let $I_{(k,m)}$ be the circuit that stores a key k and message m in a readily identifiable manner and computes

$$I_{(k,m)}(x) = \begin{cases} m & \text{if } x = k, \\ \perp, 1^{|m|} & \text{otherwise.} \end{cases}$$

Note that the circuit does not hide the length of the message, since it is revealed for all inputs. Given a polynomial ρ and an input length n, let

$$\mathcal{I}_{n,\rho} = \{I_{(k,m)} : k \in \{0,1\}^n, m \in \{0,1\}^{\rho(n)}\}$$

and form the *family of point circuits with multi-bit output* $\mathcal{I}$ as the collection of the sets $\mathcal{I}_{n,\rho}$ for all $n \in \mathbb{N}$ and all polynomials ρ.

Uses in obfuscation

Looking ahead, there are many obfuscators for the family of point circuits [23, 39, 93] under a variety of cryptographic assumptions (see Section 3.2.1 for details). Some of the constructions require that a point is accepted, so they operate over $\mathcal{P}^{1+}$, whereas others allow the circuit that rejects all inputs so they operate over $\mathcal{P}^1$.

Remember that an obfuscator receives a circuit in $\mathcal{P}^1$ as input, so its operation does not just depend on the functionality of $\mathcal{P}^1$ but also on the description of its circuitry. For this reason, we stress that circuits in $\mathcal{P}^1$ store the string w in a readily identifiable manner so the obfuscator can find it. The same is true of the other families.

All of the obfuscators for point circuits can be extended to allow multiple accepted strings or multi-bit output [12, 25, 63, 93]. In fact, sometimes the two generalizations of point circuits can be combined to form "multi-point circuits with multi-bit output" that store a list of keys and reveal the message upon receiving any key in the list. We rarely consider this functionality, although Section 6.5 examines a related one.

Entropy

In this thesis, we often wish to sample a circuit from a given family $\mathcal{C}$. Formally, this requires a *distribution ensemble* $\mathcal{X} = \{\mathcal{X}_n\}$, where for each n, $\mathcal{X}_n$ is a distribution over $\mathcal{C}_n$. We typically restrict our attention to distribution ensembles for which it is hard to guess which circuit has been sampled.

Definition 2.2. A distributional ensemble $\mathcal{X}$ over the family $\mathcal{C}$ is *well-spread* if for every polynomial ρ, all sufficiently large $n \in \mathbb{N}$, and every $C \in \mathcal{C}$, $\Pr[\mathcal{X}_n = C] < \frac{1}{\rho(n)}$. Equivalently, the *min-entropy*

$$H_\infty(\mathcal{X}_n) = -\log(\max_{C \in \mathcal{C}_n} \Pr[\mathcal{X}_n = C])$$

is a super-logarithmic function of n.

2.2 Obfuscation

In this section, we codify the Barak *et al.* definition of virtual black-box obfuscation [10] including an extension to auxiliary input by Goldwasser and Kalai [43].

Informally, an obfuscator $\mathcal{O}$ is a compiler that receives a program from a deterministic family of circuits $\mathcal{C}$ as input and outputs another program. The obfuscator has two distinct requirements: it must preserve the functionality of programs it receives, and it must garble the code of the program to hide everything except its input-output behavior.

There are many technical decisions to be made when formalizing the functionality and security guarantees. First of all, in Chapter 1 we discussed obfuscation of computer programs. A computer program can be formally represented as either a Turing machine or a circuit. Barak *et al.* consider both concepts, but in this thesis we focus on the latter. In many situations, the idea of circuit families encompasses that of Turing machines, although this turns out not to be true for obfuscation (see Section 3.3.1).

In the rest of this section, we examine more technical subtleties and eventually produce a definition of obfuscation.

2.2.1 Functionality

We consider an obfuscator $\mathcal{O}$ that receives as input some circuit C from a given family $\mathcal{C}$. The obfuscator produces a garbled circuit $\mathcal{O}(C)$ that may look very different from C, but it should have a "similar" input-output behavior. Ideally, we want the obfuscated circuit to be equivalent to the original one in the following sense.

Definition 2.3. We say that two deterministic circuits C_1 and C_2 are *functionally equivalent*, and write $C_1 \equiv C_2$, if for all inputs x it holds that $C_1(x) = C_2(x)$.

Intuitively, obfuscation should ensure that $\mathcal{O}(C) \equiv C$. However, this statement does not quite make sense, because functional equivalence is defined for deterministic circuits but $\mathcal{O}$ is a probabilistic algorithm so it may produce different circuits for different choices of its randomness. Thus,

$\mathcal{O}(C)$ is actually a distribution over many different circuits. There are several ways to extend the concept of functional equivalence to this setting.

Exact functionality: $\mathcal{O}(C)$ is functionally equivalent to C for all settings of $\mathcal{O}$'s random tape.

Almost exact functionality: $\mathcal{O}(C)$ is functionally equivalent to C most of the time. Specifically, $\mathcal{O}(C; r) \equiv C$ with overwhelming probability over the choice of the random tape r.

These two guarantees are rather strong because they make global statements about the behavior of the circuits on *all* inputs. Since nobody will query a circuit on every possible input anyway, this requirement seems unnecessarily strong. We could simply require that $\mathcal{O}(C)$ and C agree on most inputs, but not all of them, and hope that nobody finds the "bad" input values.

Sadly, this assumption is unreasonable. Consider the family of point circuits $\mathcal{P}$[1]. Every circuit of the form I_w is "close," in the sense described above, to the circuit $I_\emptyset$ that rejects all inputs; in fact, I_w and $I_\emptyset$ disagree on just one input. But this special input value w is very important! For obfuscation to have practical applications like building a secure login program, it is essential that an obfuscation of I_w accept the string w.

The moral of this story is that the functionality guarantee should hold for every input. However, there is no need for the guarantee to hold for every choice of the randomness. This yields a new functionality guarantee.

Approximate functionality: For every input x, the outputs of $\mathcal{O}(C)$ and C are equal with overwhelming probability over $\mathcal{O}$'s randomness.

The new guarantee differs from almost exact functionality in the way that the equivalence property may fail. Almost exact functionality says that most of the time, $\mathcal{O}(C)$ produces a circuit that behaves exactly like C, although some of the time it may fail. When $\mathcal{O}$ fails though, there is no guarantee on its behavior; it may look arbitrarily different than C. Intuitively, the failure of $\mathcal{O}$ is concentrated on a few bad circuits.

By contrast, approximate functionality allows for a "diffusion" of the errors. Under this guarantee, it might be the case that $\mathcal{O}(C)$ is *never* functionally equivalent to C! That is, for every setting of the random tape, there might be some input value on which $\mathcal{O}(C)$ and C differ. However, these "bad inputs" change for different choices of the random tape, so there is no single input value that is consistently bad.

All three of these definitions have their uses, so we make the weakest guarantee[2] in the definition of obfuscation but point out when stronger guarantees actually hold.

2.2.2 Virtual black-box property

Next, we codify the security guarantee provided by Barak *et al.* called the *virtual black-box property*. Remember that the goal of obfuscation is to produce "garbled" circuits. As a baseline, no matter how garbled the circuit is, one can always run it and observe its input-output behavior. In this way, the garbled circuit must be at least as useful as an oracle to the circuit.

The virtual black-box property ensures the converse: that the code of an obfuscated program is *no more* useful than an oracle. This property is codified using a simulator-based definition. Consider an adversary A that is given an obfuscated circuit $\mathcal{O}(C)$ and attempts to learn a binary[3]

[2]Note that the extensions to the virtual black-box definition in Chapter 5 are only well-defined for the stronger notion of almost exact functionality. See Section 5.2 for details.

[3]The definition becomes unachievable if we do not impose any restrictions on the predicate. See Section 1.2.2 for an explanation, Theorem 3.4 for a formal statement, and Chapter 5 for a definition that considers more predicates.

predicate about the underlying circuit C. The virtual black-box property states that there must exist a simulator S that only interacts with an oracle to C but can still can learn the same predicate that the adversary learns.

When codifying this idea, there is one technical issue to deal with concerning how well the simulator must emulate the adversary.

Stronger requirement: For every adversary A, require the existence of a simulator S that can emulate A to within negligible error.

Weaker requirement: For every adversary A and polynomial ρ, require the existence of a simulator S that can emulate A to within $\frac{1}{\rho}$ error.

The second definition is weaker in two ways. One is obvious: S emulates A to a lower degree of accuracy. The second is more subtle but equally as important: the running time of S is allowed to depend on the choice of the polynomial ρ.

On the other hand, the weaker definition is easier to satisfy. In fact, all of the known results concerning obfuscation have been found with respect to the weaker definition, and the new results in this work use the weaker definition as well. As a result, we only codify the weaker notion here.[4]

With this technical issue resolved, we can now formalize obfuscation using our functionality and security guarantees. For efficiency reasons, there is actually a third requirement that the obfuscator should not "blow up" the size of a circuit too much when garbling it.

Definition 2.4 (Obfuscation [10, 43])**.** Let $\mathcal{C}$ be a family of polynomial-size circuits. A probabilistic polynomial time (PPT) algorithm $\mathcal{O}$ is an *obfuscator for $\mathcal{C}$ with dependent auxiliary input* if the following three conditions are met.

1. *Approximate functionality*: There exists a negligible function ε such that for every $n \in \mathbb{N}$, every circuit $C \in \mathcal{C}_n$, and every $x \in \{0,1\}^n$,

$$\Pr\left[\mathcal{O}(C;r)(x) = C(x)\right] > 1 - \varepsilon(n),$$

   where the probability is taken over the randomness r. *Almost exact functionality* is a stronger condition that requires $\mathcal{O}(C;r) \equiv C$ with overwhelming probability over the random coin tosses r. Finally, if this probability always equals 1, then $\mathcal{O}$ has *exact functionality*.

2. *Polynomial slowdown*: There exists a polynomial ξ such that for every n, every circuit $C \in \mathcal{C}_n$, and every possible sequence of coin tosses r, the circuit $\mathcal{O}(C;r)$ runs in time at most $\xi(n)$.

3. *Virtual black-box*: For every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, for all auxiliary inputs $z \in \{0,1\}^*$, and for all binary predicates π,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = \pi(C, z)\right] - \Pr\left[S^C(1^n, z) = \pi(C, z)\right]\right| < \frac{1}{\rho(n)},$$

   where the first probability is taken over the coin tosses of A and $\mathcal{O}$, and the second probability is taken over the coin tosses of S. Furthermore, we require that the runtime of A and S is polynomial in the length of their first input.

We define (ordinary) *obfuscation*, without auxiliary input, in the same manner except that z is removed from the virtual black-box definition.

[4]The relationship between the two requirements is explored further in Chapter 3 and Theorem 4.2.

We make several remarks on this definition.

1. In the virtual black-box property, Barak *et al.* [10] show that it suffices to consider only the predicate π that always equals 1. We make this simplification whenever it is useful.

2. There are two distinct efficiency guarantees. First, the garbling procedure is efficient, and second, the garbled circuit is efficient.

3. Throughout this thesis, adversaries and simulators are assumed to be non-uniform unless otherwise stated. In fact, Hoeteck Wee [93] proves that non-uniform simulators are essential for some constructions. See Theorem 3.2 for details.

4. We only consider obfuscation with dependent auxiliary input in portions of Chapter 3, Section 4.4, and Chapter 5. The rest of this thesis uses the ordinary definition of obfuscation (without auxiliary input).

5. Instead of passing the auxiliary input as a string, we could equivalently consider it to be a function of the circuit C.

6. Goldwasser and Kalai [43] define another notion of auxiliary input called *independent auxiliary input*, which we explain in Definition 3.12.

7. Some obfuscators operate in the random oracle or common reference string (CRS) models. In these models, all algorithms (A, S, and $\mathcal{O}$) are given access to the random oracle or CRS. See Section 5.2.4 for more details.

8. This definition does not, in general, provide a composability guarantee [12, 25, 63].

We consider the last remark in more detail. Suppose we "concatenate" several obfuscated circuits as shown in Figure 1-4. Ideally, the resulting circuit would also be an obfuscator in the following sense.

Definition 2.5 (Composable obfuscation by concatenation [12, 25, 63])**.** A PPT algorithm $\mathcal{O}$ is a *t-composable obfuscator* for the family $\mathcal{C}$ with dependent auxiliary input if functionality and polynomial slowdown hold as before, and the virtual black-box property holds whenever the adversary and simulator are given up to t circuits in $\mathcal{C}$. Formally, for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, and for all $C_1, \ldots, C_t \in \mathcal{C}_n$, and for all auxiliary inputs $z \in \{0,1\}^*$,

$$\left|\Pr\left[A(\mathcal{O}(C_1), \ldots, \mathcal{O}(C_t), z) = 1\right] - \Pr\left[S^{C_1,\ldots,C_t}(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

where the probabilities are taken over the random coin tosses of A, S, and $\mathcal{O}$. Furthermore, we require that the runtime of A and S is polynomial in the length of their first input.

We define t-composable obfuscation without auxiliary input in the same manner except that z is removed from the definition. Additionally, we define t-self-composable obfuscation in the same manner except that $C_1 = \cdots = C_t$.

Unfortunately, the concatenation of obfuscators does not always result in a secure obfuscator. In fact, in the random oracle model there is an obfuscator that is not even 2-composable [63], and in the standard model there is a construction that is not $\omega(\log(n))$-composable [25]. We explore this issue, along with many other technical features and shortcomings of this definition, in the next chapter.

Chapter 3

Survey of Prior Works

In this chapter, we pick up where Section 1.2 and Chapter 2 left off and recap the state of program obfuscation, as it has been studied in theoretical cryptography. The goal of this chapter is *not* to provide an encyclopedic list of the known results and their proofs, but rather to provide a broad overview of the various facets of the problem. The proofs are only sketched or are left out entirely.

First, we describe several definitions of obfuscation in the literature, explaining the advantages of each. Then, we positive results found under the various definitions. Finally, we study the definitions on a theoretical level, comparing their strength and examining what is (and is not) possible under each definition.

Note that there is no new material presented in this chapter, although there are occasional references to later chapters. This chapter may be safely skipped by the uninterested reader.

3.1 Definitions

In Chapter 2, we describe one definition of obfuscation [10, 43]. It contains three separate requirements that provide functionality, efficiency, and security guarantees. The functionality and efficiency properties are relatively straightforward, and widely adopted.

The same cannot be said for the security guarantee, however. In fact, there are at least *ten* different security guarantees in the literature that codify the intuitive concept of "garbling" a circuit to be unintelligible. Furthermore, most of these definitions can be tweaked to provide:

- Strong or weak simulation, as described in Section 2.2.
- Composability under concatenation, as described in Definition 2.5.
- Dependent, independent, or no auxiliary input.

The result is a rather chaotic "zoo" of definitions with differing levels of strength and applicability. Most of the definitions are shown in Figure 3-1. One of the major goals of this chapter is to see how the security guarantees relate. In this section, we define most of the security guarantees and describe the "simple" connections between them. More complicated relationships are deferred to Section 3.3.

3.1.1 Virtual black-box

The first security guarantee in the literature (and the one used in subsequent chapters of this work) was provided by Barak *et al.* [10]. Their requirement, called the *virtual black-box property*, states

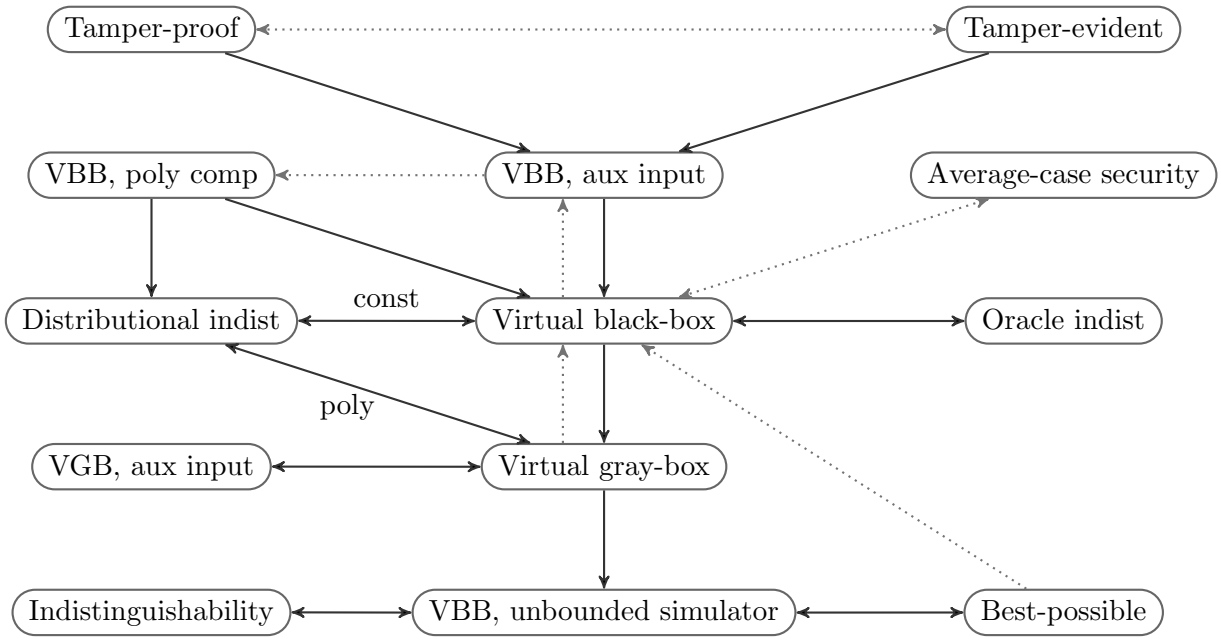

Figure 3-1: Comparison of several definitions of obfuscation. Unless otherwise noted, these definitions do not consider auxiliary input or composability. Solid lines indicate implications, with labels indicating the maximum level of composability under which the implication is known to hold. Dotted lines indicate separations under reasonable cryptographic assumptions.

that it is hard to learn anything from an obfuscated circuit that isn't immediately obvious from its input-output behavior. This idea is codified using a simulator-based definition, which says that for every non-black-box adversary that learns a predicate about an obfuscated circuit, there exists a black-box simulator that can learn the same predicate.

Barak *et al.* [10] actually provide two such definitions, one for Turing machines and one for circuits. In this section, we focus on the circuit-based definition. The Turing machine definition is discussed in Section 3.3.1.

As described in Section 2.2.2, there is one degree of freedom in codifying this property. Either there should be one simulator that emulates the adversary to within a negligible error, or there should be a sequence of simulators that emulate the adversary with smaller error terms (but at the expense of increasing the running time of the simulator). The weaker requirement was already formalized in 2.4, so it remains to define the stronger requirement.

Definition 3.1 (Strong[1] virtual black-box property)**.** For every PPT adversary A, there exists a negligible function ε and a PPT simulator S such that for all $n \in \mathbb{N}$ and $C \in \mathcal{C}_n$,

$$\left|\Pr\left[A(\mathcal{O}(C)) = 1\right] - \Pr\left[S^C(1^n) = 1\right]\right| < \varepsilon(n),$$

where the first probability is taken over the coin tosses of A and $\mathcal{O}$, and the second probability is taken over the coin tosses of S.

[1]Historically, this was the original definition of Barak *et al.* [10]. As a result, in many works, the term "virtual black-box property" is used to denote this property, whereas the property in Definition 2.4 is explicitly called the "*weak* virtual black-box property." Because all of the known constructions use the weaker variant, we take the opposite view, and default to the weaker definition unless the adjective *strong* is explicitly mentioned.

A quick note on jargon: we differentiate here between security properties and definitions of obfuscation. For instance, the "virtual black-box property" is one of three requirements in the definition of "virtual black-box obfuscation," along with functionality and efficiency requirements. Many of the other definitions in the literature use the same functionality and efficiency requirements as the one in Definition 2.4, so for the sake of brevity, we do not repeat them here. Instead, we define new security properties with the implicit understanding that each one leads to a new definition of obfuscation. For example, combining Definition 3.1 with the functionality and polynomial slowdown requirements yields a "strong virtual black-box obfuscator" $\mathcal{O}$ for family $\mathcal{C}$.

Analysis of the virtual black-box property

We make a few technical observations about the definition. Recall that we assume that adversaries and simulators are non-uniform throughout this work. This convention seems crucial to achieving virtual black-box obfuscation for certain families.

Specifically, virtual black-box obfuscation is not possible for the family of point circuits $\mathcal{P}^1$ with a simulator that is both uniform and only uses the adversary in a "black-box manner," meaning that S is allowed to emulate executions of A but cannot use more sophisticated aspects of the adversary's behavior.

Theorem 3.2. *Uniformly black-box obfuscators for the family $\mathcal{P}^1$ do not exist* [93, Prop 5.1].

While not all of the proofs of security for known constructions use a black-box simulator [23], some of them do [63, 93], and thus non-uniform simulators are required for these constructions.

One useful metric that we will use to compare the various security definitions is to consider the circuits that are *trivially obfuscatable* under each requirement, in the sense that they technically satisfy the definition but do not seem to provide any reasonable security goal. As described in Section 1.2.4, the "trivially obfuscatable" circuits for the (strong or weak) virtual black-box property are learnable ones. This concept was formalized in [63].

Definition 3.3. A family of circuits $\mathcal{C}$ is called *exactly learnable* if there exists a (possibly non-uniform) PPT oracle circuit L such that for all $C \in \mathcal{C}$, L^C outputs a polynomial sized circuit D that is functionally equivalent to C.

It is simple to construct an obfuscator $\mathcal{O}$ for a learnable family $\mathcal{C}$. The obfuscator receives a circuit $C \in \mathcal{C}$, runs L^C to produce a circuit D, and outputs D. This is an obfuscation because any simulator S^C can obtain D as well and thus emulate the adversary perfectly. However, this construction is uninteresting because it does not attempt to provide any security.

At this stage, we can also formalize the statement in Section 1.2.2 that obfuscation becomes unachievable if we do not restrict the output length of A and S.

Theorem 3.4. *A family of circuits $\mathcal{C}$ is obfuscatable against general adversaries (whose output length is unrestricted) if and only if $\mathcal{C}$ is exactly learnable* [93, Proposition 5.2].

Proof sketch. For learnable families, the simulator described above suffices even against general adversaries. Conversely, if $\mathcal{C}$ is obfuscatable against general adversaries, the simulator S for the "dummy adversary" that outputs its input can be used to construct a learner for $\mathcal{C}$. □

This theorem justifies our use of binary predicates in the virtual black-box property. We follow this convention in most of the other security properties.

Virtual gray-box

This definition weakens the virtual black-box property by giving the simulator more power. Specifically, the simulator is allowed unlimited running time, with the constraint that it must still make at most polynomially many queries to its oracle. We formalize this idea for the weaker simulation requirement; the stronger variant follows analogously.

Definition 3.5 (Virtual gray-box property [12])**.** For every PPT adversary A and polynomial ρ, there exists a (possibly inefficient) simulator S and a polynomial q such that for all sufficiently large n and any $C \in \mathcal{C}_n$,

$$\left|\Pr\left[A(\mathcal{O}(C)) = 1\right] - \Pr\left[S^C(1^n) = 1\right]\right| < \frac{1}{\rho(n)},$$

where S is limited to $q(n)$ oracle queries.

Because this definition is weaker than the virtual black-box property, it allows more "trivially obfuscatable" circuit families: any family that is exactly learnable by a simulator with unbounded running time but only polynomially-many queries. This includes many cryptographic functionalities. On the other hand, there are many families that do not have this property, such as point circuits and their extensions, for which the virtual gray-box property is meaningful.

Best-possible obfuscation

The virtual gray-box definition can be weakened further by removing the restraint on the number of oracle queries. We call this notion the "virtual black-box property with unbounded simulators."

Somewhat surprisingly, this definition turns out to be equivalent to two other definitions in the literature that are not simulator based at all! They approach the question of determining when a circuit is "sufficiently garbled" in a completely different way.

First, the *indistinguishability property* says that when $\mathcal{O}$ is given functionally equivalent circuits, it must produce indistinguishable outputs.

Definition 3.6 (Indistinguishability [10])**.** For any pair of circuits $C_1, C_2 \in \mathcal{C}$ that are functionally equivalent and have the same size, the distributions $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ are computationally indistinguishable. These distributions are taken over the randomness of $\mathcal{O}$.

Computational indistinguishability is a commonly-used idea that we will encounter much more later in this chapter, so we take the time to define it rigorously here.

Definition 3.7 (Computational indistinguishability)**.** Let $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n\in\mathbb{N}}$ be two ensembles of probability distributions. We say that $\mathcal{X}$ and $\mathcal{Y}$ are *computationally indistinguishable*, and write $\mathcal{X} \approx_c \mathcal{Y}$, if for any PPT distinguisher D, the difference

$$\left|\Pr\left[x \leftarrow \mathcal{X}_n : D(x) = 1\right] - \Pr\left[y \leftarrow \mathcal{Y}_n : D(y) = 1\right]\right|$$

is a negligible function of n.

A related definition, provided by Goldwasser and Rothblum [46], has an obfuscator choose the *most garbled* version of its input circuit. Specifically, the obfuscation of a circuit C should be some circuit D with the property that anything that can be learned from D can also be learned from any circuit that is functionally equivalent to C. In other words, D is the most unintelligible circuit that is functionally equivalent to C. A formal definition follows.

Definition 3.8 (Best-possible [46])**.** For every PPT learner L, there exists a simulator S such that for sufficiently large n and for any pair of circuits $C_1, C_2 \in \mathcal{C}$ that are functionally equivalent and have the same size, the distributions $L(\mathcal{O}(C_1))$ and $S(C_2)$ are computationally indistinguishable.

Note that the output length of the learner is not restricted. In fact, it suffices to consider the "dummy learner" that simply outputs its input.

At a first glance, the (weak) virtual black-box property with unbounded simulators seems to be a substantial weakening of the original Barak *et al.* definition, the indistinguishability property does not appear to make any security requirement at all, and the best-possible guarantee seems very powerful. As a result, the next theorem is rather counter-intuitive.

Theorem 3.9. *Virtual black-box obfuscation with unbounded simulators, indistinguishable obfuscation, and best-possible obfuscation are equivalent.*

The equivalence between the first two definitions is found in [12, Proposition 3.1], and the equivalence between the last two definitions is found in [46, Propositions 3.4 and 3.5].

3.1.2 Auxiliary input and composability

In the prior section, the virtual black-box property was weakened by giving the simulator extra power. In this section, we strengthen the virtual black-box property by giving benefits to the adversary.

Auxiliary input

In the real world, an adversary does not always attack a cryptographic scheme "in a vacuum." Often, the adversary possesses extra information that aids in her attack. Goldwasser and Kalai [43] extend the definition of obfuscation to reflect this by adding *auxiliary input*. Informally, their extension states that while auxiliary input may allow the adversary to learn more features of an obfuscated circuit, it still does not help her to do so in a non-black-box manner.

In fact, Goldwasser and Kalai form two definitions of obfuscation with respect to *dependent* and *independent* auxiliary input. The main difference between the two is that dependent auxiliary information can depend on the circuit being obfuscated, whereas independent auxiliary input is only allowed to depend on the circuit family being obfuscated (and not the precise circuit chosen).

The virtual black-box property with respect to dependent auxiliary input was already given in Definition 2.4, but we replicate it here for clarity.

Definition 3.10. For every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary inputs $z \in \{0,1\}^*$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

where the probabilities are taken over the randomness of A, S, and $\mathcal{O}$. Furthermore, we require that the runtime of A and S is polynomial in the length of their first input.

Similarly, we can add dependent auxiliary input to the strong virtual black-box property and the (strong or weak) virtual gray-box property. The latter is unnecessary, however.

Theorem 3.11. *Every virtual gray-box obfuscator is also an obfuscator w.r.t. dependent auxiliary input, whether both definitions use the strong or weak simulation requirement* [12, Proposition A.3].

Proof sketch. The simulator has unbounded running time, so it can compute the best possible auxiliary input on its own. □

Independent auxiliary input is weaker in two ways: first, we eliminate the "for all" property on circuits and consider C to be chosen randomly from $\mathcal{C}_n$, and second, the auxiliary input z must be independent of this choice.

Definition 3.12 (Virtual black-box property w.r.t. independent auxiliary input [43])**.** For every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for every polynomial ξ, for all sufficiently large n, and for all auxiliary inputs $z \in \{0,1\}^{\xi(n)}$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

where the probabilities are taken over the choice of $C \xleftarrow{U} \mathcal{C}_n$ and the randomness of A, S, and $\mathcal{O}$.

Removing the "for all" property from the definition weakens it considerably, to the extent that it is weaker than the original virtual black-box property.

Theorem 3.13. *Every virtual black-box obfuscator is also an obfuscator w.r.t. independent auxiliary input, whether both definitions use the strong or weak simulation requirement* [43, Theorem 14]*.*

As a result, we do not consider independent auxiliary input later in this thesis.

Composability

Another way to strengthen the definition is to require that it hold even when the adversary receives *many* obfuscated programs and the simulator receives many oracles. This scenario is referred to as "composability by concatenation." There are two forms of composability, depending on whether the adversary receives many obfuscations of the same circuit or of different circuits. An obfuscator that can withstand the first notion is called *self-composable*, and one that can withstand the second notion is called *fully composable*. These two notions are codified in Definition 2.5.

Composability is not, in general, implied by obfuscation alone, so it must be considered on a case-by-case basis for each construction. Doing so is worth the effort because a composable obfuscator typically allows the construction of an obfuscator for a stronger primitive. For instance, we will later show constructions for:

- An obfuscator for point circuits with multi-bit output $\mathcal{I}$, based on a sufficiently composable obfuscator for point circuits $\mathcal{P}^1$ (see Section 3.2.1)
- A similar extension for the family of hyperplane membership testing circuits (see Section 6.5)
- A CPA secure encryption scheme that tolerates weak keys, based on a sufficiently self-composable obfuscator for $\mathcal{I}$ (see Section 4.3.2)

Non-malleability

The virtual black-box definition can be strengthened in yet another way to provide non-malleability guarantees. These definitions allow the adversary and simulator to output more than one bit, but only in a restricted manner to avoid the impossibility result in Theorem 3.4. Crafting the restrictions appropriately results in two definitions of non-malleable obfuscation, one that prevents an obfuscated circuit from being tampered, and another one that detects tampering after it occurs.

These definitions can be found in Section 5.2, and their relationship to each other and the virtual black-box property (as diagrammed in Figure 3-1) is shown in Theorem 5.10. We do not consider non-malleable obfuscation further in this chapter.

3.1.3 Average-case obfuscation

In this section, we consider a new security requirement called *average-case security* that is incomparable in strength to the virtual black-box property. On the one hand, average-case security is stronger in the sense that it allows for general adversaries and simulators with unlimited input length. On the other hand, average-case security is weaker in the sense that (as the name suggests) the security property is only required to hold for a *random* circuit in the family.

As a result, this definition is best-suited for cryptographic applications in which drawing a circuit at random is meaningful. In fact, Hohenberger *et al.* [55] show that this definition is better-suited to many cryptographic tasks than the virtual black-box one because it preserves most security properties in the following sense:

> "If a cryptographic scheme is secure when the adversary is given black-box access to a program, then it remains secure when the adversary is given the obfuscated program."

This guarantee comes at the price of only achieving security for a random instance of the problem.

In order to deal with cryptographic functionalities, this definition also allows the family of circuits $\mathcal{C}$ under consideration to be *probabilistic*, whereas the other definitions only consider deterministic circuits. This change necessitates a review of the functionality property. Note that (almost) exact functionality no longer makes sense in this setting because functional equivalence is only well-defined for deterministic circuits. Even approximate functionality no longer suffices in this setting.

Instead, we define a new functionality property that states "with overwhelming probability, $\mathcal{O}(C)$ behaves *almost* identically to C on all inputs" [46]. Many cryptographic applications only require this level of functionality. We codify the "almost" requirement using a statistical distance bound, but there are reasonable alternatives, and the precise decision does not affect the results we achieve [53].

Definition 3.14 (Average-case obfuscation [53, 55])**.** A PPT algorithm $\mathcal{O}$ is an *average-case secure obfuscator* for the family $\mathcal{C}$ with auxiliary input if the following three conditions are met.

1. *Functionality*: There exists a negligible function ε such that for all $n \in \mathbb{N}$ and $C \in \mathcal{C}_n$,

$$\Pr\left[\exists x \in \{0,1\}^n : \Delta(\mathcal{O}(C)(x), C(x)) \geq \varepsilon(n)\right] \leq \varepsilon(n),$$

   where Δ denotes the statistical difference between the two distributions. The probability holds over the randomness of $\mathcal{O}$.

2. *Polynomial slowdown*: Same as before.

3. *Average-case security*: For every PPT adversary A, there exists a negligible function ε and a PPT simulator S such that for every PPT distinguisher D, every $n \in \mathbb{N}$, and every auxiliary input $z \in \{0,1\}^*$,

$$\left|\Pr\left[D^C(A(\mathcal{O}(C), z), z) = 1\right] - \Pr\left[D^C(S^C(1^n, z), z) = 1\right]\right| < \varepsilon(n),$$

   where the probabilities are taken over the random choice of $C \xleftarrow{U} \mathcal{C}_n$ and the randomness of A, D, $\mathcal{O}$, and S.

We define (ordinary) average-case secure obfuscation, without auxiliary input, in the same manner except that z is removed from the average-case security property.

We conclude this section with a discussion on the meaningfulness of average-case security. Recall from Theorem 3.4 that virtual black-box obfuscation of deterministic families $\mathcal{C}$ against general adversaries is only possible for learnable families. The proof of the theorem carries over to average-case security, but the statement weakens to consider *approximately learnable* families. Specifically, we can only obfuscate $\mathcal{C}$ under the average-case definition if they have the following property: given polynomial time and oracle access to a circuit $C \xleftarrow{U} \mathcal{C}$, one can construct a circuit D that computes C except on a small, hard to find subset of the inputs.

This is a very restrictive property on families of deterministic circuits. We can obfuscate point circuits because they are approximately learnable by the circuit $I_\emptyset$ that rejects all inputs. However, most deterministic circuits (in particular, deterministic cryptographic primitives) violate this property.

Theorem 3.15. *It is impossible to obfuscate a pseudorandom function ensemble in the average-case sense* [53, Theorem 4.6].

This issue (largely) disappears when we consider probabilistic families.

3.2 Positive results

In this section, we describe the obfuscators that have been constructed in the literature under the virtual black-box and average-case security definitions.

3.2.1 Point circuits and generalizations

Under the virtual black-box definition, the only known constructions are for point circuits $\mathcal{P}^1$ and their generalizations to multiple inputs, multi-bit output, and hyperplane membership testing. We describe the constructions for the first three families here, and defer hyperplane membership testing to Chapter 6.

There are several constructions of obfuscators based on different cryptographic assumptions or models. Sadly, none of the constructions use "standard" cryptographic assumptions like the existence of one-way functions, or even number-theoretic assumptions like the hardness of factoring. We will see later that this caveat is inherent to the problem of virtual black-box obfuscation.

Example: Newspaper puzzles

Before constructing an obfuscator for the family of point circuits, it is reasonable to determine the value of such a construction. In Section 1.2.4, we described one application of such an obfuscator to the construction of secure login programs. Here, we provide another motivating example due to [23]. Suppose Bob publishes a puzzle in a newspaper, and he wants to attach a short string s that can be used to verify a solution x to the puzzle. However, s should not provide any benefit in solving the puzzle.

One possible solution is to use a cryptographic hash function (say, SHA-1), and have Bob publish the hash of the solution to the puzzle. Naïvely, this appears sufficient, but a careful analysis shows that it fails to provide the required security for two reasons:

1. Many cryptographic hash functions only give a security guarantee (against inversion or collision-resistance) when its input is chosen uniformly at random. However, the solution to the puzzle may have a lot of structure, in which case the security guarantee may not hold.

2. Even if the puzzle solution has little structure, using a deterministic hash function is insufficient because it always reveals *some* information about the solution. That is, for any deterministic function f, $f(x)$ itself is some information on x.

Hence, we desire a *randomized* hash function h that hides all information about its input x. More precisely, an adversary should be able to learn as much information about x with $h(x;r)$ as without it (where r denotes the randomness of h). This guarantee should hold whenever x is chosen from a distribution with sufficient min-entropy (in the sense of Definition 2.2). Functions of this type are called *perfectly one-way* (POW) functions, and are constructed in [23, 25, 28].[2]

Two new definitions

It turns out that this primitive is very similar to virtual black-box obfuscations of point circuits $\mathcal{P}^1$. At a first glance, it is difficult to see the connection, because the guarantee on POWs is distributional in nature, whereas the virtual black-box property must hold for all x. Canetti [23] shows that these definitions are in fact equivalent when viewing distributions of super-logarithmic min-entropy. Such distributions are called "well-spread," and are defined in Definition 2.2.

Definition 3.16 (Distributional indistinguishability [23])**.** For any PPT adversary A with binary output and for any well-spread distribution ensemble $\mathcal{X}$ in which $\mathcal{X}_n$ takes values in $\{0,1\}^n$,

$$\langle w, A(\mathcal{O}(I_w;r))\rangle \approx_c \langle w', A(\mathcal{O}(I_w;r))\rangle$$

where w and w' are two independent samples from $\mathcal{X}_n$ and r is the randomness of $\mathcal{O}$.

Hence, obfuscation of point circuits yields a perfectly one-way hash function as well, but *only* in the strongest setting in which the hash function must tolerate any well-spread distribution. The general definition of perfectly one-way hash functions, on the other hand, is more flexible and allows functions that can tolerate different min-entropy bounds. This is useful because POWs for weaker bounds can be constructed under standard assumptions. To avoid getting sidetracked from obfuscation here, we refer the reader to [23, 28] for more on perfectly one-way hash functions.

Returning to obfuscation, the proof that the virtual black-box property is equivalent to distributional indistinguishability goes through an intermediate definition.

Definition 3.17 (Oracle indistinguishability [23])**.** For any PPT adversary A and polynomial ρ, there exists a polynomial size family of sets $\{L_n\}_{n\in\mathbb{N}}$ such that for all sufficiently large n and for all $w, w' \notin L_n$,

$$|\Pr[A(\mathcal{O}(I_w)) = 1] - \Pr[A(\mathcal{O}(I_{w'})) = 1]| < \frac{1}{\rho(n)}.$$

The preceding two definitions are *incredibly* useful. On a theoretical level, they provide much more intuition about the security guarantee provided by obfuscation of point circuits. Suppose we use such an obfuscator to construct a secure login program, as described in Section 1.2.4. Then, distributional indistinguishability says that any adversary without the password is unable to break the login program as long as the password is chosen from a well-spread distribution. Oracle

[2] POWs are constructed in [23] from cryptographic hash functions, and in [28] from universal hashing or pseudorandom functions. We refer the reader to these works for a full treatment of the constructions.

indistinguishability says that the best attack that the adversary can hope to mount is a dictionary attack. Specifically, for any adversary A, there is some list of guesses such that A effectively queries the login program on everything in this list, and fails to learn anything about the real password as long as it is not on the list.

On a practical level, these definitions are often easier to work with than the virtual black-box property in security reductions. Most of the proofs of security for constructions in this chapter, as well as the rest of this thesis, go through these intermediate definitions (see Theorems 4.2, 4.14, 5.20, and 6.11).

Theorem 3.18. *The virtual black-box property for point circuits is equivalent to the distributional indistinguishability and oracle indistinguishability properties* [23, Theorem 4].

Proof. We outline the ideas behind the three implications.

OI ⇒ VBB: This is the easiest direction. Oracle indistinguishability says that for every adversary A, there is an associated list of passwords such that A doesn't do anything more clever than querying all of the passwords in this list. Then, we build a simulator that has this list of passwords hard-coded (by non-uniformity) and simply makes all the queries.

VBB ⇒ DI: Suppose that distributional indistinguishability is false, which means that there is an adversary A such that $\Pr[A(\mathcal{O}(I_w)) = 1]$ varies non-negligibly for different values of w. Suppose we "rank" all of the passwords by the likelihood that A outputs 1 when receiving an obfuscated circuit with this password. That is, let $\gamma(w) = \Pr[A(\mathcal{O}(I_w)) = 1]$ and consider an ordered ranking of strings w by their associated $\gamma(w)$. Then, A can distinguish super-polynomially many passwords at the "top" of this list from those at the "bottom." On the other hand, a simulator that only receives oracle access to a login program has no hope of doing so (it can only implement a dictionary attack).

DI ⇒ OI: Suppose oracle indistinguishability is false, so there exists an adversary A and polynomial ρ such that for every polynomial-sized list L_n, there always exist two passwords $w, w' \notin L_n$ that A can distinguish. Then, we construct a well-spread distribution of passwords iteratively, such that A can make a non-trivial statement about obfuscated circuits hiding these passwords. Start with $L_n = \emptyset$ and find two passwords that A can distinguish. Add these two passwords to L_n, and find another pair of passwords that A can distinguish. This process can be continued for a very long time to build a super-polynomial size set L_n of passwords that A can distinguish. (The set is not limited to polynomial size because for every potential bound ρ, oracle indistinguishability is still false so we can still add another pair of passwords to the set). As before, order the passwords $w \in L_n$ by $\gamma(w) = \Pr[A(\mathcal{O}(I_w)) = 1]$. Finally, the uniform distribution over the passwords in L_n is well-spread (because the set is super-polynomial in size), and A can distinguish passwords at the "top" of the list from those at the "bottom," using the same argument as before. □

Distributional indistinguishability and oracle indistinguishability can be extended to consider point circuits with multi-bit output $\mathcal{I}$. We do not do so here, but refer the interested reader to Section 4.2.3. Extending distributional indistinguishability to encompass composability is also straightforward.

Definition 3.19 (t-Distributional indistinguishability [12])**.** For any PPT adversary A with binary output and any well-spread distribution ensemble $\mathcal{X} = \{\mathcal{X}^1, \ldots, \mathcal{X}^t\}$ over t-tuples of points in $\{0,1\}^n$ such that the projection to each coordinate $\mathcal{X}^i$ is well-spread,

$$\langle \mathcal{O}(I_{w_1}), \ldots, \mathcal{O}(I_{w_t}) \rangle \approx_c \langle \mathcal{O}(I_{w'_1}), \ldots, \mathcal{O}(I_{w'_t}) \rangle$$

where the distributions depend on the choices of $(w_1, \ldots, w_t) \leftarrow \mathcal{X}$, $(w'_1, \ldots, w'_t) \xleftarrow{U} (\{0,1\}^n)^t$, and the randomness of $\mathcal{O}$.

Note that we have cheated here by changing the structure from Definition 3.16 in two ways:

- There is no longer an adversary in the definition. Effectively, there is a "dummy adversary" that feeds its input straight to the distinguisher.
- The distributions on the two sides of the indistinguishability guarantee have different distributions. Essentially, the definition now says that $\mathcal{X}$ is "indistinguishable" from the uniform distribution, when given obfuscated programs from these distributions.

It can be shown though that these changes do not affect the strength of the definition. In particular, for $t = 1$ this is equivalent to Definition 3.16 [12].

Adding a composability guarantee to oracle indistinguishability is significantly more complicated. We do not do so here, but note that Lemma 6.13 does so in a special setting.

The (g, g^w) construction

Armed with the two new definitions, we can now build an obfuscator for point circuits and prove its security. This construction is due to Canetti [23], and it requires a group G for which computing discrete logarithms is hard.

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

1: choose a generator $g \xleftarrow{U} G$ uniformly at random

2: compute $h \leftarrow g^w$, where w is viewed as an element of G in some canonical way

Output: circuit that has g, h hardwired, and on input a string x, accepts if $g^x = h$

Note that the circuit constructed by this algorithm accepts exactly one input value, so it belongs to the family $\mathcal{P}^{1+}$. The construction is remarkably simple (and efficient), but proving its security is much more difficult and requires that G satisfy a strong variant of the Decisional Diffie-Hellman assumption.

Assumption 3.20 (Strong DDH assumption)**.** *Let n be a security parameter and let $p = 2q + 1$ be a randomly chosen n-bit safe prime. Consider the group Q of quadratic residues in $\mathbb{F}_p^*$. For any well-spread distribution ensemble $\{X_q\}$ where the domain of X_q is $\mathbb{F}_q$, for $g \xleftarrow{U} Q$, $a \leftarrow X_q$, $b, c \xleftarrow{U} \mathbb{F}_q$, the ensembles $\langle g, g^a, g^b, g^{ab} \rangle$ and $\langle g, g^a, g^b, g^c \rangle$ are computationally indistinguishable.*

In this assumption, $\mathbb{F}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$ denotes the finite field of order p. Also, remember that a "well-spread ensemble" means that the min-entropy $H_\infty(X_q)$ is a super-logarithmic function of n (see Definition 2.2 for a formal explanation).

Theorem 3.21. *The construction presented above, when used with the group $G = Q$, is a virtual black-box obfuscator for the family of point circuits $\mathcal{P}^{1+}$ under Assumption 3.20* [23].

Proof sketch. It suffices to prove that the construction satisfies distributional indistinguishability. Suppose it does not. By the arguments in the proof of Theorem 3.18, there exists an adversary A such that $\gamma(w) = \Pr[A(\mathcal{O}(I_w)) = 1]$ varies non-trivially for super-polynomially many values of w.

Note that A effectively receives g and g^w as input, so $\gamma(w) = \Pr[A(g, g^w) = 1]$. In essence, A is able to distinguish some exponents from others. Note that our construction is "re-randomizable" in the sense that given (g, g^w), it is easy to form many more tuples of the form $(\bar{g}, \bar{g}^a)$ for $\bar{g} \xleftarrow{U} Q$.

Hence, $\gamma(w)$ can be estimated by running $A(g, g^w)$ many times with fresh randomness and using a Chernoff bound.

We use this idea to form an adversary $A'(g, g^a, g^b, g^c)$ that breaks the strong DDH assumption (i.e., it can distinguish whether $c = ab$ or c is chosen uniformly at random) when a is one of the exponents that A can distinguish. The adversary A' can run $A(g, g^a)$ and $A(g^b, g^c)$ many times to estimate $\gamma(a)$ and $\gamma(\frac{c}{b})$. If $c = ab$, then these estimates should be close in value, but if c is chosen uniformly at random, then $\gamma(\frac{c}{b})$ will be noticeably distinct from $\gamma(a)$ for a noticeable fraction of the choices of c. Thus, A' breaks the strong DDH assumption, as required. □

Extensions due to composability

If the above obfuscator $\mathcal{O}$ is sufficiently composable, we can use it to build obfuscators for more complicated functionalities such as multi-point circuits $\mathcal{P}^t$ and point circuits with multi-bit output $\mathcal{I}$.[3] This begs the question: is the above construction composable? We defer treatment of this issue to Section 3.3.3. For now, we assume that the obfuscator is sufficiently composable for our needs.

First, if $\mathcal{O}$ is t-composable, then it can be used to build an obfuscator $\mathcal{O}'$ for the family of t-point circuits $\mathcal{P}^t$ using the simple concatenation construction diagrammed in Figure 1-4. This is straightforward, and intuitively follows because a simulator S with oracle access to t different single-point circuits I_{w_1}, ..., I_{w_t} has essentially the same power as a simulator S' with oracle to one t-point circuit $I_{\{w_1,\ldots,w_t\}}$. The only technical issue concerns repetition of passwords: S can count the number of times a password appears in the list $\{w_1, \ldots, w_t\}$, but S' cannot. The fix for this issue is simple: the obfuscator $\mathcal{O}'(I_{\{w_1,\ldots,w_t\}})$ replaces any repetitions with uniformly random strings. The resulting construction is secure, but at the expense of achieving only approximate functionality. See Section 5.4 for a more thorough analysis.

Second, if $\mathcal{O}$ is $(\ell + 1)$-composable, we can use it to build an obfuscator for the family of point circuits with ℓ-bit output, as shown below.

Input: key $k \in \{0,1\}^n$ and message $m \in \{0,1\}^\ell$
1: construct circuit $D_0 = \mathcal{O}(I_k)$
2: **for** $i = 1$ to ℓ **do**
3: **if** $m_i = 1$ **then**
4: form the circuit $D_i = \mathcal{O}(I_k)$
5: **else**
6: form the circuit $D_i = \mathcal{O}(I_{k'})$, where $k' \xleftarrow{U} \{0,1\}^n$
7: **end if**
8: **end for**
Output: circuit that behaves as follows on input x: if $D_0(x)$ rejects, then output $\perp$, else compute $y_i = D_i(x)$ for $i \in \{1, \ldots, \ell\}$ and output the string y

It is simple to check that the circuit constructed by this construction has the functionality of $I_{(k,m)}$. The circuit D_0 is used to determine whether the input is the correct key. If so, then subsequent circuits D_1 through D_ℓ reveal the message bit by bit.

Checking that the security guarantee holds is more complicated.

Theorem 3.22. *Suppose $\mathcal{O}$ is a $t(\ell+1)$-composable virtual black-box obfuscator for $\mathcal{P}^1$. Then, this construction yields a t-composable virtual black-box obfuscator for point circuits with ℓ-bit output* [25]. *These connections hold for virtual gray-box obfuscation as well* [12].

[3] Another use of composability is discussed in Section 6.5.

The first two statements are proved in [25, Theorem 1], and the last one is in [12, Proposition 8.2]. We note that both of the constructions in this section are *generic* in the sense that they can be based on any sufficiently composable obfuscator for point functions, not just the (g, g^w) one that we built above. Hence, we can use them in the next two constructions of point circuits as well.

Random oracle model

In this section, we present an obfuscator for the family of point circuits that operates in the random oracle model. In this model, all algorithms have access to a common oracle $R : \{0,1\}^n \to \{0,1\}^{2n}$ that computes a random function. This enables a simple obfuscation due to Lynn *et al.* [63].

Input: string $w \in \{0,1\}^n$
1: find $t = R(w)$
Output: the circuit Υ_t that has t hardwired, and on input a string x, accepts if $R(x) = t$

This construction can easily be modified to accommodate the circuit $I_\emptyset$ that rejects all inputs: just choose $t \in \{0,1\}^{2n}$ uniformly at random. As a result, we claim that this construction is an obfuscator for the family $\mathcal{P}^1$. The proof of security constructs a simulator that emulates an execution of A and monitors its oracle queries.

Theorem 3.23. *The above construction is a strong virtual black-box obfuscator for the family of point circuits $\mathcal{P}^1$ in the random oracle model* [63, Lemma 6].[4]

Proof sketch. Approximate functionality follows from the fact that R is length-increasing, so collisions are statistically unlikely. To show the virtual black-box property, let A^R be an adversary, and we need to show the existence of a simulator S^{R,I_w} that behaves like A does. The simulator chooses a fake target $t' \xleftarrow{U} \{0,1\}^{2n}$ and builds a "fake" obfuscated circuit $\Upsilon_{t'}$. Then, it emulates an execution of $A^R(\Upsilon_t)$. When A makes an oracle query q, the simulator checks if q is the correct password by feeding it into its oracle for I_w. There are two cases.

- If $q = w$, then S aborts this execution of the adversary, forms a real obfuscation of I_w (which it can do now because it has learned w), and restarts A on the new circuit.
- Otherwise, S continues the current emulation of A. In particular, it responds to A's oracle query honestly.

In this way, S emulates the behavior of A perfectly, thus achieving the strong variant of security. □

This argument also works against generic adversaries whose output length is unrestricted. Hence, Theorem 3.4 does not hold generally in the random oracle model. (We will re-visit this issue shortly.)

A similar argument shows that this construction is t-composable for any $t = \text{poly}(n)$. Thus, we can use the generic transformation above to form an obfuscator for the family of multi-point circuits in the random oracle model.

We could also use the generic transformation to get an obfuscator for $\mathcal{I}$, but there is an easier way. Suppose there are two random oracles $R_1 : \{0,1\}^{2n} \to \{0,1\}^{2n}$ and $R_2 : \{0,1\}^{2n} \to \{0,1\}^{\ell(n)}$. This can easily be constructed from a single random oracle using standard techniques. Then, we can obfuscate the family $\mathcal{I}$ using the following construction.

Input: key $k \in \{0,1\}^n$ and message $m \in \{0,1\}^{\ell(n)}$
1: choose a random string $r \xleftarrow{U} \{0,1\}^n$

[4] In Chapter 5, we show that this construction is non-malleable as well (see Theorems 5.11 and 5.16).

2: let $a = R_1(k \circ r)$, $b = R_2(k \circ r)$, and $c = b \oplus m$

Output: circuit that stores r, a, c and on input x, computes $a' = R_1(x \circ r)$ and $b' = R_2(x \circ r)$, if $a' = a$ then it outputs $b' \circ c$, otherwise it outputs $\perp$

The proof of security for this construction is similar to the one above [63, Theorem 7].

A strongly one-way permutation

In this section, we present another obfuscator for the family of point circuits due to Wee [93]. It is based on the existence of one-way permutations that are extremely hard to invert.

Assumption 3.24 (Strongly one-way permutations [93])**.** *There exists an efficiently computable family of permutations $\{\pi_n : \{0,1\}^n \to \{0,1\}^n\}_{n\in\mathbb{N}}$ such that for every PPT adversary A, there exists a polynomial ξ such that*

$$\Pr\left[x \leftarrow \{0,1\}^n : A(\pi_n(x)) = x\right] \leq \frac{\xi(n)}{2^n}.$$

In words, this assumption states that any efficient adversary can only invert the one-way permutation π_n on a polynomial number of inputs. A permutation of this form can be used to develop a function $h_n : \{0,1\}^n \times \{0,1\}^{3n^2} \to \{0,1\}^{3n^2+3n}$ as follows:

$$h_n(w; \tau_1, \tau_2, \ldots, \tau_{3n}) = (\tau_1, \tau_2, \ldots, \tau_{3n}, \langle w, \tau_1\rangle, \langle \pi(w), \tau_2\rangle, \ldots, \langle \pi^{3n-1}(w), \tau_{3n}\rangle).$$

In words, this function computes several randomly-chosen hardcore predicates on different (but related) strings. This function is the basis for Wee's obfuscator.

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

1: choose random strings $\tau_1, \ldots, \tau_{3n} \xleftarrow{U} \{0,1\}^n$

2: compute $u = h_n(w; \tau_1, \tau_2, \ldots, \tau_{3n})$

Output: circuit that stores u and on input x, accepts iff $h(x; \tau_1, \ldots, \tau_{3n}) = u$

Note that the string u includes $\tau_1, \tau_2, \ldots, \tau_{3n}$, so the check in the final step can be performed. Wee proves that this construction is a virtual black-box obfuscator [93, Proposition 3.3] without going through intermediate definitions like Canetti [23] does.

There are a few ways to strengthen this result. The assumption can be slightly weakened to consider "strongly one-way permutation ensembles," which are collections of functions for which a randomly chosen one satisfies Assumption 3.24 with overwhelming probability. Also, the construction can be modified to allow multi-bit output. To do so, we form a longer function $h_{n,\ell} : \{0,1\}^n \times \{0,1\}^{3n^2+n} \to \{0,1\}^{3n^2+6n+\ell}$ as follows:

$$h_{n,\ell}(k; \tau_1, \tau_2, \ldots, \tau_{3n}, \tau_{3n+1}) = \\ (\tau_1, \ldots, \tau_{3n+1}, \langle k, \tau_1\rangle, \langle \pi(k), \tau_2\rangle, \ldots, \langle \pi^{3n-1}(k), \tau_{3n}\rangle, \langle \pi^{3n}(k), \tau_{3n+1}\rangle, \ldots, \langle \pi^{3n+\ell-1}(k), \tau_{3n+1}\rangle).$$

Hence, $h_{n,m}$ includes the same technique as before to hide the key k, but then it includes ℓ additional hardcore bits (all using τ_{3n+1}) that can be used to hide the message with a one-time pad.

More precisely, we obfuscate the circuit $I_{(k,m)}$ by computing

$$u = h(x; \tau_1, \ldots, \tau_{3n+1}) \oplus (0^{3n^2+6n} \circ m),$$

where the τ_i's are chosen uniformly at random. Then, form the circuit that stores u, and on input x, checks if the first $3n^2 + 6n$ bits of the string $s = h(x; \tau_1, \ldots, \tau_{3n+1})$ agree with u. If so, it outputs

the remaining bits of $s \oplus u$, and otherwise it outputs $\perp$. The proof of security for this construction can be found in [93, Theorem 3.7].

Conceptually, Wee provides evidence that this assumption is both feasible and necessary. On the feasibility side, a random permutation satisfies Assumption 3.24 with overwhelming probability [93, Theorem 4.1]. Of course, a random permutation is usually not efficiently computable.

In the random oracle model, though, efficiently-computable random permutations do exist! Hence, using an oracle to a random permutation R instead of π above yields a secure obfuscator in the random oracle model. The proof of security for this obfuscator differs from the one of Lynn *et al.* [63] described above in one crucial way: the simulator in Wee's construction does *not* have to modify any queries made in its emulation of the adversary. With this restriction, Theorem 3.4 does continue to hold in the random oracle model.

Finally, we look at the necessity of strong assumptions. The following theorem justifies the use of Assumption 3.24, and rationalizes the use of strong assumptions throughout this work.

Theorem 3.25. *Suppose that public-coin obfuscators exist for the family of point circuits. Then, there exists a family of functions $\mathcal{F}$ that is mostly injective and somewhat strongly one-way.*[5]

Proof sketch. Let $\mathcal{O}$ be a public-coin obfuscator for $\mathcal{P}^1$. For every choice r of the randomness to $\mathcal{O}$, form a function $f_r(x) = \mathcal{O}(x; r)$. The family of all such functions is mostly injective by approximate functionality, and somewhat strongly one-way by the virtual black-box property. □

Note that a randomized algorithm is called "public-coin" if it reveals the results of its random coin tosses. All three of the obfuscators for point circuits $\mathcal{P}^1$ in this section are public-coin. Furthermore, the specific extensions to $\mathcal{P}^m$ and $\mathcal{I}$ by Lynn *et al.* and Wee are public-coin, as is the generic extension to $\mathcal{P}^m$. However, the general extension to $\mathcal{I}$ is *not* public-coin, as it must hide the "fake" keys k' that it obfuscates.

Applications

In this section, we have constructed many obfuscators for the families $\mathcal{P}^1$, $\mathcal{P}^m$, and $\mathcal{I}$. We pause here to survey applications of these constructions in the literature.

Login programs. First, obfuscators of point circuits can be used to construct secure login programs. By oracle indistinguishability, an adversary attacking the login program can only employ a dictionary attack. This guarantee is deceptively strong, and is not provided by the secure login programs used in practice, which use a cryptographic hash function such as SHA-1.

For instance, SHA-1 should provide 2^{80} security against collisions by the birthday bound, but there are very clever attacks on SHA-1 that succeed in 2^{63} time [33, 91], and for this reason SHA-1 is claimed to be "broken." However, the analysis of collision-resistance is irrelevant to the problem at hand because most humans do *not* choose their passwords uniformly at random, but rather choose structured passwords; say, 8 alphanumeric characters. As a result, there is a more pressing concern against finding second-preimages, since a dictionary attack will succeed in at most 2^{48} time. It is unclear whether this is the best possible attack. In fact, it might be possible to break SHA-1 trivially on structured inputs (say, in 2^{10} time) even if security holds on uniform ones. By contrast, obfuscation does guarantee that a dictionary attack is the best possible.

[5] See [93, Proposition 4.4] for formal definitions of these two properties.

Encryption schemes. Obfuscators for point circuits with multi-bit output can be used to form symmetric key encryption schemes. This application was first noted by [25], and it is expanded upon in Chapter 4. Essentially, obfuscation yields an encryption scheme with strong leakage resilience and key-dependent message security. These properties are satisfied in a stronger way than currently possible under standard assumptions. Conversely, the known encryption schemes provide partial notions of obfuscation that can be realized under standard assumptions.

Subsequently, Bitanski and Canetti [12] show that the connection extends to public key encryption for obfuscators that are "re-randomizable" in the sense described in the proof of Theorem 3.21. Of the three constructions presented in this section, only the one of Canetti [23] is re-randomizable.

Furthermore, the composability of obfuscation is related to the type of security provided by the corresponding encryption scheme. In Chapter 4, we show that the standard notion of obfuscation yields semantic security, whereas self-composable obfuscation yields CPA security. Bitanski and Canetti [12] show that full composability of an obfuscator corresponds to a "multiple key, multiple message" secure encryption scheme, which remains secure even if the adversary receives encryptions of multiple messages under multiple keys, where the keys and messages are arbitrarily correlated.

Other applications. In addition to the puzzle applications already described, Canetti [23] shows how obfuscators for point circuits can be used to remove random oracles from cryptographic constructions in certain cases, and to build *content-concealing* digital signature schemes whose signatures do not reveal any information about the message that was signed. The solution to the latter problem is elegant: instead of signing a message m, output an obfuscator of the point circuit I_m along with a signature of this circuit. Verification of a putative message m' proceeds by checking that $I_m(m')$ accepts and that the signature on the circuit is valid.

Lynn *et al.* [63] show how obfuscation of $\mathcal{I}$ can be used to form complex access control mechanisms that allow users with different access privileges to access different parts of a graph. Additionally, they show how to obfuscate regular expressions. Specifically, they form circuits that reveal a hidden message if its input satisfies a certain regular expression (rather than just an equality test in the case of $\mathcal{I}$). Finally, they perform "fuzzy" equality testing that can test whether its input is "close" to a stored password in a tree metric.[6]

These results exploit the fact that the virtual black-box property is compatible with black-box reductions. Therefore, any family that is black-box reducible to $\mathcal{P}^m$ or $\mathcal{I}$ is also obfuscatable.

Theorem 3.26. *Let $\mathcal{C}$ and $\mathcal{D}$ be two families of circuits. Suppose there exist two (ensembles of) circuits P, Q such that for every $C \in \mathcal{C}$ there is a $D \in \mathcal{D}$ such that $C \equiv P^D$ and $D \equiv Q^C$, and furthermore that D can be found efficiently. If $\mathcal{D}$ is virtual black-box obfuscatable, then $\mathcal{C}$ is too.*

Proof sketch. Let $\mathcal{O}$ be an obfuscator for $\mathcal{D}$, and we form an obfuscator $\mathcal{O}'$ for $\mathcal{C}$ as follows: given $C \in \mathcal{C}$, find the $D \in \mathcal{D}$ such that the above property holds and output $P^{\mathcal{O}(D)}$. The proof of security constructs a simulator for $\mathcal{O}'$ by using the corresponding simulator for $\mathcal{O}$ in a black-box manner. See [63, Lemma 1] for more details. □

3.2.2 Cryptographic applications

In this section, we review the known constructions under average-case obfuscation. It is possible to obfuscate point circuits under this definition: in fact, applying any one-way permutation to the password will do [53, Theorem 4.5]! This is not quite so illuminating as the constructions in the prior section because the average-case guarantee only has to hold for a uniformly chosen password.

[6] This should not be confused with the work of Dodis and Smith [39], who obfuscate programs that can tolerate much more expressive types of error but under an average-case security definition.

Instead, average-case obfuscation is best suited toward cryptographic tasks, as shown in the following heuristic by Hohenberger *et al.* [55]. Suppose a cryptographic scheme has the following two properties:

1. The scheme is secure against black-box adversaries with oracle access to functionality C selected randomly from a family $\mathcal{C}$.
2. A distinguisher with oracle access to C can test whether an adversary can break the security guarantee of the scheme.

Then, the cryptographic scheme is also secure against adversaries that are given an average-case obfuscation of C.

This heuristic will come in handy in the applications in this section toward transforming symmetric primitives into public ones, constructing secure re-encryption and secure encrypted signature schemes, and shuffling ciphertexts in public.

Symmetric to public transformations

This section describes the work of Hofheinz *et al.* [53], which studies the cryptographic primitives that can (and cannot) be obfuscated using average-case obfuscation. The goal of this section is to transform symmetric key primitives into public key ones.

Recall from Theorem 3.15 that deterministic primitives like pseudorandom functions cannot be obfuscated under the average-case definition. Hence, we consider two probabilistic primitives: symmetric key encryption and message authentication codes (MACs). Ideally, we hope to construct public key versions of these primitives using obfuscation. We illustrate the concept using encryption.

Given a symmetric key encryption scheme, let Enc_k denote the encryption circuit used by the scheme with secret key k. Typically, k is hardcoded into the circuit in a readily identifiable manner. If we can obfuscate Enc_k, the resulting circuit would allow messages to be encrypted without revealing any information about k. Hence, $\mathcal{O}(\mathrm{Enc}_k)$ could be used as the encryption routine to a public key encryption scheme, along with the same key generation and decryption circuits as before.

Hofheinz *et al.* [53] show that this intuitive concept is realizable, with two caveats. First, there are some symmetric key encryption schemes that cannot be obfuscated in this manner [53, Remark 4.1], and second, the transformation only holds for some forms of security. We explore these two issues in detail.

Consider a symmetric key encryption scheme that includes the key pair of a digital signature scheme in its secret key, and whose encryption routine digitally signs every ciphertext and includes a copy of the verification key in its output. By the functionality requirement, an obfuscator must then be able to sign ciphertexts as well because the distinguisher in the average-case security definition can obtain the correct verification key. This is impossible by the unforgeability of the signature scheme, so obfuscation is impossible for encryption schemes of this form.

On the other hand, there are encryption schemes that are obfuscatable. Take any public key encryption scheme with key pair (sk, pk), and view it as a symmetric key scheme whose secret key consists of sk and pk. These symmetric key schemes can be trivially obfuscated by restoring the public key structure. There may be other symmetric key schemes that can be obfuscated, and the result of such a transformation would also be a public key encryption scheme.

Theorem 3.27. *Let* $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ *be a CPA secure symmetric key encryption scheme. If* Enc_k *is average-case obfuscatable, the result is a CPA secure public key encryption scheme. However, the corresponding property does not hold for CCA security* [53, Theorems 4.1 and 4.2].

Proof sketch. By the CPA security of the symmetric key encryption scheme, no adversary with black-box access to Enc_k can break the scheme. Average-case security dictates that the same is true if the adversary is given $\mathcal{O}(\text{Enc}_k)$ as well, resulting in a public key encryption scheme. On the other hand, CCA security considers adversaries that are given a *decryption* oracle, which is not protected by average-case security. In other words, it doesn't obey the Hohenberger *et al.* heuristic. □

In theory, this procedure could be used to find new public key encryption schemes, which would be beneficial as we currently know many symmetric key schemes but only a few public key schemes under specific number-theoretic assumptions. However, thus far no additional public key schemes have been found using this method.

Similarly, we can hope to transform a message authentication code into a digital signature scheme by obfuscating the verification routine of the MAC. The results are similar to the encryption setting: it is possible to preserve "verifier only attacker" (VOA) security, in which the attacker only receives oracle access to a verification oracle, but it is not possible to preserve existential unforgeability under chosen message attacks, which provide a signing oracle to the attacker as well. Furthermore, there are MACs that are unobfuscatable in the average-case sense.

Proxy re-encryption

One specific application of the randomized definition is to construct proxy re-encryption schemes [55], which we illustrate with an example. Suppose Bob decides to go on vacation and asks his webmail provider Alice to forward his email to David, so David can respond to any problems while Bob is away. Simply passing on the messages is insufficient because Bob regularly receives encrypted email, so David won't be able to read it.

A simple solution to this problem is for Bob to give Alice his secret key. Then, Alice can decrypt his email, encrypt it again under David's public key, and forward the message to David. Let C be the circuit that performs this procedure with Bob's secret key sk_B and David's public key pk_D hardcoded. Given a ciphertext c encrypted under pk_B, the circuit produces a ciphertext of the same message that is encrypted under pk_D.

However, Bob doesn't trust Alice, so he doesn't want to give away his secret key. Instead, he wants to give Alice a "token" that allows her to perform the re-encryption procedure without being able to read his emails. This can be accomplished by obfuscating the simple "decrypt, and then re-encrypt" functionality, so Alice can execute it without knowing Bob's secret key.

This functionality does obey the heuristic above, and we claim that average-case security provides a strong guarantee here. If Alice were simply given a black box implementing C, she cannot use it in a meaningful way because she doesn't understand the inputs or outputs to the box (since they are encrypted and she doesn't know sk_B or sk_D). Thus, average-case security dictates that the same is true if Alice possesses an obfuscated circuit $\mathcal{O}(C)$.

Hohenberger *et al.* [55] construct an obfuscator for the proxy re-encryption functionality for the public key encryption scheme of Boneh, Boyen, and Shacham [16]. This encryption scheme requires a group G of order p with a bilinear map $\mathbf{e} : G \times G \to G_T$, and operates as follows.

KeyGen: Select a random $g \xleftarrow{U} G$, and $x, y \xleftarrow{U} \mathbb{F}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$. Set $sk = (g, x, y)$ and $pk = (g, g^x, g^y)$.

Encrypt: Given a message $m \in G$, select $r, s \xleftarrow{U} \mathbb{F}_p$ and output $[g^{rx}, g^{sy}, g^{r+s} \cdot m]$.

Decrypt: Given a ciphertext $[U, V, W]$, compute $W/(U^{1/a} \cdot V^{1/b})$.

It is straightforward to check that decryption operates correctly. Our goal is to construct a secure proxy re-encryptor. Suppose Bob and David have secret keys $sk_B = (g_B, x_B, y_B)$ and $sk_D =$

(g_D, x_D, y_D), and consider a ciphertext $[U, V, W]$ that encodes the message m under Bob's public key. Then,

$$[U^{x_D/x_B}, V^{y_D/y_B}, W]$$

is a valid encryption of m under David's public key.

Therefore, Bob and David can combine their secret keys to form the quotients $\frac{x_D}{x_B}$ and $\frac{y_D}{y_B}$, which Alice can then use to compute the proxy re-encryption functionality. However, this reveals too much information to Alice. For instance, she could compute $\frac{x_B}{x_D}$ and $\frac{y_B}{y_D}$, and execute the re-encryption functionality *backwards* to send David's emails to Bob.[7]

The solution to this problem provided by Hohenberger *et al.* [55] is to hide these quotients "in the exponent" and give Alice $(g_D)^{x_D/x_B}$ and $(g_D)^{y_D/y_B}$. Alice can still "meld" the quotients $\frac{x_D}{x_B}$ and $\frac{y_D}{y_B}$ into a ciphertext using the bilinear group. Specifically, given a ciphertext $[U, V, W]$ intended for Bob, she can form the ciphertext

$$[\mathbf{e}(U, (g_D)^{x_D/x_B}), \mathbf{e}(V, (g_D)^{y_D/y_B}), \mathbf{e}(W, g_D)].$$

We claim that this ciphertext is a valid encryption of m intended for David, except that it involves group elements in the target group G_T. Luckily, David can still decrypt the message if it comes from a small (polynomial sized) message space. Furthermore, Hohenberger *et al.* show that this construction satisfies average-case security. That is, Alice cannot use the proxy re-encryption functionality in a non-black-box way even with $(g_D)^{x_D/x_B}$ and $(g_D)^{y_D/y_B}$. A formal treatment of this statement can be found in [55, Theorem 4].

Delegatable signatures

Another application of average-case security, due to Hada [48], is to the problem of delegatable signatures. We illustrate the problem by picking up where the last example left off. Suppose that while on vacation, Bob uses a public computer to write a message to David. He wants to digitally sign his email so David can trust its authenticity. However, Bob doesn't have his signing key because he is on vacation (and even if he did, he wouldn't want to store the key on the public computer).

Anticipating this issue before the trip, Bob wants to give his webmail provider Alice a token that will allow her to sign messages to David on his behalf. However, this token should not reveal Bob's signing key, nor should it give Alice the ability to sign messages and send them to other people.[8] To enforce the last constraint, the token should not only sign Bob's message but also encrypt it using David's public key, so only he can read it.

More formally, suppose Bob uses an existentially unforgeable signature scheme, and David uses a CPA secure public key encryption scheme. Consider the "encrypted signature" circuit C_{sk_B, pk_D} that stores Bob's signing key and David's public key, and on input a message m, digitally signs m using sk_B and then encrypts the result using pk_D. If Alice only has black box access to this circuit, then she would be unable to learn sk_B or send signed messages to other users by existential unforgeability. Thus, our hope is that the security of the encrypted signature scheme remains intact if Alice is given an average-case obfuscation of C.

Sadly, our hope is misplaced here. Average-case obfuscation alone does not suffice for this scheme because (as we saw before for MACs) existential unforgeability does not obey the Hohen-

[7] Another undesirable property of this method is that David's secret key is required to generate Alice's token, which was not necessary in the naïve "decrypt and re-encrypt" functionality.

[8] Note that a malicious Alice could sign bogus messages (that did not originate from Bob) and send them to David. There is nothing we can do to prevent this attack. Instead, the goal is to prevent all other attacks.

berger *et al.* heuristic. Hada [48] resolves this issue by strengthening the average-case obfuscation guarantee to hold in the presence of related oracles. Informally, *average-case obfuscation with dependent oracles* in a set T requires that the security guarantee in Definition 3.14 hold even if the distinguisher, who is already given an oracle C, is also given oracle access to $T(C)$. In essence, T is a form of auxiliary input, but it is an oracle instead of a string.

Under this stronger definition, existential unforgeability is preserved by average-case obfuscation, where the dependent oracle T is a signing oracle. Hada shows how to construct an obfuscator for C_{sk_B,pk_D} using the bilinear encryption scheme used above [16] and the digital signature scheme of Waters [92]. Then, Hada shows how to use the encrypted signature functionality to build a signcryption scheme. We refer the reader to [48] for details.

Shuffling ciphertexts

Our final application of obfuscation with a randomized definition, due to Adida and Wikström [1], is to shuffling ciphertexts in public. This can be used in the mix-nets of voting algorithms.

Given a list of ciphertexts, suppose we wish to "shuffle" the messages around and create a new list of ciphertexts that contains the same messages but in a randomized order. We could decrypt all of the messages, apply a random permutation, and then encrypt all of the messages again, but decryption requires knowledge of the secret key. An obfuscation of the "decrypt, permute, and encrypt" procedure allows the shuffling procedure to be done in a public manner, so observers can verify that the shuffling is done properly without knowing the permutation that connects the messages.

Adida and Wikström construct an obfuscator for this functionality based on the Boneh, Goh, Nissim [17] or Paillier [76] cryptosystems, and show that the application to mix-nets is feasible.

> "The Paillier construction is practical for normal sized voting precincts in the USA: $N \approx$ 2000 full length messages can be accommodated, and, given one week of pre-computing, the obfuscated shuffle can be evaluated overnight. Furthermore, all constructions are easily parallelized [1]."

However, their construction does not achieve average-case security, but rather a weaker guarantee called *public key obfuscation* that was defined by Ostrovsky and Skeith [75]. In this definition, the security of the obfuscator and the underlying functionality depend on the same public key. Additionally, this definition does not explicitly consider the difference between black-box and non-black-box access to a program [55]. We refer the reader to [1] for details on the security definition and construction.

3.3 Connections

In this section, we delve deeper into the conceptual details behind the various definitions. First, we examine the limits of the definitions by demonstrating circuits that *cannot* be obfuscated under them. Then, we distinguish between the strength of the various definitions. Finally, we examine issues related to the composability of obfuscation.

3.3.1 Impossibility

Immediately after the seminal work of Barak *et al.* [10] codified the virtual black-box definition, they proved that it is impossible to achieve in general. In principle, there are two ways that obfuscation might be impossible to achieve.

1. There may exist some "unobfuscatable" programs for which every garbled variant reveals non-black-box information.

2. It might be possible to obfuscate all programs, but not "simultaneously" due to the efficiency constraint. In other words, perhaps the only obfuscators that exist are inefficient.

It turns out that the first, stronger, impossibility result holds. To describe the problem, it helps to switch from circuits to Turing machines.[9] Obfuscation of a family of Turing machines $\mathcal{T}$ uses the same functionality and virtual black-box properties as Definition 2.4, but the polynomial slowdown property must hold on a per-input basis.

Definition 3.28 (Polynomial slowdown for Turing machines [10])**.** There exists a polynomial ξ such that for every TM $M \in \mathcal{T}$, $|\mathcal{O}(M)| \leq \xi(|M|)$, and for all inputs $x \in \{0,1\}^*$, if $M(x)$ halts in t steps then $\mathcal{O}(M)(x)$ halts in at most $\xi(t)$ steps.

It is harder to obfuscate Turing machines than to obfuscate circuits [10, Proposition 2.3] because a Turing machine can be run on itself: that is, a description of M can be fed back into M as input. This issue does not arise with circuit obfuscation because a circuit has a fixed input length that is usually shorter than its description length.

The general impossibility result exploits this property of Turing machines, and uses two obfuscated Turing machines to understand each other.

Theorem 3.29 (General impossibility)**.** *There exists a family of Turing machines that cannot be obfuscated* [10, Proposition 3.4 and Theorem 3.5].

Proof sketch. First, we prove the general impossibility of 2-composable obfuscation. Let $\mathcal{M}$ be the family of all Turing machines of the forms

$$M_{(k,m)}(x) = \begin{cases} m & \text{if } x = k, \\ \bot, 1^{|m|} & \text{otherwise.} \end{cases} \qquad N_{(k,m)}(\langle T \rangle) = \begin{cases} m & \text{if } T(k) = m, \\ \bot & \text{otherwise.} \end{cases}$$

In words, $M_{(k,m)}$ is a point TM with multi-bit output (i.e., the Turing machine analog of a circuit in $\mathcal{I}$), but N is a more complicated circuit that receives the description of a Turing machine as input! Essentially, N is a "helper" TM that knows the correct input k to query.[10]

For any potential obfuscator $\mathcal{O}$, an adversary with two obfuscated Turing machines $P = \mathcal{O}(M_{(k,m)})$ and $Q = \mathcal{O}(N_{(k,m)})$ can execute $Q(\langle P \rangle)$, which outputs m. However, a simulator with oracle access to $M_{(k,m)}$ and $N_{(k,m)}$ has no hope of learning m.

Finally, we combine the two TMs $M_{(k,m)}$ and $N_{(k,m)}$ into a single Turing machine $M_{(k,m)} \# N_{(k,m)}$ that chooses whether to run M or N by conditioning on its first input bit. The family of combined TMs is unobfuscatable (in the standard sense, with just one copy) by a similar argument. □

Proving the general impossibility of circuit obfuscators is harder. The first part of the argument works as before to show that the circuits $I_{(k,m)} \in \mathcal{I}$ and those of the form

$$J_{(k,m)}(\langle C \rangle) = \begin{cases} m & \text{if } C(k) = m, \\ \bot & \text{otherwise,} \end{cases}$$

[9] Note that this is the only section in the thesis that considers Turing machine obfuscation.

[10] Technically, $N_{(k,m)}$ as presented here is uncomputable. To make it computable (and efficient), we impose a time bound. Thus, N emulates T for a certain number of time steps, and if T hasn't halted by then, N outputs $\bot$.

cannot be simultaneously obfuscated, as long as $J_{(k,m)}$ has sufficiently long input length that it can accept a description of $\mathcal{O}(I_{(k,m)})$ as input. However, the above method to combine two TMs into one does not suffice for circuits because it would force $I_{(k,m)}$ and $J_{(k,m)}$ to have the same input length. Details on how to overcome this problem are explained in [10, Theorem 3.8].

Theorem 3.30. *If one-way functions exist, then there exists an unobfuscatable family of circuits.*

Barak *et al.* provide several amplifications to this result.

1. The impossibility proof for TMs reletivizes, and thus holds in the random oracle model as well, but the proof for circuits does not [10, Propositions 4.14 and 4.15].

2. Suppose symmetric key encryption schemes, MACs, digital signature schemes, or pseudorandom function ensembles exist. Then, there exists an unobfuscatable version of the primitive [10, Theorem 4.12]. These results are similar to those in Section 3.2.2 for average-case security.

3. If factoring Blum integers is hard, then there exists an unobfuscatable circuit family that can be computed by constant-depth threshold circuits (i.e., in TC^0) [10, Theorem 4.13].

The last result rules out the (otherwise appealing) notion of obfuscating a low-depth complexity class. One might hope to avoid these impossibility results by using a weaker definition.

Bitanski and Canetti [12] show that the impossibility result for TMs immediately extends to the virtual gray-box setting, but the proof for circuits does not. In fact, they could not rule out the possibility of universal virtual gray-box obfuscation.

However, Goldwasser and Rothblum [46] provide evidence that this is not the case. They formulate a different impossibility proof for circuits under a stronger variant of best-possible obfuscation in which Definition 3.8 holds with statistical (rather than computational) indistinguishability.

Theorem 3.31. *If the family of 3-CNF circuits (a subfamily of AC^0) can be statistically best-possible obfuscated, then the polynomial hierarchy collapses to the second level* [46, Theorem 4.1].

On the other end of the spectrum, the Barak *et al.* impossibility results trivially extend to the stronger definition with dependent auxiliary input. Goldwasser and Kalai [43] show even stronger impossibility results in this setting. Specifically, at least one of the following is unobfuscatable:

1. The class of "point filter-circuits" for every NP-complete language. These are indicator circuits of the form $\delta_{x,b}$ that reveal a bit b when given an NP witness to x as input.

2. Every class with super-polynomial pseudo entropy. This includes every class of pseudorandom functions,[11] and certain types of encryption and digital signature schemes.

Thus, obfuscation with auxiliary input is impossible for very low complexity circuits or for many cryptographically interesting ones.

While the above impossibility results are generic in nature, we conclude this section with an impossibility result for point circuits. Recall from Chapter 2 that an obfuscator $\mathcal{O}$ does not just depend on the functionality of its input circuit, but rather on its *representation* as well. For this reason, we stressed that the circuits $I_w \in \mathcal{P}^1$ store w in a readily identifiable manner. An interesting question, first posed by Tal Malkin, is whether every representation of point circuits can be obfuscated.

[11] Note the difference from the Barak *et al.* result above that there exists an unobfuscatable family of PRFs. This criterion rules out *all* PRFs, just as Theorem 3.15 does in the average-case setting.

Consider a family of obfuscated point circuits

$$\hat{\mathcal{P}}^1 = \{\mathcal{O}(I_w; r) : w \in \{0,1\}^n, r \in \{0,1\}^{\xi(n)}\}.$$

Intuitively, $\hat{\mathcal{P}}^1$ should be trivially obfuscatable (i.e., there is no need to obfuscate circuits *again*). Unfortunately, this intuition does not quite hold because an adversary may be able to distinguish between obfuscations of the same circuit under different randomness values $\mathcal{O}(I_w; r)$ and $\mathcal{O}(I_w; r')$. Goldwasser and Rothblum [46] show this for the construction of Lynn *et al.*

Theorem 3.32. *In the random oracle model, there exists a family of circuits $\mathcal{C}$ that is functionally equivalent*[12] *to $\mathcal{P}^1$ but cannot be best-possible obfuscated* [46, Theorem 5.1].

This impossibility result trivially extends to stronger security guarantees such as the virtual black-box property.

3.3.2 Separations

In Section 3.1, we defined several security properties and showed simple implications between them (the down arrows in Figure 3-1). In this section, we show that the corresponding "up" relationships are unlikely to hold.

Best-possible obfuscation

First, Goldwasser and Rothblum [46] provide evidence that best-possible obfuscation is unlikely to imply virtual black-box obfuscation. Their separation considers "probabilistic ordered binary decision diagrams (POBDDs)," which are logarithmic-space Turing machines whose input tape can only move from left to right.

They can also be easily represented as circuits, although we do not describe the transformation here. Let $\mathcal{B}$ denote the circuit family of all POBDDs under this representation (in particular, $\mathcal{B}$ includes the family of point circuits $\mathcal{P}^1$). This family has a special property: given any POBDD $C \in \mathcal{B}$, it is possible to compute the (unique) smallest circuit $D \in \mathcal{C}$ such that $C \equiv D$ [21].

Theorem 3.33. *The family $\mathcal{B}$ is best-possible obfuscatable, but there does not exist a virtual black-box obfuscator for $\mathcal{B}$ whose resulting circuits are also in $\mathcal{B}$* [46, Propositions 3.2 and 3.3].

Proof. Let $\mathcal{O}$ be the obfuscator such that $\mathcal{O}(C) \triangleq D$, the canonical representative for C. Then, $\mathcal{O}$ clearly satisfies indistinguishability, which is equivalent to best-possible obfuscation.

On the other hand, suppose $\mathcal{O}'$ is any algorithm whose outputs are also circuits in $\mathcal{B}$. An adversary $A(\mathcal{O}(C))$ can compute the canonical circuit $D \equiv C$, but the simulator S^C cannot. Thus, $\mathcal{O}'$ cannot be a virtual black-box obfuscator for $\mathcal{B}$. □

Note that this separation does not show that virtual black-box obfuscation of $\mathcal{B}$ is impossible in general: there could be an obfuscator $\mathcal{O}'$ for $\mathcal{B}$ whose outputs are not constrained to POBDDs.

Virtual gray-box

A recent result of Bitanski and Canetti [12] separates the strength of virtual gray-box and virtual black-box obfuscation. Note that, among other uses, this immediately yields a new separation between best-possible and virtual black-box obfuscation.

[12] We extend the notion of functional equivalence to families by saying that for every $C \in \mathcal{C}$, there exists $P \in \mathcal{P}^1$ such that $C \equiv P$, and vice-versa.

Theorem 3.34. *If one-way permutations exist, then there exists a family $\mathcal{T}$ of Turing machines that are virtual gray-box obfuscatable but not virtual black-box obfuscatable. This separation also holds for circuits if there exists a pseudorandom function ensemble that is exactly learnable by an unbounded adversary with polynomially many queries* [12, Proposition 4.1].

Proof sketch. We only consider the TM separation here, which is based on the technique used in Theorem 3.29. Let

$$f = \{f_n : \{0,1\}^n \to \{0,1\}^n\}$$

be a family of one-way permutations. For any k and m, let $F_{(k,m)}$ be the Turing machine that outputs $f_n(k)$ and $f_n(m)$ on every input value, and let M and N be the TMs defined in Theorem 3.29. We claim that the family of Turing machines

$$\mathcal{T} = \{F_{(k,m)} \# M_{(k,m')} \# N_{(k,m)} : n \in \mathbb{N}, k, m, m' \in \{0,1\}^n\}$$

is trivially obfuscatable under the virtual gray-box definition because it is learnable by an unbounded simulator. Indeed, the simulator can query $F_{(k,m)}$ to learn $f_n(k)$ and $f_n(m)$, invert the one-way permutation to find k and m, and query $M_{(k,m')}(k)$ to find m'.

On the other hand, $F_{(k,m)}$ is useless to an efficient simulator, so $\mathcal{T}$ cannot be obfuscated under the virtual black-box definition. Specifically, an efficient adversary can determine whether $m = m'$ but an efficient simulator has no hope of doing so. □

Auxiliary input

The final result in this section shows that the virtual black-box property with dependent auxiliary input seems to be a non-trivial extension of the original virtual black-box property. Hence, auxiliary input does not come "for free." By contrast, the corresponding relationship does hold for the virtual gray-box property (see Theorem 3.11).

Specifically, Goldwasser and Kalai [43] show that the obfuscator for point circuits constructed by Wee [93] is unlikely to permit dependent auxiliary input. Recall from Section 3.2.1 that Wee's construction relies on a strongly one-way permutation π and a function of the form

$$h_n(w; \tau_1, \tau_2, \ldots, \tau_{3n}) = (\tau_1, \tau_2, \ldots, \tau_{3n}, \langle w, \tau_1 \rangle, \langle \pi(w), \tau_2 \rangle, \ldots, \langle \pi^{3n-1}(w), \tau_{3n} \rangle).$$

Consider auxiliary input on w that is chosen in the following manner: choose a random string $r \xleftarrow{U} \{0,1\}^n$ and reveal

$$z(w, r) \triangleq (w \oplus r, \pi(w) \oplus r, \ldots, \pi^{3n-1}(w) \oplus r).$$

For any choice of r, there exists an adversary that can compute x from $h_n(w; \tau_1, \tau_2, \ldots, \tau_{3n})$ and $z(w, r)$. However, Goldwasser and Kalai conjecture that this auxiliary input function is one-way, so a simulator cannot use it to learn w.

3.3.3 Composability

Recall that a composable obfuscator remains secure against adversaries who possess multiple obfuscated circuits (see Definition 2.5 for details). There are two types of composability: self composability refers to the possibility of receiving many obfuscations of the same circuit, whereas full composability allows for obfuscations of different circuits as well.

We saw the usefulness of composable obfuscation in Section 3.2.1, which shows how to construct an obfuscator for $\mathcal{I}$ given a composable obfuscator for $\mathcal{P}^1$. Even self-composability yields valuable

security guarantees: we will see in Section 4.3 that a self-composability of an obfuscator for $\mathcal{I}$ is related to the ability to prove CPA security for its corresponding encryption scheme.

In this section, we show that composability is a non-trivial extension of the virtual black-box property, and then we examine how the relationships between the security definitions in Figure 3-1 are affected by composition.

Separations

It is rather simple to see that composability appears to be more powerful than the virtual black-box property alone. We begin with a simple separation in the random oracle model.

Theorem 3.35. *In the random oracle model, there exists a virtual black-box obfuscator that is not even 2-composable* [63, Claim 3].

Proof. The construction of Lynn *et al.* [63] described in Section 3.2.1 is deterministic. Hence, given two obfuscated circuits, an adversary can easily determine whether they accept the same string, which a simulator cannot do. □

Note that obfuscators in the standard model must be randomized for exactly this reason [23]. Hence, a similar proof will not work, and in fact an analog of Theorem 3.35 is not known in the standard model. We can prove a slightly weaker bound, however.

Theorem 3.36. *Let $\mathcal{O}$ be a virtual black-box obfuscator for $\mathcal{P}^1$. Then, there exists an obfuscator $\mathcal{O}'$ for $\mathcal{P}^1$ that is not $\Omega(n)$ self-composable* [25, Lemma 1]. *Furthermore, if $\mathcal{O}$ is virtual black-box with dependent auxiliary input, then $\mathcal{O}'$ is too* [12, Proposition A.1].

Proof sketch. Define $\mathcal{O}'$ as follows:

$$\mathcal{O}'(w; r, s) = (\mathcal{O}(w; r), s, \langle w, s \rangle),$$

where $\langle w, s \rangle = \sum_i w_i s_i \pmod 2$. Thus, $\mathcal{O}'$ outputs the circuit made by $\mathcal{O}$ along with a Goldreich-Levin bit. Using the Goldreich-Levin theorem [42], it can be shown that $\mathcal{O}'$ is an obfuscator, and that $\mathcal{O}'$ allows dependent auxiliary input if $\mathcal{O}$ does. However, an adversary with n copies of $s, \langle w, s \rangle$ for different randomness s can compute w by solving the system of n linear equations. □

Implications

In spite of the above difficulty results, composable obfuscators for point circuits are still desired due to their applicability. In this section, we show two methods to prove composability. The following theorem of Bitanski and Canetti helps in this search.

Theorem 3.37. *For any $t = \text{poly}(n)$, an obfuscator for point circuits is t-distributionally indistinguishable (Definition 3.19) if and only if it is t-virtual gray-box. For constant t, the equivalence extends to the t-virtual black-box property as well (Definition 2.5)* [12, Theorem 5.1].

The proof of this theorem uses some of the ideas in the proof of Theorem 3.18 along with novel proof techniques. We refer the reader to [12, Section 5.3] for a complete proof.

Conceptually, this theorem shows that many of the connections in Figure 3-1 compose for obfuscators of $\mathcal{P}^1$. In particular, it shows that the standard (non-composable) virtual black-box and virtual gray-box properties are equivalent for point circuits. This statement extends to point circuits with multi-bit output as well.

Theorem 3.38. *Let $\mathcal{O}$ be a virtual gray-box obfuscator for the family of point circuits with ℓ-bit output. Then, $\mathcal{O}$ is also a virtual black-box obfuscator for this family.*

On a practical level, this theorem yields a way to prove the composition of the Canetti (g, g^w) construction described in Section 3.2.1.

Assumption 3.39 (t-Strong DDH assumption)**.** *There exists a sequence of groups*

$$\mathcal{G} = \{G_n : n \in \mathbb{N}, p_n = \text{poly}(n) \text{ is prime}\}$$

with efficient representation and computation of the group operations, such that the following holds for any distribution ensemble $\mathcal{X} = \{\mathcal{X}_n\}$ over vectors in $\mathbb{F}_{p_n}^t$ whose projection to each coordinate is well-spread:

$$\{(g_1, g_a^{a_1}), \ldots, (g_t, g_t^{a_t}) : \boldsymbol{g} \xleftarrow{U} G_n^t, \boldsymbol{a} \leftarrow \mathcal{X}_n\} \approx_c \{(g_1, g_a^{a_1}), \ldots, (g_t, g_t^{a_t}) : \boldsymbol{g} \xleftarrow{U} G_n^t, \boldsymbol{a} \xleftarrow{U} \mathbb{F}_{p_n}^t\}$$

Suppose that the t-strong DDH assumption holds. By generalizing the argument in the proof of Theorem 3.21, it can be shown that this assumption implies that the (g, g^w) construction is t-distributionally indistinguishable. As a result, Theorems 3.22 and 3.37 yield the following:

- If t is a constant, then there exists a virtual black-box obfuscator for point circuits with $(t-1)$-bit output.
- If t is polynomial in n, then the above statement holds for virtual gray-box security.

Note that the length bound in the obfuscator for $\mathcal{I}$ depends on the parameter t. Thus, obfuscating longer length messages requires making a stronger assumption. On a positive note, Bitanski and Canetti provide evidence for the feasibility of this assumption for any $t = \text{poly}(n)$ by showing that it holds in the generic group model [12, Proposition 7.2].[13]

Finally, Bitanski and Canetti also consider the composability of obfuscators with auxiliary input. Recall from Theorem 3.36 that such obfuscators are not, in general, $\Omega(n)$ self-composable. The following theorem provides a partial converse.

Theorem 3.40. *Any virtual black-box obfuscator with auxiliary input is also t-self-composable for any constant t* [12, Proposition A.2].

We emphasize that this result holds for an obfuscator for any family, not just point circuits or their generalizations. We will see a meaningful use of self-composability in the next chapter.

[13] Details on the generic group model can be found in Section 6.3.2.

Chapter 4

Obfuscation and Symmetric Encryption

This chapter is based on joint work with Ran Canetti, Yael Kalai, and Daniel Wichs [26, 27].

4.1 Introduction

In addition to its broader appeal, obfuscation has the ability to solve many cryptographic problems using a two-step framework: find the simplest program that solves the problem if security were not a concern, and obfuscate this simple program to obtain the required security. However, as described in Section 3.2, only a few such applications are currently known, and most of them use a definition of obfuscation that only provides average-case security.

The goal of this chapter is to relate virtual black-box obfuscation to the problem of symmetric key encryption. If security were not a concern, then "ciphertexts" in a symmetric key encryption scheme could simply be point circuits with multi-bit output $I_{(k,m)}$. These ciphertexts are secure against attackers that only use $I_{(k,m)}$ in a black-box way. However, in the real world, no security is provided because malicious users could simply read the message m from a description of $I_{(k,m)}$. Instead, Canetti and Dakdouk [25] observe that obfuscations of $I_{(k,m)}$ under the virtual black-box definition, which they call *digital lockers*, do provide the necessary security in the real world.[1]

We expand upon their work and show tight connections between obfuscation and encryption. It turns out that digital lockers enable a very strong form of symmetric encryption that simultaneously achieves leakage resilience and key-dependent message (KDM) security. The strength of digital lockers is primarily due to the fact that the virtual black-box property must hold *for all* $I_{(k,m)}$.

On a conceptual level, our result provides a formal connection between leakage resilience and KDM security. Many of the previously known constructions and proof techniques to solve these two problems are similar (see [18, 71], [6, 38], and [3, 6] for examples). Hence, the two problems always appeared to be correlated, although no formal link between the problems had existed.

On a practical level, the standard virtual black-box definition appears too strong to achieve under standard cryptographic assumptions (see Theorem 3.25). We demonstrate several meaningful weakenings of the definition that correspond to the properties desired by encryption schemes. These connections allow us to take advantage of the large amount of research in leakage resilience and KDM security [3, 5, 6, 8, 9, 18, 22, 38, 50, 52, 54, 58, 71] to construct digital lockers under standard assumptions that satisfy partial notions of obfuscation. The specific connections are shown below.

[1] There are a few known obfuscators for digital lockers based on different cryptographic assumptions [25, 63, 93]. See Section 3.2.1 for details. The results of this chapter are generic in the sense that they apply to any construction.

Semantically secure encryption with:	Is equivalent to digital lockers with:
α-weak keys	α-entropic security for indep messages
fully weak keys	fully entropic security, indep messages
auxiliary input	auxiliary input
CPA security	self-composability
KDM security	dependent messages

On the other hand, constructions for digital lockers satisfying the full virtual black-box requirement are only known under very strong assumptions, such as the existence of fully composable point circuit obfuscators [25]. We provide new evidence of the difficulty of obfuscation under standard assumptions by applying Haitner and Holenstein's [50] black-box impossibility result for KDM security to the digital locker setting.

Organization

In Section 4.2, we define the various encryption and digital locker obfuscation terms that are used throughout the chapter. Then, we provide the basic connection between symmetric encryption and obfuscation with weak keys in Section 4.3, and extend the connections to the auxiliary input and KDM settings in Sections 4.4 and 4.5. Finally, we discuss the implications of the connections in Section 4.6.

4.2 Definitions

In this section, we study many possible security guarantees that can be imposed on obfuscation of point circuits with multi-bit output and symmetric key encryption.

4.2.1 Obfuscation

Recall that a "point circuit with multi-bit output" $I_{(k,m)}$ stores a key k and message m in a readily identifiable manner and computes

$$I_{(k,m)}(x) = \begin{cases} m & \text{if } x = k, \\ \bot, 1^{|m|} & \text{otherwise.} \end{cases}$$

In particular, the circuit does not hide the length of the message. The family of all such circuits is $\mathcal{I} = \{I_{(k,m)} : k, m \in \{0,1\}^*\}$. We generalize the virtual black-box definition from Definition 2.4 by loosening the "for all" requirement, and impose a distributional guarantee instead.

Definition 4.1 (Generalized obfuscation[2] of $\mathcal{I}$)**.** An obfuscator of point circuits with multi-bit output is a PPT algorithm $\mathcal{O}$ which takes as input[3] a circuit $I_{(k,m)} \in \mathcal{I}$ and outputs a circuit that satisfies the following properties.

1. *Correctness*[4]: For all $k, x \in \{0,1\}^n$ and $m \in \{0,1\}^{\text{poly}(n)}$,

$$\Pr\left[C \leftarrow \mathcal{O}(I_{(k,m)}) : C(x) \neq I_{(k,m)}(x)\right] \leq \text{negl}(n),$$

where the probability is taken over the randomness of the obfuscator algorithm.

[2]Recall that A and S are allowed to be non-uniform throughout this thesis.

[3]Because k and m are extractable from the description of $I_{(k,m)}$, the obfuscator effectively receives them as input.

[4]This property is called "approximate functionality" in Definition 2.4, but in this chapter we purposely change terminology to parallel the terms used in symmetric key encryption.

2. *Polynomial slowdown*: For any k, m, and random tape of $\mathcal{O}$, the size of the circuit $\mathcal{O}(I_{(k,m)})$ is polynomial in $|k| + |m|$.

3. *Entropic security*: We say that the scheme has $\alpha(n)$-entropic security if for any PPT adversary A with binary output and any polynomial ℓ, there exists a negligible function ε and a PPT simulator S such that for all jointly-distributed $\{X_n, Y_n\}_{n\in\mathbb{N}}$ where X_n takes values in $\{0,1\}^n$, Y_n takes values in $\{0,1\}^{\ell(n)}$, and the min-entropy[5] $H_\infty(X_n) \geq \alpha(n)$, we have

$$\left|\Pr\left[A(\mathcal{O}(I_{(k,m)})) = 1\right] - \Pr\left[S^{I_{(k,m)}}(1^n, 1^\ell) = 1\right]\right| \leq \varepsilon(n),$$

where the probability is taken over the randomness of $(k,m) \leftarrow (X_n, Y_n)$, and the randomness of A, S, and $\mathcal{O}$. We say that a scheme has *fully-entropic security* if it has $\alpha(n)$-entropic security for all super-logarithmic functions $\alpha(n)$.[6]

Intuitively, the notion of α-entropic security generalizes (and weakens) the virtual black-box property by providing a minimum guarantee on the entropy of the key, whereas the virtual black-box property has no such guarantee. We connect the two notions through fully-entropic security, which is the strongest type of entropic security that we can hope for, since a key distribution of logarithmic entropy is trivially broken by a dictionary attack.

Informally, it appears that fully-entropic security and the virtual black-box property impose similar security guarantees under similar conditions, so they should be equivalent. However, the definitions have a few differences: the former allows a different simulator for each entropy threshold α, but requires a negligible error in simulation, while the latter allows a different simulator for each simulation error ρ but requires the simulator to work for all distributions regardless of entropy.

Despite these differences, we show that fully-entropic security implies the virtual black-box property, which connects our notion of α-entropic security to the existing notions of obfuscation.

Theorem 4.2. *If $\mathcal{O}$ is an obfuscator for $\mathcal{I}$ that satisfies fully-entropic security (as in Definition 4.1), then $\mathcal{O}$ also satisfies the virtual black-box property (as in Definition 2.4).*

The proof of this theorem is rather technical, and is deferred to Section 4.2.3. The idea is to extend the technique used in [23] to show that a distribution-based definition implies the virtual black-box property in the case of point circuits. At a high level, the distributional definition says that if a user chooses a key from a well-spread distribution, then an adversary cannot learn anything from an obfuscated point circuit beyond the fact that the key is from this distribution, so in particular the key is hard to determine. We show how to extend Canetti's distributional definition to the multi-bit setting and prove that fully-entropic security implies this distributional requirement, and therefore the virtual black-box property as well. The proof makes extensive use of non-uniformity, and as a result the simulator in the virtual black-box property must be non-uniform even if the adversary is uniform.

Fully-entropic security, as well as virtual black-box security, are quite strong, and difficult to satisfy. The notion of α-entropic security, for some particular $\alpha \in \omega(\log(n))$, corresponds to a meaningful weakening of that notion where security is only provided when the input comes from a reasonably random source. A similar weakening of obfuscation, in the special case of point circuits, was also considered by Canetti, Micciancio and Reingold [28] in the context of perfectly one-way hash functions.

[5] See Definition 2.2 for a formal definition of min-entropy.

[6] In the virtual black-box property (Definition 2.4), the simulator receives 1^n and an oracle to $I_{(k,m)}$, but it does not receive 1^ℓ as input. However, the simulator can easily learn 1^ℓ by querying its oracle on any input (say, 0^n) and noting the length of its response. Therefore, in this chapter we simply give the simulator 1^ℓ when convenient.

We pause here to address an important question: is α-entropic security the best way to generalize the virtual black-box definition? Instead of restricting attention to distribution with sufficient min-entropy, one might instead give the simulator the ability to ask its oracle more queries, by a factor of $2^{\alpha(n)}$ (i.e., the simulator is no longer polynomial time bounded). In Section 4.2.4, we show that this alternative relaxed notion is actually implied by α-entropic security, which is why we choose to study the entropic definition.

We consider several additional variants of obfuscation throughout the paper. First, we propose an additional weakening of the definition called security for independent messages, where we require that the distribution on the message m is independent from that of the key k.

Definition 4.3 (Independent messages)**.** We say that an obfuscator $\mathcal{O}$ is $\alpha(n)$-entropically secure *for independent messages* if we restrict the definition of $\alpha(n)$-entropic security only to distributions $\{X_n, Y_n\}$ where X_n and Y_n are independently distributed. We define the notion of *fully-entropic security for independent messages* analogously.

We also define a stronger variant of plain obfuscation that provides some composability guarantees. We specialize Definition 2.5 to the family $\mathcal{I}$. There are two variants: for full composition we require that the security of obfuscation is preserved even if the adversary gets many freshly and independently obfuscated circuits, which may be related in arbitrary ways (i.e., both the keys and the messages may differ).[7] For self-composition we require that all the obfuscated circuits have the same key k, although the messages may differ and may be correlated. (For point circuits, self-composition boils down to the case of many obfuscated versions of the same circuit.)

Definition 4.4. An obfuscator $\mathcal{O}$ for $\mathcal{I}$ with $\alpha(n)$-entropic security is said to be *fully composable* if for any PPT adversary A with binary output and any polynomials t and ℓ, there exists a PPT simulator S such that for all distributions $\{(X_n, Y_n)\}_{n\in\mathbb{N}}$, where $X_n = X_n^{(1)}, \ldots, X_n^{(t)}$ with $X_n^{(i)}$ taking values in $\{0,1\}^n$, $Y_n = Y_n^{(1)}, \ldots, Y_n^{(t)}$ with $Y_n^{(i)}$ taking values in $\{0,1\}^{\ell(n)}$, and $H_\infty(X_n) \geq \alpha(n)$, we have:

$$|\Pr[A(\mathcal{O}(I_{k_1,m_1}), \ldots, \mathcal{O}(I_{k_t,m_t})) = 1] - \Pr[S^{I_{(k_1,m_1)},\ldots,I_{(k_t,m_t)}}(1^n, 1^\ell) = 1]| \leq \mathrm{negl}(n),$$

where the probabilities are over $(k_1, \ldots, k_t, m_1, \ldots, m_t) \leftarrow (X_n, Y_n)$ and over the randomness of A, S, and $\mathcal{O}$.

If the above holds only for the distributions X_n where $\Pr[k_1 = k_2 \ldots = k_t] = 1$, then we say that $\mathcal{O}$ is *self-composable*.

The notions of composability extend naturally to obfuscators with fully-entropic security, where we require that the above definition holds for all $\alpha(n) \in \omega(\log(n))$. It also extends to obfuscators for independent messages, where we restrict the definition to the case where X_n and Y_n are independent. (However, there is no independence assumption among the coordinates within X_n or Y_n.)

4.2.2 Encryption with weak keys

A symmetric encryption scheme is defined by efficient algorithms $(\mathrm{Enc}, \mathrm{Dec})$. Technically, a symmetric key encryption scheme requires a key generation algorithm as well, but in this chapter we consider schemes that withstand deviations from the key generation protocol, so the specific algorithm is not important to us.[8]

[7] Recall that composability is not, in general, implied by obfuscation alone [12, 25].

[8] For the sake of completeness, throughout this work we may assume that the key generation algorithm chooses an n-bit string uniformly at random.

We say that the encryption scheme is semantically secure for $\alpha(n)$-weak keys if the usual notion of semantic security holds even when the key comes from any weak source of entropy $\alpha(n)$. We propose the following definition of symmetric key encryption with weak keys.

Definition 4.5 (Symmetric encryption with weak keys)**.** We say that an encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ has *CPA security for $\alpha(n)$-weak keys* if there exists an efficient algorithm $D(n, \ell)$ running in time $\mathrm{poly}(n, \ell)$ such that, for all PPT adversaries A and all distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$, we have:

$$\left|\Pr[\mathrm{CPA}_0^{X,D}(A, n) = 1] - \Pr[\mathrm{CPA}_1^{X,D}(A, n) = 1]\right| \leq \mathrm{negl}(n)$$

where the games $\mathrm{CPA}_b^{X,D}(A, n)$ for $b = 0, 1$ are defined via the following experiment:

1: $k \leftarrow X_n$
2: **repeat**
3: A submits a query m and receives a ciphertext c where:
in game $\mathrm{CPA}_0^{X,D}$ the challenger sets $c \leftarrow \mathrm{Enc}_k(m)$
in game $\mathrm{CPA}_1^{X,D}$ the challenger sets $c \leftarrow D(n, |m|)$
4: **until** A halts
5: **output** the result of A

The algorithm $D(n, \ell)$ can keep persistent state during the repeat loop. We define *semantic security with α-weak keys* via the games $\mathrm{SEM}_0^{X,D}$ and $\mathrm{SEM}_1^{X,D}$ that are identical to the CPA games except that the repeat loop is performed only once.

We say that an encryption scheme is CPA (resp., semantically) secure for *fully weak keys* if it is CPA (resp., semantically) secure for $\alpha(n)$-weak keys for all $\alpha(n) \in \omega(log(n))$.

Note that when $\alpha(n) = n$ (i.e., the key is chosen uniformly at random), the above definition is equivalent to the standard notion of CPA/semantic security, since we can simply define $D(n, \ell)$ to output fresh encryptions of the form $\mathrm{Enc}_k(0^\ell)$, where k is initially chosen uniformly at random and re-used for all queries. On the other hand, when considering $\alpha(n)$-weak keys, the above definition is somewhat stronger than just requiring that the adversary cannot distinguish between an encryption of m and that of some set message, such as 0^ℓ. In particular, it requires that there is a single *universal* distribution D on ciphertexts that is indistinguishable from encryption with *any* key distribution X_n of sufficient min-entropy. For example, consider an encryption scheme which, along with the ciphertext, always outputs the first bit of the secret key. Although such scheme might satisfy a natural definition whereby encryptions of m_0 and m_1 are indistinguishable, it could never satisfy the above definition, even for $\alpha(n) = n - 1$. The reason is that the ciphertext distribution is now different depending on whether the keys come from a distribution that fixes the first bit at 0 versus one which fixes the first bit at 1. Although our definition is stronger than one may need, we will show that it is necessary and sufficient for our equivalence with obfuscation to hold. Moreover, all natural constructions of encryption schemes with weak keys that we know of achieve the above definition.

We also define a "wrong-key detection" property, which will be needed to achieve correctness in obfuscation.

Definition 4.6. An encryption scheme satisfies the *wrong-key detection property* if for all $k \neq k' \in \{0, 1\}^n$ and $m \in \{0, 1\}^{\mathrm{poly}(n)}$, $\Pr[\mathrm{Dec}_{k'}(\mathrm{Enc}_k(m)) \neq \perp] \leq \mathrm{negl}(n)$.

We note that a similar, but weaker, property called confusion freeness was defined in [68]. For

confusion freeness, the keys k, k' are random and independent, while we consider a worst-case choice of k, k' so the probability above is only over the randomness of the encryption scheme.

Lemma 4.7 shows that, in the case of semantic security, wrong-key detection can always be achieved via a simple transformation. We note, however, that this transformation no longer works in the case of CPA security.

Lemma 4.7. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be a semantically-secure encryption scheme for* $\alpha(n)$*-weak keys and let* $\mathcal{H}$ *be a pairwise-independent permutation family. Define an encryption scheme* $(\mathrm{Enc}', \mathrm{Dec}')$ *by:*

$$\mathrm{Enc}'_k(m) \triangleq \begin{cases} \textit{Choose: } h \leftarrow \mathcal{H}, r \leftarrow U_n \\ \textit{Output: } \langle r, h, c = \mathrm{Enc}_{h(k)}(r \circ m)\rangle \end{cases}$$

$$\mathrm{Dec}'_k(\langle r, h, c\rangle) \triangleq \begin{cases} \textit{Compute: } (r' \circ m') = \mathrm{Dec}_{h(k)}(c) \\ \textit{Output: } m' \textit{ if } r' = r \textit{ and } \perp \textit{ otherwise,} \end{cases}$$

where $\circ$ *denotes string concatenation. Then,* $(\mathrm{Enc}', \mathrm{Dec}')$ *is a semantically secure encryption scheme for* $\alpha(n)$*-weak keys with wrong-key detection. The above also holds if we replace "*$\alpha(n)$*" with "fully."*

Proof. Let us first show that the modification preserves semantic security for $\alpha(n)$-weak keys. Let $D(n, \ell)$ be the distribution for which the semantic security of $(\mathrm{Enc}, \mathrm{Dec})$ is satisfied, and define $D'(n, \ell) = D(n, \ell + n)$. Then, for any adversary A attacking the modified scheme $(\mathrm{Enc}', \mathrm{Dec}')$, and any distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ we have

$$\begin{aligned} &|\Pr[\mathrm{SEM}_0(A, n) = 1] - \Pr[\mathrm{SEM}_1(A, n) = 1]| \\ &\quad = \big|\Pr\left[m \leftarrow A(1^n), r \leftarrow U_n h \leftarrow \mathcal{H}, k \leftarrow X_n, c \leftarrow \mathrm{Enc}_{h(k)}(r \circ m_0) : A(1^n, c) = 1\right] \\ &\qquad\qquad - \Pr\left[m \leftarrow A(1^n), c \leftarrow D'(n, m) = D(n, |m| + n) : A(1^n, c) = 1\right]\big| \\ &\quad \leq \max_{r\in\{0,1\}^n, h\in\mathcal{H}} |\Pr\left[m \leftarrow A(1^n), k \leftarrow h(X_n), c \leftarrow \mathrm{Enc}_k(r \circ m) : A(1^n, c) = 1\right] \\ &\qquad\qquad - \Pr\left[m \leftarrow A(1^n), c \leftarrow D(n, |m| + n) : A(1^n, c) = 1\right]| \end{aligned}$$

which is negligible due to the semantic security of the original $(\mathrm{Enc}, \mathrm{Dec})$ scheme and the fact that, for any fixed permutation h, the distribution $h(X_n)$ has the same entropy as X_n.

Now we show that the modified scheme has wrong-key detection. Assume otherwise, that there is some polynomial ρ and infinitely many values n for which there exists $k \neq k' \in \{0,1\}^n$, $m \in \{0,1\}^{\mathrm{poly}(n)}$ such that

$$\begin{aligned} \Pr[\mathrm{Dec}'_{k'}(\mathrm{Enc}_k(m)) \neq \perp] &= \Pr_{h\leftarrow\mathcal{H}, r\leftarrow U_n}[\mathrm{Dec}_{h(k')}(\mathrm{Enc}_{h(k)}(r \circ m)) \text{ has a prefix } r] \\ &= \Pr_{k\leftarrow U_n, k'\leftarrow U_n, r\leftarrow U_n}[\mathrm{Dec}_{k'}(\mathrm{Enc}_k(r \circ m)) \text{ has a prefix } r] \geq \rho(n). \end{aligned}$$

Now we show that the final inequality contradicts the semantic security of $(\mathrm{Enc}, \mathrm{Dec})$ (even for uniform keys). In particular, consider an adversary A which queries the challenger on the messages $m* = r \circ m$, for a random r and for the m which contradicts correctness and satisfies the inequality. On input c, the adversary A picks a random $k' \leftarrow U_n$ and outputs 1 iff $\mathrm{Dec}_{k'}(c)$ begins with r. By the inequality, we see that in the semantic-security game SEM_0 (where the encryption is of the message m) A outputs 1 with probability $\rho(n)$. On the other hand, in SEM_1, no matter what distribution D the challenger samples from, the result is independent of r and therefore, the probability that A outputs 1 is the probability that $\mathrm{Dec}_{k'}(\mathrm{Enc}_k(m_0))$ begins with r, which is at most $1/2^n$. Therefore the adversary A has a non-negligible advantage in the semantic-security game for any D, which gives a contradiction. □

4.2.3 Proof of Theorem 4.2

Now we return to the two technical statements about α-entropic security that were described in Section 4.2.1: proving Theorem 4.2 and comparing different $\alpha(n)$ security properties. Before proceeding, we caution that these details are rather technical in nature. A reader who is uninterested in these technical details (and willing to take our assertions about α-entropic security in Section 4.2.1 on faith) in may safely skip to Section 4.3.

With that disclaimer out of the way, in this section we prove Theorem 4.2, which states that an obfuscator $\mathcal{O}$ with fully-entropic security satisfies the virtual black-box property in Definition 2.4. Note that rather than stating the virtual black-box property must hold for all circuits (that is, all k and m), we could instead require that it hold for all distributions X_n and Y_n without changing the strength of the guarantee. With this change in place, the virtual black-box definition appears to be stronger than fully-entropic security because it does not impose any constraint on the min-entropy of the distribution X_n. However, we show in this section that this is not the case.

We do not work with the virtual black-box definition directly, but rather use two intermediate definitions. These definitions were presented in the survey for the case of point circuits (Definitions 3.16 and 3.17), and we extend them here to the multi-bit setting.

Definition 4.8 (Distributional indistinguishability [23])**.** For any PPT adversary A with binary output and for any distribution $\{X_n, Y_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \in \omega(\log(n))$, the distributions

$$\langle k, m, A(\mathcal{O}(I_{(k,m)}))\rangle \text{ and } \langle k, m, A(\mathcal{O}(I_{(k',m')}))\rangle$$

are computationally indistinguishable, where $(k, m), (k', m') \leftarrow \{X_n, Y_n\}$ independently.

Definition 4.9 (Oracle indistinguishability [23])**.** For any PPT adversary A with binary output and any polynomial ρ, there exists a polynomial-sized family of sets of keys $\{L_n\}_{n\in\mathbb{N}}$ such that for all sufficiently large n, all $k, k' \notin L_n$ and all m, m',

$$\left|\Pr[A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[A(\mathcal{O}(I_{(k',m')})) = 1]\right| < \frac{1}{\rho(n)}.$$

We prove Theorem 4.2 by showing that fully-entropic security implies the distributional indistinguishability property, which in turn implies oracle indistinguishability, which finally implies the virtual black-box property. The main ideas in these proofs are adapted from [23] to the multi-bit setting.

Lemma 4.10. *If an obfuscator $\mathcal{O}$ satisfies fully-entropic security, then it satisfies distributional indistinguishability.*

Proof. Suppose for the sake of contradiction that $\mathcal{O}$ does not satisfy distributional indistinguishability, so there exists an adversary A, distribution $\{X_n, Y_n\}$ with $H_\infty(X_n) \in \omega(\log(n))$, distinguisher D, and polynomial ρ such that for infinitely many values of n,

$$\left|\Pr[D(k, m, A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[D(k, m, A(\mathcal{O}(I_{(k',m')})) = 1]\right| \geq \frac{1}{\rho(n)},$$

where (k, m) and (k', m') are independently sampled from $\{X_n, Y_n\}$. Let $\beta(n) = H_\infty(X_n)$. We show that the same adversary A breaks the fully-entropic security of $\mathcal{O}$ because it breaks α-entropic security for $\alpha(n) = \beta(n) - \log(10\rho(n))$. Note that $\alpha \in \omega(\log(n))$ as desired.

Define $\gamma_{k,m} = \Pr[A(\mathcal{O}(I_{(k,m)})) = 1]$, where the probability is taken over the randomness of A and $\mathcal{O}$. It follows from the above inequality that there exist two sets $Z_1, Z_0 \subset \{0,1\}^n$ such that:

- For any $(k,m) \in Z_1$, $(k',m') \in Z_0$ we have $\gamma_{k,m} - \gamma_{k',m'} > \frac{1}{10\rho(n)}$.
- The sets are fairly large: for $(k,m) \leftarrow \{X_n, Y_n\}$, $\Pr[(k,m) \in Z_1] = \Pr[(k,m) \in Z_0] = \frac{1}{10\rho(n)}$.

Let $\{X_n^1, Y_n^1\}$ and $\{X_n^0, Y_n^0\}$ be the distributions formed by taking $\{X_n, Y_n\}$ and conditioning on the event that a key-message pair is chosen from Z_1 or Z_0, respectively. We claim that the two distributions $\{X_n^b, Y_n^b\}$ for $b \in \{0,1\}^n$ each have min-entropy α. This holds because for any k, m,

$$\Pr[\{X_n^b, Y_n^b\} = (k,m)] \leq \Pr[\{X_n, Y_n\} = (k,m)] \cdot 10\rho(n)$$

since equality holds if $(k,m) \in Z_b$ and the left side probability equals 0 if $(k,m) \notin Z_b$. Therefore, by the union bound,

$$\Pr[X_n^b = k] \leq \Pr[X_n = k] \cdot 10\rho(n) \leq 2^{-\beta} \cdot 10\rho(n) = 2^{-\alpha}.$$

As a result, given any PPT simulator $S^{I_{(k,m)}}$ where (k,m) is chosen from either $\{X_n^1, Y_n^1\}$ or $\{X_n^0, Y_n^0\}$, the simulator only queries the correct key k with negligible probability. Hence,

$$\Pr[(k,m) \leftarrow \{X_n^1, Y_n^1\} : S^{I_{(k,m)}}(1^n) = 1] - \Pr[(k,m) \leftarrow \{X_n^0, Y_n^0\} : S^{I_{(k,m)}}(1^n) = 1]$$

is negligible. On the other hand, we know that

$$\Pr[(k,m) \leftarrow \{X_n^1, Y_n^1\} : A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[(k,m) \leftarrow \{X_n^0, Y_n^0\} : A(\mathcal{O}(I_{(k,m)})) = 1] > \frac{1}{10\rho(n)}.$$

As a result, it follows by the triangle inequality that for any simulator S, either

$$\Pr[(k,m) \leftarrow \{X_n^1, Y_n^1\} : A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[(k,m) \leftarrow \{X_n^1, Y_n^1\} : S^{I_{(k,m)}}(1^n) = 1] > \frac{1}{20\rho(n)},$$

or the corresponding inequality holds for $\{X_n^0, Y_n^0\}$. Hence, one of these distributions breaks the α-entropic security of $\mathcal{O}$ and thus the fully-entropic security of $\mathcal{O}$, as desired. □

Lemma 4.11. *If an obfuscator $\mathcal{O}$ satisfies distributional indistinguishability, then it satisfies oracle indistinguishability.*

Proof. Assume for the sake of contradiction that there exist a PPT adversary A and polynomial ρ that break oracle indistinguishability. We define the following constants:

$$\gamma_{k,m} = \Pr[A(\mathcal{O}(I_{(k,m)})) = 1],\ m_k^1 = \arg\max_m\{\gamma_{k,m}\},\ m_k^0 = \arg\min_m\{\gamma_{k,m}\},\ \gamma_k^1 = \gamma_{k,m_k^1},\ \gamma_k^0 = \gamma_{k,m_k^0}.$$

Because A and ρ break oracle indistinguishability, for any polynomial-sized family of sets $\{L_n\}_{n\in\mathbb{N}}$ and for infinitely many n, there exist $k, k' \notin L_n$ and m, m' such that $\gamma_{k,m} - \gamma_{k',m'} \geq \frac{1}{\rho(n)}$. Without loss of generality, we can assume $m = m_k^1$ and $m' = m_{k'}^0$, since $\gamma_{k,m_k^1} \geq \gamma_{k,m}$ and $\gamma_{k',m_{k'}^0} < \gamma_{k',m'}$.

Hence, for any polynomial-sized family of sets $\{L_n\}_{n\in\mathbb{N}}$ and for infinitely many values of n, there exist $k, k' \notin L_n$ such that

$$\gamma_k^1 - \gamma_{k'}^0 \geq \frac{1}{\rho(n)}. \tag{4.1}$$

For a given constant $c \in \mathbb{N}$, construct the family of sets $\{L_n^c\}_{n\in\mathbb{N}}$ in the following manner. The set $L_n^c = S_n^c \cup T_n^c$, where S_n^c is the set of the n^c keys k with the maximal values of γ_k^1, and T_n^c is

the set of n^c keys k' with the minimal $\gamma^0_{k'}$. Clearly, $|L^c_n| \leq |S^c_n| + |T^c_n| = 2n^c$ so the family $\{L^c_n\}$ is polynomially-bounded in size. Hence, for any $c \in \mathbb{N}$, and for all n such that equation (4.1) holds for the family $\{L^c_n\}$, we have that any keys $k \in S^c_n$ and $k' \in T^c_n$ satisfy $\gamma^1_k - \gamma^0_{k'} \geq \frac{1}{\rho(n)}$.

Next, we form the families $\{\hat{S}_n\}_{n\in\mathbb{N}}$ and $\{\hat{T}_n\}_{n\in\mathbb{N}}$ as follows. Given n, let c_n be the largest value such that equation (4.1) is satisfied with respect to n and $L^{c_n}_n$. Then, $\hat{S}_n$ is defined recursively, as follows.

1. The base case is $\hat{S}_0 = S^{c_0}_0$.
2. For $n > 0$, let n' be such that $\hat{S}_{n-1} = S^{c_{n'}}_{n-1}$. Then, $\hat{S}_n$ equals the largest set out of $S^{c_n}_n$ and $S^{c_{n'}}_n$.

We define $\hat{T}_n$ analogously. Finally, we form the distribution $\{X_n, Y_n\}$ that is uniform over the key-message pairs (k, m^1_k) for all $k \in \hat{S}_n$ and the key-message pairs $(k', m^0_{k'})$ for all $k' \in \hat{T}_n$. This distribution is well-spread, because given any polynomial n^d, there exists a value n_0 such that $|\hat{S}_n| = |\hat{T}_n| > n^d$ for all $n > n_0$.

We show that there exists a distinguisher D such that for infinitely many values of n,

$$\Pr[D(k, m, A(\mathcal{O}(I_{(k,m)}))) = 1] - \Pr[D(k, m, A(\mathcal{O}(I_{(k',m')}))) = 1] \geq \frac{1}{3\rho(n)}, \tag{4.2}$$

where (k, m) and (k', m') are independently drawn from $\{X_n, Y_n\}$.

We construct the distinguisher D as follows. Let $\hat{\gamma}_n$ be a constant such that $\gamma^1_k - \hat{\gamma}_n \geq \frac{1}{2\rho(n)}$ for all $k \in \hat{S}_n$ and $\hat{\gamma}_n - \gamma^0_k \geq \frac{1}{2\rho(n)}$ for all $k \in \hat{T}_n$. This is known to D by non-uniformity.[9] The distinguisher receives as input a key k, message m, and bit b. It estimates $\gamma_{k,m}$ by sampling $A(\mathcal{O}(I_{(k,m)}))$ for many independent choices of the randomness for A and $\mathcal{O}$. If its estimate of $\gamma_{k,m}$ is at least $\hat{\gamma}_n$, then D outputs b. Otherwise, it outputs $1 - b$.

We demonstrate that the distinguisher D satisfies equation (4.2) for all n such that $\hat{S}_n = S^c_n$ for some c. There are infinitely many such n's. Our distribution has the property that for $(k, m) \leftarrow \{X_n, Y_n\}$, the value $\gamma_{k,m}$ is bigger than $\hat{\gamma}_n$ with probability $\frac{1}{2}$ and is smaller with probability $\frac{1}{2}$. Also, the distinguisher will make the correct determination on whether $\gamma_{k,m}$ is bigger or smaller than $\hat{\gamma}_n$ with overwhelming probability.

The distinguisher receives as input k, m, and a bit $b = A(\mathcal{O}(I_{(k',m')}))$, where (k', m') either equals (k, m) or is an independent sample from $\{X_n, Y_n\}$. Some basic probability calculations show that

$$\Pr[D = 1] \begin{cases} \geq \frac{1}{2} + \frac{1}{2\rho(n)} - \text{negl}(n) & \text{when } \gamma_{k,m} \text{ and } \gamma_{k',m'} \text{ are both larger or both smaller than } \hat{\gamma}_n, \\ \leq \frac{1}{2} - \frac{1}{2\rho(n)} + \text{negl}(n) & \text{when } \gamma_{k,m} \text{ and } \gamma_{k',m'} \text{ are separated by } \hat{\gamma}_n, \end{cases}$$

where this probability is taken over $(k, m) \leftarrow \{X_n, Y_n\}$ and the randomness of D, A, and $\mathcal{O}$. When $(k', m') = (k, m)$, the first case always holds, and when (k', m') is an independent sample, the two cases each hold with probability $\frac{1}{2}$. Therefore,

$$\Pr[D = 1 : (k', m') = (k, m)] - \Pr[D = 1 : (k', m') \leftarrow \{X_n, Y_n\}] \geq \frac{1}{2\rho(n)} - \text{negl}(n),$$

as desired. □

[9] Recall that throughout this thesis, adversaries and simulators are allowed to be non-uniform. Here, the distinguisher is an adversary that "attacks" the notion of distributional indistinguishability, so we allow it to be non-uniform as well. In fact, non-uniformity is critical to this proof.

Lemma 4.12. *If an obfuscator $\mathcal{O}$ satisfies oracle indistinguishability, then it satisfies the virtual black-box property.*

Proof. Assume that oracle indistinguishability holds. Let A be a PPT adversary with binary output and let $\{L_n\}$ be the polynomial-sized family of sets associated to A. We form a simulator $S^{I_{(k,m)}}$ that queries its oracle on all of the keys in L_n. If $k \in L_n$, then the simulator learns k and m, and it emulates an execution of $A(\mathcal{O}(I_{(k,m)}))$. In this case, its simulation is perfect.

Otherwise, the simulator can run $A(\mathcal{O}(I_{(k',m')}))$ for any $k' \notin L_n$ and any m'. By α-oracle indistinguishability,

$$\Pr[S^{I_{(k,m)}}(1^n) = 1] = \Pr[A(\mathcal{O}(I_{(k',m')})) = 1] \approx \Pr[A(\mathcal{O}(I_{(k,m)})) = 1]$$

where the $\approx$ denotes a negligible difference in probability. Finally, the simulator's runtime is bounded by the size of L_n and the runtime of A, so S runs in polynomial time as desired. □

The three lemmas together prove Theorem 4.2.

4.2.4 Comparison of α-obfuscation definitions

In this section, we discuss the definition of α-entropic security. The goal of entropic security is to weaken the virtual black-box property in Definition 2.4, which is very strong and thus far can only be satisfied under non-standard assumptions such as a strong variant of the DDH assumption [23], an exponentially hard to invert one-way function [93], or the random oracle model [63].[10]

We believe there are two natural ways to weaken the virtual black-box property. First, we can increase the min-entropy requirement, as α-entropic security does. Second, we can give the simulator a super-polynomial runtime so it can make more queries to its oracle.

In this work, we choose to do the former. This section justifies that decision by showing that α-entropic security implies a virtual black-box style definition in which the simulator receives a boost to its running time.

Definition 4.13 (α-runtime security)**.** For any PPT adversary A with binary output, there exists a negligible function $\varepsilon(n)$ and a simulator S running in time $\varepsilon(n) \cdot 2^{\alpha(n)}$ such that for all distributions $\{X_n, Y_n\}_{n\in\mathbb{N}}$ with X_n taking values in $\{0,1\}^n$ and Y_n taking values in $\{0,1\}^{\mathrm{poly}(n)}$, we have:

$$\left|\Pr\left[A(\mathcal{O}(I_{(k,m)})) = 1\right] - \Pr\left[S^{I_{(k,m)}}(1^n)\right]\right| \leq \mathrm{negl}(n),$$

where the probability is taken over the randomness of $(k, m) \leftarrow (X_n, Y_n)$, A, S, and $\mathcal{O}$.

In this definition, we choose to give the simulator $\mathrm{negl}(n) \cdot 2^{\alpha(n)}$ running time so that it does not quite have enough time to query everything in the support of a distribution with min-entropy $\alpha(n)$, but other than this restriction S has the "largest" runtime possible.

Theorem 4.14. *If an obfuscator satisfies α-entropic security, then it satisfies α-runtime security.*

The rest of this section is devoted to a proof of this theorem. We do not prove the theorem directly, but rather go through an intermediate definition from [23].

Definition 4.15 (α-oracle indistinguishability [23])**.** For any PPT adversary A with binary output, there exists a negligible function $\varepsilon(n)$ and a family of sets $\{L_n\}_{n\in\mathbb{N}}$ such that $|L_n| \leq \varepsilon(n) \cdot 2^{\alpha(n)}$ and for all $k, k' \notin L_n$ and all m, m',

$$\left|\Pr[A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[A(\mathcal{O}(I_{(k',m')})) = 1]\right| < \mathrm{negl}(n).$$

[10] These constructions are all described in Section 3.2.1.

Lemma 4.16. *If an obfuscator satisfies α-oracle indistinguishability, then it satisfies α-runtime security.*

The proof of this lemma is identical to that of Lemma 4.12.

Proof. Let A be a PPT adversary with binary output. We form a simulator $S^{I_{(k,m)}}$ that queries its oracle on all of the keys in L_n. If $k \in L_n$, then the simulator learns k and m, and it emulates an execution of $A(\mathcal{O}(I_{(k,m)}))$. In this case, its simulation is perfect.

Otherwise, the simulator can run $A(\mathcal{O}(I_{(k',m')}))$ for any $k' \notin L_n$ and any m'. By α-oracle indistinguishability,

$$\Pr[S^{I_{(k,m)}}(1^n) = 1] = \Pr[A(\mathcal{O}(I_{(k',m')})) = 1] \approx \Pr[A(\mathcal{O}(I_{(k,m)})) = 1],$$

where the $\approx$ denotes a negligible difference in probability, as desired. □

Lemma 4.17. *If an obfuscator satisfies α-entropic security, then it satisfies α-oracle indistinguishability.*

Proof. Let $\mathcal{O}$ be an obfuscator satisfying α-entropic security, and let A be an adversary. We wish to show the existence of a negligible function ε and family of sets $\{L_n\}_{n\in\mathbb{N}}$ that satisfy α-oracle indistinguishability.

Given any k and m, we define $\gamma_{k,m} = \Pr[A(\mathcal{O}(I_{(k,m)})) = 1]$. Also, we define the following constants:

$$\mu_k = \operatorname*{average}_m\{\gamma_{k,m}\}, \quad m_k^1 = \arg\max_m\{\gamma_{k,m}\}, \quad m_k^0 = \arg\min_m\{\gamma_{k,m}\}, \quad \sigma_k = \gamma_{k,m_k^1} - \gamma_{k,m_k^0}$$

(Note that if the arg max or arg min are simultaneously fulfilled by many messages, then it suffices to pick any one of them arbitrarily.) Also, let S be the PPT simulator associated with A by α-entropic security. Clearly, $S^{I_{(k,m)}}$ does not learn any information about m unless it queries the correct key k, which it can only do for polynomially many keys.

By α-entropic security, it follows that there exists a negligible function ε' such that at most $\varepsilon' \cdot 2^\alpha$ keys have a non-negligible σ_k. If this were not the case, then there exists some polynomials ρ, ξ such that there are at least $\frac{2^{\alpha(n)}}{\rho(n)}$ keys k with $\sigma_k > \frac{1}{\xi(n)}$. Let X_n be the distribution that is uniform over these $\frac{2^{\alpha(n)}}{\rho(n)}$ keys and $2^{\alpha(n)} - \frac{2^{\alpha(n)}}{\rho(n)}$ other keys chosen arbitrarily, and let Y_n^1 and Y_n^0 be two distributions on messages such that for a given key k, the distribution Y_n^b always chooses the message m_k^b. Both distributions $\{X_n, Y_n^b\}$ have min-entropy α, and we know that

$$\Pr[(k,m) \leftarrow \{X_n, Y_n^1\} : A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[(k,m) \leftarrow \{X_n, Y_n^0\} : A(\mathcal{O}(I_{(k,m)})) = 1] > \frac{1}{\rho(n)\xi(n)}.$$

On the other hand, the simulator cannot distinguish between these two distributions, so by the triangle inequality property, either

$$\Pr[(k,m) \leftarrow \{X_n, Y_n^1\} : A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr[(k,m) \leftarrow \{X_n, Y_n^1\} : S^{I_{(k,m)}}(1^n) = 1] > \frac{1}{2\rho(n)\xi(n)},$$

or the corresponding inequality holds for $\{X_n, Y_n^0\}$, which breaks the α-entropic security.

Hence, there are at most $\varepsilon' \cdot 2^\alpha$ keys k such that σ_k is non-negligible. Let L_n' be the set of these keys. Next, we look at μ_k, and claim that there exists a negligible function ε'' and a set L_n'' of size at most $\varepsilon'' \cdot 2^\alpha$ such that for all $k, k' \notin L_n''$, $|\mu_k - \mu_{k'}|$ is negligible.

If this were not the case, then at least $\frac{2^\alpha}{\text{poly}(n)}$ keys have μ_k that are noticeably separated from $\mu = \text{average}_k\{\mu_k\}$. Using a similar proof to the one for σ_k above, we can then form two well-spread distributions over these keys such that the adversary can distinguish them but the simulator cannot.

Finally, let $\varepsilon = \varepsilon' + \varepsilon''$ and let $\{L_n\}_{n\in\mathbb{N}}$ be a family of sets with $L_n = L'_n \cup L''_n$. For all keys $k, k' \notin L_n$, we know that σ_k, $|\mu_k - \mu'_k|$, and $\sigma_{k'}$ are negligible, which means that for all messages m and m',

$$\Pr[A(\mathcal{O}(I_{(k,m)})) = 1] \approx \mu_k \approx \mu'_k \approx \Pr[A(\mathcal{O}(I_{(k',m')})) = 1],$$

where $\approx$ denotes a negligible difference in probability, as desired. □

The two lemmas combine to prove Theorem 4.14.

4.3 Encryption with weak keys and obfuscation with independent messages

4.3.1 Semantically secure encryption

In this section, we show equivalence between semantically secure encryption with weak keys and obfuscation of the family of point circuits with multi-bit output $\mathcal{I}$ for independent messages.

Theorem 4.18. *Let $\alpha(n) \in \omega(\log(n))$. There exist obfuscators for $\mathcal{I}$ with $\alpha(n)$-entropic security for independent messages if and only if there exist semantically secure encryption schemes with wrong-key detection for $\alpha(n)$-weak keys. Furthermore, the above also holds if we replace "$\alpha(n)$" with "fully."*

We prove the "if" and "only if" directions in Lemmas 4.19 and 4.20, respectively.

Lemma 4.19. *Let $\alpha(n) \in \omega(\log(n))$ and let $\mathcal{O}$ be an obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security for independent messages. Let $\text{Enc}_k(m) \triangleq \mathcal{O}(I_{(k,m)})$, $\text{Dec}_k(C) \triangleq C(k)$ where the ciphertext C is interpreted as a circuit. Then the encryption scheme (Enc, Dec) is semantically secure with $\alpha(n)$-weak keys and has the wrong-key detection property.*

Proof. The correctness of decryption follows from the correctness of obfuscation, so it suffices to prove the security of the encryption scheme with $\alpha(n)$-weak keys. Fix any adversary A and any distribution $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$. The distribution $\{Y_n\}$ is defined by running $A(1^n)$ and outputting the message m that A gives to its challenger. Define the distribution $D(n,\ell) = \mathcal{O}(I_{(k,m)})$ where $(k,m) \leftarrow (U_n, U_\ell)$. Then, by the $\alpha(n)$-entropic security of obfuscation, there exists a simulator S such that

$$\begin{aligned}
&\left|\Pr[\text{SEM}_0^{X,D}(A,n) = 1] - \Pr[\text{SEM}_1^{X,D}(A,n) = 1]\right| \\
&\quad= \left|\Pr_{(k,m)\leftarrow(X_n,Y_n)}[A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr_{(k,m)\leftarrow(U_n,U_\ell)}[A(\mathcal{O}(I_{(k,m)})) = 1]\right| \\
&\quad\leq \left|\Pr_{(k,m)\leftarrow(X_n,Y_n)}[A(\mathcal{O}(I_{(k,m)})) = 1] - \Pr_{(k,m)\leftarrow(X_n,Y_n)}[S^{I_{(k,m)}}(1^n,1^\ell) = 1]\right| && (4.3) \\
&\qquad+ \left|\Pr_{(k,m)\leftarrow(X_n,Y_n)}[S^{I_{(k,m)}}(1^n,1^\ell) = 1] - \Pr_{(k,m)\leftarrow(U_n,U_\ell)}[S^{I_{(k,m)}}(1^n,1^\ell) = 1]\right| && (4.4) \\
&\qquad+ \left|\Pr_{(k,m)\leftarrow(U_n,U_\ell)}[S^{I_{(k,m)}}(1^n,1^\ell) = 1] - \Pr_{(k,m)\leftarrow(U_n,U_\ell)}[A(\mathcal{O}(I_{(k,m)})) = 1]\right| && (4.5)
\end{aligned}$$

by the triangle inequality. Finally, (4.3) and (4.5) are negligible by the entropic security of obfuscation, and (4.4) is negligible because the only way that a PPT simulator can get anything from its oracle is by querying it on the input k, which happens with negligible probability when k comes from a source of super-logarithmic entropy $\alpha(n)$. □

Lemma 4.20. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be an encryption scheme with semantic security for* $\alpha(n)$*-weak keys and with the wrong-key detection property. We define the obfuscator* $\mathcal{O}$ *for* $\mathcal{I}$ *which, on input* $I_{(k,m)}$*, computes a ciphertext* $c = \mathrm{Enc}_k(m)$ *and outputs the circuit* C_c *that has* c *hard-coded and is defined by* $C_c(x) \triangleq \mathrm{Dec}_x(c)$*. Then,* $\mathcal{O}$ *has* $\alpha(n)$*-entropic security for independent messages.*

Proof. First, we show the correctness property of the obfuscator. Fix $k, x \in \{0,1\}^n$ and $m \in \{0,1\}^{\mathrm{poly}(n)}$. If $k = x$, then

$$\Pr\left[C \leftarrow \mathcal{O}(I_{(k,m)}) : C(x) \neq I_{(k,m)}(x)\right] = \Pr\left[\mathrm{Dec}_k(\mathrm{Enc}_k(m)) \neq m\right] \leq \mathrm{negl}(n)$$

by the correctness of encryption. On the other hand, if $k \neq x$ then

$$\Pr\left[C \leftarrow \mathcal{O}(I_{(k,m)}) : C(x) \neq I_{(k,m)}(x)\right] = \Pr\left[\mathrm{Dec}_x(\mathrm{Enc}_k(m)) \neq \bot\right] \leq \mathrm{negl}(n)$$

by the wrong-key detection of encryption.

The polynomial slowdown property of the obfuscator follows from the fact that the size of the circuit is only proportional to the ciphertext size and the size of the decryption circuit, which are polynomial in $|k|$ and $|m|$.

Lastly, we show $\alpha(n)$-entropic security for independent messages. Let $D(n, \ell)$ be the distribution defined by the semantic security of the encryption scheme. For any PPT adversary A, we define the simulator S which chooses a random ciphertext c from the distribution $D(n, \ell(n))$ and runs A on a circuit C_c constructed using the ciphertext c. Then,

$$\begin{aligned}&\left|\Pr\left[(k,m) \leftarrow (X_n, Y_n) : A(\mathcal{O}(I_{(k,m)})) = 1\right] - \Pr\left[(k,m) \leftarrow (X_n, Y_n) : S^{I_{(k,m)}}(1^n, 1^\ell) = 1\right]\right| \\ &\quad = \left|\Pr\left[(k,m) \leftarrow (X_n, Y_n),\, c \leftarrow \mathrm{Enc}_k(m) : A(C_c) = 1\right] - \Pr\left[c \leftarrow D(n,\ell) : A(C_c) = 1\right]\right|,\end{aligned}$$

which is negligible by semantic security. □

4.3.2 CPA encryption and self-composable obfuscation

In this section, we show equivalence between CPA secure encryption with weak keys and self-composable obfuscators for independent messages.

Theorem 4.21. *Let* $\alpha(n) \in \omega(\log(n))$*. There exist self-composable obfuscators for* $\mathcal{I}$ *with* $\alpha(n)$*-entropic security for independent messages if and only if there exist CPA secure encryption schemes for* $\alpha(n)$*-weak keys and the wrong-key detection property. The above also holds if we replace "*$\alpha(n)$*" with "fully."*

Again, we prove the two sides of the "if and only if" separately. Going from obfuscation to encryption, it would be natural to define $\mathrm{Enc}_k(m) = \mathcal{O}(I_{(k,m)})$. However, we instead define $\mathrm{Enc}_k(m) = (\mathcal{O}(I_{(k,r)}), m \oplus r)$ for a uniformly chosen r. The reason for this is that the messages m chosen by the adversary in the CPA game can depend adaptively on prior ciphertexts. However, for composable obfuscation, the distributions Y_i of the messages m_i are independent of prior obfuscated circuits. We get around this by making sure that the obfuscation is applied to a random value.

Lemma 4.22. *Let $\alpha(n) \in \omega(\log(n))$ be an arbitrary function. Let $\mathcal{O}$ be a self-composable obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security for independent messages. We define an encryption scheme by*

$$\mathrm{Enc}_k(m) \triangleq (\mathcal{O}(I_{(k,r)}), m \oplus r), \qquad \mathrm{Dec}_k(C, y) \triangleq C(k) \oplus y,$$

where r is uniformly random and C is interpreted as a circuit. This encryption scheme is CPA secure with $\alpha(n)$-weak keys.

Proof. The correctness of decryption and the wrong-key detection property follow from the correctness of obfuscation. For the CPA security of the encryption scheme with $\alpha(n)$-weak keys, we define the distribution $D(n, \ell)$ that chooses a uniformly random $k \leftarrow U_n$ in the beginning, and then on each invocation, outputs $(r, \mathcal{O}(I_{(k,r')}))$ for uniformly random and independent $r, r' \leftarrow U_\ell$. We need to show that for all PPT adversaries A and all distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$, we have

$$\left|\Pr[\mathrm{CPA}_0^{X,D}(A, n) = 1] - \Pr[\mathrm{CPA}_1^{X,D}(A, n) = 1]\right| \leq \mathrm{negl}(n) \tag{4.6}$$

for the CPA attack game defined in Definition 4.5.

Fix a PPT adversary A and let t be an upper bound on the number of queries that A sends to its encryption oracle (including the challenge query). There exist some values of the random coins $(r_1, \ldots, r_t)$ used by the encryption algorithm during the computation of ciphertexts $(r_i \oplus m, I_{(k,m)})$ that maximizes the difference in (4.6). Set the distributions $Y_1^{(0)}, \ldots, Y_{t+1}^{(0)}$ to the singletons $r_1, \ldots, r_t$ and set $Y_1^{(1)}, \ldots, Y_t^{(1)}$ to be uniform on $\{0,1\}^\ell$.

We define an adversary $A'_{(r_1,\ldots,r_t)}(C_1, \ldots, C_t)$ that attacks the obfuscation scheme as follows: A' simulates the CPA game with A so that, whenever A queries its oracle on messages m_i or asks for a challenge ciphertext (for $i = 1, \ldots, t$), the adversary A' responds with $(C_i, r_i \oplus m_i)$. Notice that when $C_1, \ldots, C_t$ are obfuscations of points $r_1, \ldots, r_t \leftarrow Y_1^{(0)}, \ldots, Y_{t+1}^{(0)}$ under a random $k \leftarrow X_n$, then the above simulation is equivalent to $\mathrm{CPA}_0^{X,D}$. On the other hand, when $C_1, \ldots, C_t$ are obfuscations of uniformly random $r_1, \ldots, r_t \leftarrow Y_1^{(1)}, \ldots, Y_{t+1}^{(1)}$ under a uniformly random key $k \leftarrow U_\ell$, then the above is equivalent to $\mathrm{CPA}_1^{X,D}$. Therefore,

$$\begin{aligned}
&\left|\Pr\left[\mathrm{CPA}_0^{X,D}(A, n) = 1\right] - \Pr\left[\mathrm{CPA}_1^{X,D}(A, n) = 1\right]\right| \\
&\qquad \leq \Big|\Pr\left[C_i \leftarrow \mathcal{O}(I_{(k,r_i)}), (k, r_1, \ldots, r_t) \leftarrow X_n, Y_1^{(0)}, \ldots, Y_{t+1}^{(0)} : A'(C_1, \ldots, C_t) = 1\right] \\
&\qquad\quad - \Pr\left[C_i \leftarrow \mathcal{O}(I_{(k,r_i)}), (k, r_1, \ldots, r_t) \leftarrow U_n, Y_1^{(1)}, \ldots, Y_{t+1}^{(1)} : A'(C_1, \ldots, C_t) = 1\right]\Big|,
\end{aligned}$$

which is negligible because, by the definition of self-composable obfuscation, there is a simulator that simulates both sides of the difference equivalently. □

The other direction is shown via the same construction as in the case of semantic security.

Lemma 4.23. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be an encryption scheme with CPA security for $\alpha(n)$-weak keys and having the wrong-key detection property. We define the obfuscator $\mathcal{O}$ for $\mathcal{I}$ which, on input $I_{(k,m)}$, computes a ciphertext $c = \mathrm{Enc}_k(m)$ and outputs the circuit C_c defined by $C_c(x) = \mathrm{Dec}_x(c)$. Then, $\mathcal{O}$ is a self-composable obfuscator with $\alpha(n)$-entropic security for independent messages.*

Proof. The correctness and polynomial slowdown properties follow from the same argument as that in the proof of Lemma 4.20. It remains to show that $\mathcal{O}$ is self-composable with $\alpha(n)$-entropic security for independent messages.

For any PPT adversary A and polynomial t, we define the simulator S which, on input 1^n and 1^ℓ, chooses t random ciphertexts $c_1, \ldots, c_t$ from the distribution $D(n, \ell)$ as defined by CPA encryption, and runs A on a circuits $(C_{c_1}, \ldots, C_{c_t})$ constructed using the ciphertexts $c_1, \ldots, c_t$. Then, for any distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ where X_n is distributed over $\{0,1\}^n$ with $H_\infty(X_n) \geq \alpha(n)$, and t messages $m_1, \ldots, m_t \in \{0,1\}^{\ell(n)}$, we have

$$\left| \Pr_{k\leftarrow X_n}\left[A(\mathcal{O}(I_{k,m_1}), \ldots, \mathcal{O}(I_{k,m_t})) = 1\right] - \Pr_{k\leftarrow X_n}\left[S^{I_{k,m_1},\ldots,I_{k,m_t}}(1^n, 1^\ell) = 1\right]\right|$$
$$\leq \left| \Pr_{k\leftarrow X_n, \{c_i\leftarrow \mathrm{Enc}_k(m_i)\}_{i=1}^t}\left[A(\{C_{c_i}\}_{i=1}^t) = 1\right] - \Pr_{\{c_i\leftarrow D(n,\ell)\}_{i=1}^t}\left[A(\{C_{c_i}\}_{i=1}^t) = 1\right]\right|$$

which is negligible by CPA security. □

4.4 Auxiliary input

In this section we define semantic and CPA secure encryption with *auxiliary input* that comes from a family of functions $\mathcal{Z}$. The adversary chooses a function $z \in \mathcal{Z}$ of her choice and learns $z(k)$.[11] Similarly, we define (self-composable) obfuscation with independent messages and auxiliary input family $\mathcal{Z}$, where the adversary and simulator both get $z(k)$ for some $z \in \mathcal{Z}$. Then, we show that all of the results of Section 4.3 extend naturally to the auxiliary input setting.

Definition 4.24. An encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ has CPA security for $\alpha(n)$-weak keys and *auxiliary inputs in* $\mathcal{Z}$ if there exists an efficient algorithm $D(n, \ell)$ running in time $\mathrm{poly}(n, \ell)$, such that, for all PPT adversaries A and all distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$,

$$\left|\Pr[\mathrm{CPA}_0^{X,D}(A, n) = 1] - \Pr[\mathrm{CPA}_1^{X,D}(A, n) = 1]\right| \leq \mathrm{negl}(n),$$

where the games $\mathrm{CPA}_b^{X,D}(A, n)$ for $b = 0, 1$ are defined via the following experiment:

1: $k \leftarrow X_n$
2: A submits a function $z \in \mathcal{Z}$ and receives $z(k)$
3: **repeat**
4: $\quad A$ submits a query m
5: $\quad$ set $c_0 \leftarrow \mathrm{Enc}_k(m)$, $c_1 \leftarrow D(n, |m|)$ and give c_b to A
6: **until** A halts
7: **output** the result of A

We define semantic security with $\alpha(n)$-weak keys and auxiliary inputs in $\mathcal{Z}$ via the games $\mathrm{SEM}_b^{X,D}$ that are identical to the CPA games except that the repeat loop is performed only once.

Definition 4.25. A self-composable obfuscator $\mathcal{O}$ for $\mathcal{I}$ with $\alpha(n)$-entropic security allows *auxiliary inputs from* $\mathcal{Z}$ if for any PPT adversary A with binary output and polynomial t, there exists a PPT simulator S such that for all polynomials ℓ, auxiliary input $z \in \mathcal{Z}$ and all jointly-distributed $\{(X_n, Y_n)\}_{n\in\mathbb{N}}$ where X_n takes values in $\{0,1\}^n$, $Y_n = Y_n^{(1)}, \ldots, Y_n^{(t)}$ with each $Y_n^{(i)}$ taking values

[11] Note that this is only interesting for families $\mathcal{Z}$ where each $z \in \mathcal{Z}$ is hard to invert, as otherwise $z(k)$ completely reveals k and no security is possible. Often, it makes sense to restrict $\mathcal{Z}$ much further, such as requiring that $z(k)$ is exponentially hard to invert.

in $\{0,1\}^{\ell(n)}$, and $H_\infty(X_n) \geq \alpha(n)$, we have:

$$\left|\Pr[A(z(k), \mathcal{O}(I_{(k,m_1)}), \ldots, \mathcal{O}(I_{(k,m_t)})) = 1] - \Pr[S^{I_{(k,m_1)},\ldots,I_{(k,m_t)}}(z(k), 1^n, 1^\ell) = 1]\right| \leq \text{negl}(n),$$

where the probabilities are over $(k_1, \ldots, k_t, m_1, \ldots, m_t) \leftarrow (X_n, Y_n)$ and the randomness of A, S, and $\mathcal{O}$.

An ordinary, non-composable, obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security and auxiliary inputs from $\mathcal{Z}$ satisfies the above definition for $t = 1$.

The definitions extend naturally to obfuscators with fully-entropic security, where we require that the above definition holds for all $\alpha(n) \in \omega(\log(n))$. It also extends to obfuscators for independent messages, where we restrict the definition to the case where X_n and Y_n are independent (but without an independence assumption among the coordinates within X_n or Y_n).

Equivalences

The equivalences between encryption and obfuscation extend naturally to the auxiliary input setting, based on the same constructions that were used in Section 4.3.

Theorem 4.26. *Let $\alpha(n) \in \omega(\log(n))$ and let $\mathcal{Z}$ be a family of functions. There exist obfuscators for point circuits with multi-bit output that are $\alpha(n)$-entropic secure with independent messages and auxiliary inputs from $\mathcal{Z}$ if and only if there exist semantically secure encryption schemes for $\alpha(n)$-weak keys and auxiliary inputs from $\mathcal{Z}$ that have the wrong key detection property. The equivalence also holds if we replace "$\alpha(n)$" with "fully."*

Furthermore, equivalence also holds between self-composable obfuscators and CPA secure encryption schemes with the above properties.

To prove this theorem, simply give A and S auxiliary input throughout the proofs of Lemmas 4.19, 4.20, 4.22, and 4.23. We omit further details here, and refer the interested reader to [27, Lemmas 4.1 to 4.4] for a complete proof.

4.5 KDM encryption

4.5.1 Semantically secure KDM encryption

In this section, we show equivalence between encryption with key-dependent messages (KDM) and obfuscation with dependent messages. First, we define the notion of semantically-secure KDM encryption with $\alpha(n)$-weak keys.

Definition 4.27. A symmetric key encryption scheme (Enc, Dec) is semantically secure for key-dependent messages and $\alpha(n)$-weak keys if there exists a distribution $D(n, \ell)$ that is efficiently sampleable in time $\text{poly}(n, \ell)$ such that, for all (possibly randomized) functions f, all PPT adversaries A, and all distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$, we have:

$$\left|\Pr[\text{KDM}_0^{X,D}(A, n) = 1] - \Pr[\text{KDM}_1^{X,D}(A, n) = 1]\right| \leq \text{negl}(n),$$

where $\text{KDM}_b^{X,D}(A, n)$ is defined via the following experiment:

1: choose $k \leftarrow X_n$, $c_0 \leftarrow \text{Enc}_k(f(k))$, and $c_1 \leftarrow D(n, \ell)$, where ℓ is the output length of f
2: **output** $A(c_b)$

Unlike standard definitions of KDM security, we do not require that f is an efficient function. We now show that semantically secure encryption with KDM and weak key security is equivalent to the standard notion of obfuscation, which allows for dependent input.

Theorem 4.28. *Let $\alpha(n) \in \omega(\log(n))$. There exist obfuscators for $\mathcal{I}$ with $\alpha(n)$-entropic security for dependent messages if and only if there exist semantically secure KDM encryption schemes with $\alpha(n)$-weak keys and wrong-key detection. The equivalence also holds if we replace "$\alpha(n)$" with "fully."*

The proof of the above theorem is very similar to that of Theorem 4.21. We simply observe that allowing the adversary to get encryptions of values $f(k)$ corresponds to having a distribution Y_n that depends on X_n by the relationship $Y_n = f(X_n)$. Conversely, for any joint distribution $\{X_n, Y_n\}$, we can define some (probabilistic, and possibly inefficient) function f so that $Y_n = f(X_n)$. We give the details below. Again, we prove the two sides of the "if and only if" separately.

Lemma 4.29. *Let $\alpha(n) \in \omega(\log(n))$ be an arbitrary function, and $\mathcal{O}$ be an obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security. Define $\mathrm{Enc}_k(m) \triangleq \mathcal{O}(I_{(k,m)})$ and $\mathrm{Dec}_k(C) \triangleq C(k)$, where the ciphertext C is interpreted as a circuit. The resulting encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ is semantically secure for key-dependent messages with $\alpha(n)$-weak keys and wrong-key detection.*

Proof. The correctness of decryption follows from the correctness of obfuscation. For the security of the resulting KDM encryption scheme with $\alpha(n)$-weak keys, let us fix any adversary A, function f, and distribution $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$. Define $Y_n = f(X_n)$. By the entropic security of obfuscation with dependent messages, there exists a simulator S such that:

$$
\begin{aligned}
&|\Pr[\mathrm{KDM}_0(A,n)=1] - \Pr[\mathrm{KDM}_1(A,n)=1]| \\
&\quad= \left| \Pr_{(k,m)\leftarrow(X_n,Y_n)}[A(\mathcal{O}(I_{(k,m)}))=1] - \Pr_{k\leftarrow X_n}[A(\mathcal{O}(I_{(k,0^\ell)}))=1] \right| \\
&\quad\leq \left| \Pr_{(k,m)\leftarrow(X_n,Y_n)}[A(\mathcal{O}(I_{(k,m)}))=1] - \Pr_{(k,m)\leftarrow(X_n,Y_n)}[S^{I_{(k,m)}}(1^n,1^\ell)=1] \right| && (4.7) \\
&\qquad+ \left| \Pr_{(k,m)\leftarrow(X_n,Y_n)}[S^{I_{(k,m)}}(1^n,1^\ell)=1] - \Pr_{k\leftarrow X_n}[S^{I_{(k,0^\ell)}}(1^n,1^\ell)=1] \right| && (4.8) \\
&\qquad+ \left| \Pr_{k\leftarrow X_n}[S^{I_{(k,0^\ell)}}(1^{|k|},1^\ell)=1] - \Pr_{k\leftarrow X_n}[A(\mathcal{O}(I_{k,0^\ell}))=1] \right| && (4.9)
\end{aligned}
$$

by the triangle inequality. Finally, (4.7) and (4.9) are negligible by the definition of entropic security of obfuscation with dependent messages, and (4.8) is negligible because the only way that a PPT simulator can get anything from its oracle is by querying it on the input k, which happens with negligible probability when k comes from a source of super-logarithmic entropy $\alpha(n)$. □

Lemma 4.30. *Let $(\mathrm{Enc}, \mathrm{Dec})$ be an encryption scheme with semantic KDM security for $\alpha(n)$-weak keys with wrong-key detection. We define an obfuscator $\mathcal{O}$ for $\mathcal{I}$ which, on input $I_{(k,m)}$, computes a ciphertext $c = \mathrm{Enc}_k(m)$ and outputs the circuit C_c that has c hard-coded and is defined by $C_c(x) \triangleq \mathrm{Dec}_x(c)$. Then, $\mathcal{O}$ has $\alpha(n)$-entropic security for dependent messages.*

Proof. The correctness and polynomial slowdown properties follow the argument in Lemma 4.20. For entropic security, let $D(n,\ell)$ be the distribution defined by the semantic security of the encryption scheme. For any PPT adversary A, we define the simulator S which chooses a random ciphertext c from the distribution $D(n,\ell(n))$ and runs A on a circuit C_c constructed using the ciphertext c.

Then, for any distribution ensemble $\{X_n, Y_n\}_{n\in\mathbb{N}}$ where X_n is distributed over $\{0,1\}^n$ with $H_\infty(X_n) \geq \alpha(n)$, define the function $f(k)$ which, on input k, outputs a random sample from the distribution of Y_n conditioned on $X_n = k$. Note that this function may not be efficient, but our definition of KDM encryption allows this. Then,

$$\left|\Pr_{(k,m)\leftarrow(X_n,Y_n)}\left[A(\mathcal{O}(I_{(k,m)}))=1\right] - \Pr_{(k,m)\leftarrow(X_n,Y_n)}\left[S^{I_{(k,m)}}(1^n,1^\ell)=1\right]\right| = \left|\Pr_{k\leftarrow X_n, c\leftarrow \mathrm{Enc}_k(f(k))}\left[A(C_c)=1\right] - \Pr_{c\leftarrow D(n,\ell)}\left[A(C_c)=1\right]\right|,$$

which is negligible by semantic security. □

4.5.2 Multi-KDM encryption and self-composable obfuscation

In this section, we explore a notion of CPA security with KDM and weak keys. We essentially show results analogous to those in Section 4.3.2 connecting CPA encryption (without KDM) to obfuscation with independent messages, but only if we restrict ourselves to a non-adaptive attacker who chooses the function f of the secret key prior to seeing any ciphertexts.

Definition 4.31 (Multi-KDM encryption)**.** A symmetric encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ is *multi-KDM secure* for $\alpha(n)$-weak keys if there exists a distribution $D(n,\ell)$ such that for any $t = \mathrm{poly}(n)$, any functions $f_1, \ldots, f_t$, any PPT adversary A, and any distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$, we have

$$\Pr\left[A(\mathrm{Enc}_k(f_1(k)), \ldots, \mathrm{Enc}_k(f_t(k)) = 1\right] - \Pr\left[c_i \leftarrow D(n,\ell_i) : A(c_0,\ldots,c_t)=1\right] \leq \mathrm{negl}(n),$$

where ℓ_i is the output size of f_i.

Theorem 4.32. *Let $\alpha(n) \in \omega(\log(n))$ be an arbitrary function. Let $\mathcal{O}$ be a self-composable obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security (for dependent messages). We define an encryption scheme by*

$$\mathrm{Enc}_k(m) \triangleq \mathcal{O}(I_{k,m}), \qquad \mathrm{Dec}_k(C) \triangleq C(k),$$

where C is interpreted as a circuit. This scheme is multi-KDM secure with $\alpha(n)$-weak keys.

Proof. The correctness of decryption follows from the correctness of obfuscation. For the multi-KDM security of the encryption scheme, fix any PPT adversary A, polynomial t, functions $f_1, \ldots, f_t$, and distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ with $H_\infty(X_n) \geq \alpha(n)$. Then,

$$\left|\Pr_{k\leftarrow X_n}\left[A(\mathrm{Enc}_k(f_1(k)),\ldots,\mathrm{Enc}_k(f_t(k)))=1\right] - \Pr_{k\leftarrow X_n}\left[A(\mathrm{Enc}_k(0^{\ell_1}),\ldots,\mathrm{Enc}_k(0^{\ell_t}))=1\right]\right| = \left|\Pr_{k\leftarrow X_n}\left[A(\mathcal{O}(I_{(k,f_1(k))}),\ldots,\mathcal{O}(I_{(k,f_t(k))}))=1\right] - \Pr_{k\leftarrow X_n}\left[A(\mathcal{O}(I_{(k,0^{\ell_1})}),\ldots,\mathcal{O}(I_{(k,0^{\ell_t})}))=1\right]\right|.$$

We claim that the above quantity is negligible. Because $\mathcal{O}$ is a self-composable obfuscator, there exists a simulator S that can distinguish $I_{(k,f_1(k))},\ldots,I_{(k,f_t(k))}$ from $I_{(k,0^{\ell_1})},\ldots,I_{(k,0^{\ell_t})}$ as well as A can. However, S can only distinguish the functionalities by querying the correct key k, which happens with negligible probability since k is chosen from a distribution of super-logarithmic entropy. Therefore, the above quantity is negligible as desired. □

Theorem 4.33. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be an encryption scheme with multi-KDM security for* $\alpha(n)$*-weak keys with wrong-key detection. We define an obfuscator* $\mathcal{O}$ *for* $\mathcal{I}$ *which, on input* $I_{(k,m)}$*, computes a ciphertext* $c = \mathrm{Enc}_k(m)$ *and outputs the circuit* C_c *that has* c *hard-coded and is defined by* $C_c(x) \triangleq \mathrm{Dec}_x(c)$*. Then,* $\mathcal{O}$ *is a self-composable obfuscator with* $\alpha(n)$*-entropic security for dependent messages.*

Proof. The correctness and polynomial slowdown properties follow the argument in Lemma 4.20, so it remains to show that $\mathcal{O}$ is self-composable with $\alpha(n)$-entropic security for dependent messages.

Let $D(n, \ell)$ be the distribution defined by the semantic security of the encryption scheme. For any PPT adversary A and polynomial t, we define the simulator S which chooses t random ciphertexts $c_1, \ldots, c_t$ from the distribution $D(n, \ell(n))$ in the multi-KDM security definition, and runs A on the circuits $(C_{c_1}, \ldots, C_{c_t})$ constructed using the ciphertexts $c_1, \ldots, c_t$. Then, for any distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ where X_n is distributed over $\{0,1\}^n$ with $H_\infty(X_n) \geq \alpha(n)$, and any t functions $f_1, \ldots, f_t$, we have

$$\begin{aligned}
&\left|\Pr[k \leftarrow X_n : A(\mathcal{O}(I_{k,f_1(k)}), \ldots, \mathcal{O}(I_{k,f_t(k)})) = 1] - \Pr[S^{I_{k,f_1(k)},\ldots,I_{k,f_t(k)}}(1^n, 1^\ell) = 1]\right| \\
&\leq \left|\Pr\left[k \leftarrow X_n, c_i \leftarrow \mathrm{Enc}_k(f_i(k)) : A(C_{c_1}, \ldots, C_{c_t}) = 1\right] - \Pr\left[c_i \leftarrow D(n,\ell) : A(C_{c_1}, \ldots, C_{c_t}) = 1\right]\right|,
\end{aligned}$$

which is negligible because the encryption scheme is multi-KDM secure. □

4.6 Implications

We now show how to use the above equivalence results between encryption with weak keys and obfuscation of point circuits with multi-bit output to derive new results in both directions.

4.6.1 Encryption with fully weak keys

Prior work on leakage-resilient encryption and encryption with weak-keys has given results of the following form:

1. Fix any constant $\varepsilon > 0$ and let $\alpha(n) = n^\varepsilon$.
2. Based on ε, construct an encryption scheme that achieves security for $\alpha(n)$-weak keys.

There are several concerns with the above two-step approach. Firstly, we may not have a lower bound on the key entropy at design time. Thus, in practice it may be difficult to decide which ε to use when constructing the encryption scheme. A scheme that is designed for some specific ε may not provide *any* security guarantees for key distributions whose entropy is less than n^ε. As a result, we may be tempted to be conservative with the choice of ε at design time. On the other hand, when taking an excessively small value of ε in the above constructions, we are forced to reduce the exact-security of the system (say, by working in a group of description length n^ε) or reduce the efficiency of the system proportionally with $n^{1/\varepsilon}$, leading to poorer security or performance even if the system is later only used with uniformly random keys! Secondly, none of the prior results allow for slightly super-logarithmic entropy thresholds such as $\alpha(n) = \log^{1+\varepsilon}(n)$, even if ε is specified *a priori*.

In contrast, an encryption scheme with security for fully weak keys provides the corresponding advantages. More specifically, the order of quantifiers now requires that there is a single encryption scheme, parametrized only by the security parameter n (but *not* by ε), which simultaneously achieves security for all $\alpha(n) \in \omega(\log(n))$. The exact-security of the scheme may depend on $\alpha(n)$

(since there is always a way to break the scheme in $2^{\alpha(n)}$ time), but this relationship is now more fluid, with the exact-security gracefully degrading for smaller $\alpha(n)$. In particular, the security guarantees are meaningful even for $\alpha(n) = \log^{1+\varepsilon}(n)$, and there is no single threshold above which the scheme is secure and below which it is insecure. This is a significant advantage, as it does not require one to decide at design time on the tradeoff between allowed entropy levels and achieved security/efficiency. However, there are no encryption schemes in the literature that can withstand fully weak keys.

Fortunately, we do have an obfuscator for point circuits with multi-bit output [25] that is easily seen to be self-composable. (See Section 3.2.1 for details.) Using our connection between obfuscation and encryption (Lemma 4.22), we get the first symmetric-key encryption scheme with CPA security for fully weak keys (albeit under a strong assumption).

Theorem 4.34. *Under the strong DDH assumption (Assumption 3.20), there exists a CPA-secure symmetric encryption scheme with security against fully weak keys. In particular, this means that there is a* single *scheme, parametrized only by the security parameter n, such that security of the scheme is maintained when the key is chosen from* any *distribution of entropy* $\alpha(n) \in \omega(\log(n))$.

Such a construction may be important in practice if we want to use secrets such as passwords or biometrics for encryption. In this case, it is impractical to design a scheme targeted at the minimal amount of entropy that one wishes to support, secure if the actual entropy of the secret used exceeds this threshold. Instead, by using a scheme with CPA security for fully weak keys, we can employ a single encryption scheme whose security varies depending on the entropy of the secret key.

The only downside is that Assumption 3.20 is very strong. A potentially weaker formulation would be to limit the min-entropy of X_n to be at least some specific super-logarithmic function $\alpha(n)$. This way, we would obtain a parametrized version of Theorem 4.34 that relates the strength of the security guarantee to the strength of the assumption. It is important to note that the construction itself is independent of the parameter α. Thus, we obtain a single encryption scheme that provides a range of security guarantees, depending on the strength of the assumption.

4.6.2 Achieving obfuscation with independent messages

It is fairly simple to construct $\alpha(n)$-entropically secure obfuscation for independent messages, when $\alpha(n) = n^{\varepsilon}$ for some constant $\varepsilon \geq 0$. First we construct a semantically secure encryption scheme with $\alpha(n)$-weak keys. This can be done by simply extracting a sufficient amount of uniform randomness from the key k, using a strong randomness extractor Ext, and then using the result as a one time pad to encrypt the message. For variable-length messages, we also need to expand the extracted randomness to an appropriate size using a pseudo-random generator prg. In particular, we define

$$\mathrm{Enc}_k(m) = \langle r, \mathrm{prg}(\mathrm{Ext}(k; r)) \oplus m \rangle,$$

where r is a uniformly random seed for the extractor. The output length of Ext and the input length of prg are set to some value v which is sufficiently small that the output of the extractor is (statistically) close to uniform, and sufficiently large that the output of the prg is pseudo-random.[12]

One can use this encryption scheme to construct one which also has the wrong-key detection property using Lemma 4.7. Such a scheme yields an obfuscator for $\mathcal{I}$ with $\alpha(n)$-entropic security for independent messages by Lemma 4.20.

[12] For example, if we choose $v = n^{\varepsilon}/2$, then an extractor based on universal-hash functions will produce an output which is $2^{-v/2} = \mathrm{negl}(n)$-close to uniform, and the output of the pseudorandom generator is $\mathrm{negl}(n^{\varepsilon}/2) = \mathrm{negl}(n)$-pseudorandom. However, this does not generalize to smaller values of α, such as $\alpha(n) = \log^2(n)$.

Self-composition

The above construction of semantically-secure encryption using extractors does not generalize to CPA security. In fact, achieving CPA secure encryption with weak keys seems to be a much harder problem, which has received much attention in recent works [3, 38, 71]. We now show how to use these results to achieve self-composable entropically secure obfuscation for independent messages. On a high level, we would simply like to just apply our result connecting such encryption and obfuscation (Lemma 4.23) "out of the box." However, there are several issues that we must deal with first.

Efficiently-sampleable distributions. The works of [3, 38, 71] are concerned with "key leakage," where the adversary gets to learn some (short) function of the secret key whose output length is at most λ bits. Conditioned on such leakage, the key can be thought of as being derived from a special type of weak source with entropy $\alpha(n) \approx n - \lambda$. It turns out that the constructions are also secure when the key is chosen from an arbitrary, but *efficiently-sampleable* weak source of entropy $\alpha(n)$ [71]. Therefore, our results for obfuscation will only translate to the case where the distribution obfuscated program is efficiently sampleable.

Public keys and parameters. Only the scheme of [38] is explicitly designed for the symmetric key setting. The schemes of [3, 71] are public key encryption schemes. As noted, such schemes are secure when the key generation procedure uses randomness that comes from a weak source. Therefore such schemes naturally translate to the symmetric key setting, where the randomness of the key generation algorithm is the shared secret key. Unfortunately, these schemes also rely on *public parameters* which are chosen uniformly at random and are available to the key generation algorithm. Therefore, we will only get an obfuscator in the presence of public parameters. In general, having public knowledge is undesirable (indeed, it violates one of our design principles from Section 1.2.1), but in the specific case of digital lockers it may be reasonable to allow.

Note that in the context of standard obfuscation, public parameters are never needed since the obfuscator $\mathcal{O}$ could always sample fresh parameters each time it runs. However, when considering composable obfuscation, this equivalence does not hold since future uses of the obfuscator might compromise the security of prior uses.

Uniform ciphertexts. Recall that our definition of CPA security is slightly different than the standard one (we require that the ciphertexts of any message are indistinguishable from some universally specified distribution) and has not been explicitly analyzed by the encryption schemes in [3, 38, 71]. However, all of the schemes have the property that ciphertexts are indistinguishable from uniform, which is enough to satisfy our definition.

Wrong-key detection. The wrong-key detection property is explicitly analyzed in [38] but not in [3, 71]. The latter two schemes have the property that, given the public parameters, it is computationally difficult to find k, k' such that $\mathrm{Dec}_{k'}(\mathrm{Enc}_k(m)) \neq \perp$. This translates to a *computational correctness* property for the obfuscator where, given the public parameters, it is computationally difficult to find k, m, and x such that $\mathcal{O}(I_{(k,m)})(x) \neq I_{(k,m)}(x)$.

The following theorem summarizes the connections.

Theorem 4.35. *For any constant $\varepsilon > 0$, there exists a self-composable obfuscator for $\mathcal{I}$ with independent messages under any of the following assumptions:*

1. *Decisional Diffie-Hellman (DDH) with n^{ε}-entropic security, based on [71].*

2. *Learning With Errors (LWE) with n^{ε}-entropic security, based on [3].*

3. *Learning Subspaces with Noise with εn-entropic security, based on [38].*

The constructions only work for efficiently-sampleable key distributions. Furthermore, the first two constructions require public parameters and only achieve computational correctness.

4.6.3 Difficulty of obfuscation with dependent messages

The connection between encryption and obfuscation also yields new negative results for the more standard notion of obfuscation that allows for dependent messages, and in particular for the standard virtual black-box property. We rely on a recent result of Haitner and Holenstein [50], which shows that there can be *no* black-box reduction from a semantically secure encryption scheme with security against key-dependent messages to any "standard" cryptographic assumption, such as the existence of trapdoor one-way or claw-free permutations, or specific algebraic assumptions like the hardness of factoring, DDH, or Learning with Errors.[13] By Theorem 4.28, we have a reduction from semantically secure encryption schemes with security against key-dependent messages to digital locker obfuscation with n-entropic security (i.e., uniformly random keys). Therefore, the Haitner and Holenstein's black-box separation [50] applies to obfuscation as well.

Theorem 4.36. *No construction of an obfuscator for point circuits with multi-bit output with $\alpha(n)$-entropic security for dependent messages can be proven secure via a black-box reduction to any "standard" cryptographic assumption. This holds for any α, even $\alpha(n) = n$ (i.e., uniformly random keys).*

Canetti and Dakdouk [25] showed that composable obfuscation of point circuits can be used to construct digital lockers. Thus we get the following as a corollary.

Corollary 4.37. *No construction of a composable obfuscator for point circuits with $\alpha(n)$-entropic security can be proven secure via a black-box reduction to any "standard" cryptographic assumption. This holds for any α, even $\alpha(n) = n$ (i.e., uniformly random keys).*

We note that the impossibility result of [50] only considers semantically secure encryption with variable-length messages and does not rule out KDM security when the message size is shorter than the key. In fact, [25] constructs digital lockers[14] with $\alpha(n)$-entropic security for dependent messages, with the constraint that the message size is (significantly) smaller than the key size (i.e., circuits $I_{(k,m)}$ where $|m| < |k|$). These constructions only relied on standard cryptographic assumptions such as collision-resistant hash functions. The above theorem shows that such constructions do not generalize to variable-length messages, where the message size can exceed the key size.

In contrast, we showed how to construct self-composable digital locker obfuscators with $\alpha(n)$-entropic security under standard assumptions, in the case of variable-length independent messages. It seems that there is little hope in generalizing this approach to the standard notion of obfuscation, which allows key-dependent messages.

[13] More precisely, the notion of "cryptographic assumption" is formalized in [50] as any game between an attacker and a challenger in which we assume that all PPT attackers have a negligible success probability. Hence, the impossibility result does not exclude proofs of security in the random oracle model, reductions to non-standard assumptions that cannot be formulated as a game between an adversary and a challenger (such as the knowledge of exponent assumption [11, 49]), or non-black-box reductions.

[14] See Section 3.2.1 for an overview of the construction.

Chapter 5

Non-malleable Obfuscation

This chapter is based on joint work with Ran Canetti [31, 32].

5.1 Introduction

We motivated program obfuscation in Section 1.1 as the software analog to tamper-proof hardware. However, the virtual black-box property (Definition 2.4) and its extensions[1] do not quite satisfy this analogy.

The problem is that tamper-proofing a hardware chip provides two distinct security guarantees: first, that an adversary does not learn what the chip is doing "under the hood," and second, that the adversary cannot tinker with the chip to alter its behavior (say, by inserting or cutting wires). By contrast, the virtual black-box property only gives an unlearnability guarantee. It does not explicitly rule out malleability attacks, where an adversary that sees an obfuscated program is able to generate another (potentially obfuscated) program that is related to the original one.

This is not simply a theoretical concern, but can be desirable in practice. We motivate the problem with an example: suppose that Alice, Bob, and Charles are three graduate students that share a computer. Obfuscating the computer's login program protects the students' passwords from outsiders, but we may also want to prevent tampering of the login program by the students themselves. For instance, a malicious Alice might attempt to remove Bob's access to the computer. We could thwart this attack by protecting the login program in tamper-proof hardware, but we want an anti-tamper guarantee in software.[2]

The virtual black-box property does not explicitly forbid such attacks, but it does intuitively appear to prevent tampering.

Naïve claim. *A virtual black-box obfuscator $\mathcal{O}$ (as in Definition 2.4) is non-malleable in the sense that an adversary with an obfuscated circuit $\mathcal{O}(C)$ cannot form a new circuit that is related to $\mathcal{O}(C)$ except in trivial ways such as querying $\mathcal{O}(C)$ at a polynomial number of locations and forming a new circuit that depends on its responses.*

Proof sketch. By the virtual black-box property, there exists a simulator that is associated with this adversary. The simulator only has oracle access to C, so any action that it undertakes can only depend on querying C at a polynomial number of locations (perhaps adaptively). Therefore, the same should hold for the adversary. □

[1] In this chapter, we use the Goldwasser and Kalai definition of obfuscation with dependent auxiliary input [43].

[2] We discuss this example in far more detail in Section 1.3.2, and strongly encourage the reader to review it before proceeding.

However, this "proof" is incorrect because the virtual black-box property only considers adversaries and simulators that output one bit, not entire circuits. Furthermore, the problem here is not just a faulty proof, but rather that the underlying claim is incorrect.

The virtual black-box property does not contain an anti-tampering guarantee, and in fact the known obfuscators for the family of multi-point circuits are malleable [12, 23, 28] in the sense that an adversary can modify an obfuscated circuit $\mathcal{O}(I_w)$ into a new one $\mathcal{O}(I_{w'})$ such that she understands the relationship between w and w'. For instance, in the above example, Alice can remove Bob's access to the computer. The known obfuscators for the family of point circuits with multi-bit output in the standard model are even more vulnerable, as the adversary can easily modify the hidden message in a meaningful way [12, 25, 93].[3]

The goal of this chapter is to incorporate a non-malleability guarantee into the virtual black-box property, thus completing the analogy between obfuscation and tamper-proof hardware. We do so by strengthening the virtual black-box definition so it applies to adversaries and simulators that can output more than one bit. By doing so, the fake "proof" from above will hold, so the definition will prevent tampering as desired. On the other hand, we do not want to strengthen the definition so much that it becomes impossible to achieve. Striking the right balance here proves to be rather delicate.

We also give a definition that provides a tamper-evident guarantee, detecting attacks rather than preventing them. This model requires that somebody is capable of performing the tamper-evidence check, which may be a worthwhile tradeoff in certain situations to achieve extra security. For instance, in our example above, suppose that Alice no longer wants to remove Bob's access to the computer, but instead wishes to play a prank on Bob and Charles by modifying the login program so it accepts their passwords concatenated to the string "Alice is great." Tamper-proofing cannot prevent this attack,[4] but tamper-evidence can detect it.

Organization

In Section 5.2, we provide rigorous definitions for the two notions of non-malleability and explain the subtleties involved in their creation. In Sections 5.3 and 5.4, we construct non-malleable obfuscators of both flavors for the family of multi-point circuits.

5.2 Defining non-malleable obfuscation

In this section, we rigorously define the two variants of non-malleable obfuscation, which we call tamper-proof and tamper-evident obfuscation.[5] They are roughly equivalent to the guarantees provided by tamper-proof and tamper-evident hardware: the goal of tamper-proofing is to *prevent* attacks, whereas the goal of tamper-evidence is to *detect* them.

5.2.1 Tamper-proof obfuscation

We obtain tamper-proof obfuscation by generalizing the virtual black-box definition to allow the adversary and simulator to output programs instead of bits. Intuitively, a tamper-proof obfuscation has the property that an adversary, given the obfuscated code to a program, can only make a related program if it could have already done so given only black-box access to the program.

[3] See Section 3.2.1 for details on these constructions.

[4] We describe why this is so in Section 1.3.2.

[5] In the original work upon which this chapter is based [31, 32], the two variants were called "functional" and "verifiable" non-malleable obfuscation, but we change to a more intuitive terminology in this thesis.

This is problematic in general, because the simulator cannot emulate all programs that the adversary can produce [10, 93]. For example, consider the adversary that outputs its input. Then, the simulator has oracle access to a circuit and has to produce a program that is functionally equivalent to its oracle. This is impossible unless the circuit is learnable with oracle queries, in which case the entire concept of obfuscation is uninteresting (as described in Section 1.2.4).

However, it is unfair to demand that the simulator do this much work. After all, the adversary's input is a program but the simulator's input is just an oracle. At the very least, the adversary can output a program that uses its input program in a black-box manner, and the simulator should have the same ability. Therefore, we allow the program that the simulator produces to make oracle queries to the original circuit as well.

To capture the effectiveness of an adversary's modification, we introduce a polynomial-time computable relation E that receives the adversary's input program and output program. The adversary succeeds in the modification if E accepts it. The definition of non-malleability ensures that for every relation E, the simulator can perform a successful modification with the approximately the same probability as the adversary.

One technical concern about the relation E is the manner in which it receives the adversary's input and output programs. The goal of tamper-proofing is to compare the functionality of these programs, and not their underlying code, so E should operate in the same manner when given functionally equivalent inputs. Our definition resolves this issue by giving the relation a "canonical" member of the family that is equivalent to the adversary's output program. (See the Discussion section below for more detail on this issue.)

Additionally, in many situations, the adversary knows some a-priori useful information on the obfuscated program, so we allow dependent auxiliary information in the definition of non-malleability. For instance, in the motivating example from the Introduction in which Alice wishes to modify a login program, she possesses the knowledge of her own password.

Definition 5.1. Let $\mathcal{C}$ and $\mathcal{D}$ be families of polynomial-size circuits, and let $\mathcal{O}$ be a PPT algorithm. We say that $\mathcal{O}$ is an *obfuscator for $\mathcal{C}$ that is tamper-proof over $\mathcal{D}$* if the following three conditions hold:

1. *Almost exact functionality*: There exists a negligible function ε such that for every n and every circuit $C \in \mathcal{C}_n$,
$$\Pr\left[\mathcal{O}(C;r) \equiv C\right] > 1 - \varepsilon(n),$$
where the probability is taken over the randomness r.

2. *Polynomial slowdown*: There exists a polynomial ξ such that for every n, every circuit $C \in \mathcal{C}_n$, and every possible sequence of coin tosses r, the description length $|\mathcal{O}(C;r)| \leq \xi(n)$.

3. *Tamper-proofing*: for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$ (that may depend on the circuit C),
$$\Big|\Pr\left[P \leftarrow A(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C,D) = 1\right] - \Pr\left[Q \leftarrow S^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } E(C,D) = 1\right]\Big| < \frac{1}{\rho(n)},$$
where the probabilities are over the coin tosses of A, $\mathcal{O}$, and S. We require that the runtime

of A and S is polynomial in the length of their first input.[6]

If $\mathcal{D} = \mathcal{C}$, we simply say that $\mathcal{O}$ is a *tamper-proof obfuscator for* $\mathcal{C}$.

Note that we use the Goldwasser and Kalai method of incorporating dependent auxiliary input into the definition [43]. Also, recall that adversaries and simulators are allowed to be non-uniform throughout the thesis.

5.2.2 Discussion

We make several remarks about this definition.

Almost exact functionality

The minimal functionality requirement here is stronger than the one in Definition 2.4. Approximate functionality only guarantees that an obfuscated program $\mathcal{O}(C; r)$ is "close" in functionality to C. However, $\mathcal{O}(C; r)$ might never have the same functionality as C does, for any choice of r. By contrast, almost exact functionality requires that the two circuits have identical functionality for most choices of r.

Our definition checks the functionality of $\mathcal{O}(C; r)$ on all inputs, so the stronger notion is important here. With approximate functionality alone, it may be difficult to find a circuit $D \in \mathcal{D}_n$ that is equivalent in functionality to the program that the adversary constructs unless one uses a very large family $\mathcal{D}_n$ (which is a bad idea for a different reason that is described below).

We note that most of the constructions in this work satisfy exact functionality.

Determinism of $\mathcal{C}$ and $\mathcal{D}$

In our definition, the adversary and simulator are supposed to output a circuit that is equivalent in functionality to a circuit in $\mathcal{D}$. Our concept of functional equivalence (from Definition 2.3) only makes sense for deterministic circuits. Therefore, throughout this work we assume that circuits in $\mathcal{C}$ and $\mathcal{D}$ are deterministic. In particular, the circuits that the simulator outputs in our proofs must also be deterministic.

Bivariate relation

In this definition, the bivariate relation E is allowed to depend on the choice of circuit $C \in \mathcal{C}_n$. Thus, restricting attention to univariate relations $E(D)$ results in an equivalent definition. We use a bivariate relation only to emphasize the fact that E depends on both C and D.

Possible definitions for E

As mentioned above, an important feature of the definition is that E only depends on the functionality of the adversary's output P and simulator's output Q^C, and not on the code of these circuits. We found three possible ways to enforce this condition on E.

First, we can constrain E to receive only oracle access to the program P or Q^C. As a result, it follows immediately that E only depends on the functionality of these programs, and not on their underlying code. Unfortunately, this definition is too weak, because there are many natural predicates that cannot be tested by relations of this type.

[6] Recall that the notation $\equiv$ denotes functionally equivalent circuits, as described in Definition 2.3.

For instance, suppose the adversary is given an obfuscation of the point circuit I_w and wishes to create a new point circuit $I_{w'}$ such that the first bit of w and w' are equal. No polynomial-time relation E (even ones that know w, since E can depend on w) can test the adversary's probability of success given only oracle access to $I_{w'}$. We believe that relations of this type are meaningful, and therefore we want a definition that can test for them.

Second, we can give the relation E full access to the code of P or Q^C, but restrict our attention to relations that have identical output when given two functionally equivalent programs. Specifically, we only consider relations E such that given any programs $C \in \mathcal{C}_n$, $D \in \mathcal{D}_n$, and P, P' such that $D \equiv P \equiv P'$, it follows that $E(C, P) = E(C, P')$. Unfortunately, this restriction is so strong that E cannot take advantage of having access to the code of its input programs. It is still impossible to compute many relations, such as the one described in the previous paragraph.

Specifically, the virtual black-box definition guarantees that any relation $E(I_w, \mathcal{O}(I_{w'}))$ cannot compute whether w and w' have the same first bit with probability greater than $\frac{1}{2}$. Thus, the condition that we impose on E is that it has the same probability of success even when it is given I_w and $I_{w'}$ as inputs, in which case E has enough information to perform the computation with probability 1 but cannot take advantage of this information because it must still fail half of the time anyway in order to be consistent with the condition.

Third, we can allow all polynomial-time relations E, but instead of providing the code of P or Q^C as input to E, we provide the code of a functionally equivalent member in $\mathcal{D}_n$. This is the option we use in Definition 5.1 above, because it clearly satisfies the requirement that E only depend on the functionality of the adversary and simulator's output, and it is a stronger definition that can test for many relations that the previous two definitions cannot. For these reasons, we choose to use relations of this type in the definition of tamper-proof obfuscation.

One technical point to keep in mind is that the relation E takes the description of circuits in $\mathcal{C}_n$ and $\mathcal{D}_n$ as input. As a result, this definition is dependent upon the representation of the circuits in these families, and not just the functionality of these circuits. Therefore, we should choose a representation of the circuit families that enables relations to extract important information easily from the description of a circuit, as we have done with the families $\mathcal{P}^m$ and $\mathcal{I}$ in Section 2.1. With an appropriate representation, our definition allows the relation E to "peel off" any obfuscation from the circuits that it receives.

Output family $\mathcal{D}$

According to our definition, an adversary succeeds only if it outputs a circuit that is equivalent to a circuit in the family $\mathcal{D}$. The most natural family to choose is $\mathcal{D} = \mathcal{C}$, but we allow $\mathcal{D}$ to be different from $\mathcal{C}$ in order to consider a wider range of adversaries. For instance, perhaps $\mathcal{C}$ is the family of point circuits, but we are concerned with adversaries that produce two-point circuits. The f definition provides the flexibility to accommodate this.

Of course, there is no reason to stop there: we may also be concerned with an adversary that produces a three-point circuit, or a four-point circuit, or any circuit for that matter. It would be nice if our definition simultaneously covered all possible outputs of the adversary, and not just those in a specific family. In other words, we would like a tamper-proof obfuscator when $\mathcal{D}$ is the family of all circuits.

Unfortunately, this is not possible for many circuit families of interest such as multi-point circuits $\mathcal{P}^{m+}$ or point circuits with multi-bit output $\mathcal{I}$. Intuitively, the family of all circuits is so big that it allows A to output the obfuscated code that it receives as input, which the simulator cannot do.

Theorem 5.2. *Let $\mathcal{D}_n$ be the set of all circuits with input length n and binary output, and let $\mathcal{D} = \{\mathcal{D}_n\}$. Let $\mathcal{C}$ be a circuit family with the following two properties:*

1. *$\mathcal{C}$ is not learnable, so for every simulator S, there exists a circuit $C \in \mathcal{C}_n$ such that the circuit S, with only oracle access to C, cannot output the description of a circuit that is equivalent in functionality to C except with negligible probability.*[7]

2. *Given a circuit $C \in \mathcal{C}_n$ and a circuit P that is equivalent to a member of $\mathcal{C}_n$, one can test in polynomial time whether $C \equiv P$.*

Then, there are no obfuscators $\mathcal{O}$ for $\mathcal{C}$ that are tamper-proof over $\mathcal{D}$.

Proof. Suppose that there exists an obfuscator $\mathcal{O}$ for $\mathcal{C}$ that is tamper-proof over $\mathcal{D}$. Informally, we will derive a contradiction by creating an adversary that takes its input program $\mathcal{O}(C)$, views it as a string s, and encodes the string in a circuit $D \in \mathcal{D}_n$ in a readily identifiable way. By unlearnability, the simulator cannot construct the code of any program that is functionally equivalent to C, which establishes the separation.

One simple choice is to make the circuit $D = I_s$, that is, D is the point circuit that accepts only the string s. Unfortunately, this naive idea does not work due to a technical constraint, namely that D is supposed to be a circuit in $\mathcal{D}_n$, and hence D should have n bits of input. However, the circuit I_s requires $|s|$ bits of input, which is usually greater than n. Instead, we form a circuit $D \in \mathcal{D}_n$ whose truth table is determined by s in such a way that s can be recovered by performing a small number of executions of D.

Given a circuit $C \in \mathcal{C}_n$ and the empty auxiliary input $z = \emptyset$, let A be the adversary that takes its input program $P = \mathcal{O}(C)$ and views it as a binary string s. Then, A appends a termination character $\perp$ to s, so s is now a string with a ternary alphabet. Next, A applies a canonical ternary-to-binary encoding of the string s into a new string s'. Now, A forms the circuit $D \in \mathcal{D}_n$ whose truth table equals the string s' concatenated with the all 0s string. Note that the truth table for D is 2^n bits long, whereas the length of the string s' is only polynomially large in n, so for sufficiently large n the entire string s' can be encoded into the truth table for D. Finally, A outputs D.

Let $E(C, D)$ be the relation that does the following:

1. It queries D on enough input values to recover the string s', stopping once it views the encoded version of the $\perp$ symbol.

2. It decodes s' into the string s and views it as the binary representation of a program. In this way, E has recovered the program P.

3. Return 1 if and only if P is equivalent to its first input C, which by assumption it can test in polynomial time.

Note that the adversary always encodes a circuit that is equivalent in functionality to C, so it passes the relation test with probability 1. On the other hand, the circuit family $\mathcal{C}$ is not learnable, so for every simulator S, there exists a circuit $C \in \mathcal{C}_n$ such that S passes the relation test with only negligible probability. Therefore, A does not have a satisfactory simulator, so the tamper-proofing property is not true. Therefore, $\mathcal{O}$ is not an obfuscator for $\mathcal{C}$ that is tamper-proof over $\mathcal{D}$, as desired. □

[7] In Section 1.2.4, we described why learnable circuit families are trivially obfuscatable but uninteresting. This statement codifies the meaning of "learnable."

The key step in this proof is showing a bijection between integers s and members of the family of all circuits. Because the integer s is upper-bounded by $2^{\xi(n)}$, where ξ is the polynomial from the polynomial slowdown property, the impossibility result extends to any family $\mathcal{D}$ such that every $\mathcal{D}_n$ has an efficient ordering of at least $2^{\xi(n)}$ circuits. These families can be much smaller than the family of all circuits.

Comparison to virtual black-box obfuscation

Now that we have introduced a new definition of obfuscation, it is natural to compare it to the old one. We show that the tamper-proofing property implies the virtual black-box property (at least for reasonable choices of the circuit family $\mathcal{D}$), which justifies our terminology of using the word "obfuscation" in Definition 5.1.

Theorem 5.3. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families, and let $\mathcal{O}$ be an obfuscator for $\mathcal{C}$ that is tamper-proof over $\mathcal{D}$ (as defined in Definition 5.1). Additionally, suppose that for sufficiently large n, at least one of the following is true:*

- *$\mathcal{D}_n$ is non-trivial, in the sense that there exist two circuits D_0, $D_1 \in \mathcal{D}_n$ such that $D_0 \not\equiv D_1$.*
- *$\mathcal{C}_n$ is trivial, in the sense that all of the circuits in $\mathcal{C}_n$ have the same functionality.*

Then, $\mathcal{O}$ satisfies the virtual black-box requirement for $\mathcal{C}$. As a result, $\mathcal{O}$ is an obfuscator for $\mathcal{C}$ with respect to dependent auxiliary information (as defined in Definition 2.4). This theorem also holds if neither the virtual black-box property nor tamper-proofing allows auxiliary information.

Intuitively, the theorem holds because an adversary that outputs programs can use this channel to transmit a single bit b by outputting the program D_b, but the theorem is not clear *a priori* since the notion of success for the adversary is more restricted in the tamper-proof obfuscation definition. One consequence of this theorem is that all impossibility results pertaining to the virtual black-box property immediately carry over to the non-malleability setting [10, 43].

The rest of this section is devoted to a proof of the theorem. While the proof is fairly technical, the idea is rather simple. Given an adversary A with binary output, we construct a new adversary A' (that is not constrained to binary output) in the following manner. Let D_0 and D_1 be two distinct circuits in $\mathcal{D}_n$, and without loss of generality suppose that there is some input value x such that $D_0(x) = 0$ and $D_1(x) = 1$. Then, A' emulates an execution of A until it returns a bit b, upon which A' outputs the circuit D_b. In essence, the adversary A' uses its ability to output programs in order to communicate a single bit of information.

By tamper-proofing, there exists a simulator S' corresponding to A'. Now we reverse the above procedure and form a simulator S with binary output as follows: S emulates an execution of S' until it outputs a circuit Q, and then S outputs the single bit $Q(x)$. We claim that S is a simulator for A that satisfies the virtual black-box requirement. This seems plausible because S extracts the single bit of information that A' and S' were trying to communicate. Indeed, if S' was an "ideal" simulator that always outputs D_0 or D_1 with the same probabilities that A' does, then it is trivially easy to show that S satisfies the virtual black-box property.

However, even though A' always outputs one of D_0 or D_1, it is not true that the same must be true of S', so the above proof technique fails. In fact, it is not immediately clear that the output of S' must be equivalent to *any* member of $\mathcal{D}_n$. Luckily, this statement turns out to be true, and it suffices to prove the theorem.

Proof. Let A be an adversary with binary output and let ρ be a polynomial. We need to construct a simulator S for A that satisfies the virtual black-box property. We prove this statement in two cases.

First, consider all n such that $\mathcal{D}_n$ is non-trivial. In this case, there exist two circuits $D_0, D_1 \in \mathcal{D}_n$ that do not have the same functionality. That is, there exists at least one input value x such that $D_0(x) \neq D_1(x)$. Because the outputs are not equal, they differ in at least one bit position i, and without loss of generality we assume that $D_0(x)$ equals 0 in this bit position whereas $(D_1(x))_i = 1$.

We use A to form an adversary A' that is not constrained to binary output in the following manner: A' stores A, D_0, and D_1 in some readily identifiable way (by non-uniformity), and on input $\mathcal{O}(C)$ and auxiliary information z, the circuit A' operates as follows.

1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit b
2: **output** the program D_b

Given an adversary A' and polynomial 2ρ, tamper-proofing guarantees the existence of a simulator S' such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all PPT relations E,

$$\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } E(C, D) = 1\right]\Big| < \frac{1}{2\rho(n)}. \quad (5.1)$$

Finally, we construct a simulator S with binary output that nonuniformly hardcodes x and i and operates as follows, when given auxiliary information z and access to an oracle C:

1: S emulates an execution of $(S')^C(1^n, z)$ until it returns a string s
2: **if** s is not the valid encoding of a circuit **then**
3: **output** 0
4: **end if**
5: let Q be the circuit that is represented by s
6: **output** the bit $(Q^C(x))_i$

We now show that this simulator S satisfies the virtual black-box property.

In order to analyze this claim, we must understand the behavior of S'. Even though A' always outputs either D_0 or D_1, the simulator S' does not have to do the same. In fact, it is not immediately clear how often S' outputs any member of $\mathcal{D}_n$. Luckily, S' almost always outputs a member of $\mathcal{D}_n$, which we prove by using the inequality (5.1) on two relations.

First, let $\tilde{E}(C, D)$ be the relation that accepts if and only if $D(x)_i = 1$. Note that $\tilde{E}$ has x and i nonuniformly hardcoded, and the output of $\tilde{E}$ is independent of the choice of C. Then, for all $n \in \mathbb{N}$, $C \in \mathcal{C}_n$, and $z \in \{0,1\}^*$,

$$\begin{aligned}\Pr\left[A(\mathcal{O}(C), z) = 1\right] &= \Pr\left[A'(\mathcal{O}(C), z) \text{ outputs } D_1\right] \\ &= \Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \tilde{E}(C, D) = 1\right].\end{aligned}$$

The first equality is an immediate consequence of the construction of A', and the second equality follows from the fact that A' always outputs either D_0 or D_1, and of the two only D_1 satisfies the relation. Using this equality, along with the precise behavior of the relation $\tilde{E}$, tamper-proofing

states that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary information z,

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right]\Big| < \frac{1}{2\rho(n)}. \quad (5.2)$$

Second, let $\hat{E}$ be the relation such that $\hat{E}(C, D) = 1$ if and only if $D(x)_i = 0$. Using the same analysis as above, it follows that

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 0\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 0\right]\Big| < \frac{1}{2\rho(n)}.$$

Because A is an adversary with binary output, $\Pr\left[A(\mathcal{O}(C), z) = 1\right] + \Pr\left[A(\mathcal{O}(C), z) = 0\right] = 1$. Applying the triangle inequality to the previous two inequalities, it follows that

$$\Big|\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right] + \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 0\right] - 1\Big| < \frac{1}{2\rho(n)}.$$

Of course, for any circuit D and input x, it is always the case that either $(D(x))_i = 1$ or $(D(x))_i = 0$. As a result, the above inequality simplifies to

$$\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C\right] > 1 - \frac{1}{2\rho(n)}.$$

Therefore, with very large probability, S' outputs a valid encoding of a circuit that is equivalent to a member of $\mathcal{D}_n$.

Finally, by the construction of S from S', it is easy to see that

$$\Pr\left[S^C(1^n, z) = 1\right] = \Pr\left[Q \leftarrow (S')^C(1^n, z) : (Q^C(x))_i = 1\right].$$

Furthermore, we showed that Q fails to be equivalent to a member of $\mathcal{D}_n$ with small probability, so

$$\Big|\Pr\left[S^C(1^n, z) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right]\Big| < \frac{1}{2\rho(n)}.$$

Applying the triangle inequality to this inequality and (5.2) yields

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\Big| < \frac{1}{\rho(n)},$$

so S satisfies the virtual black-box property as desired.

Now we prove the theorem for values of n such that $\mathcal{C}_n$ and $\mathcal{D}_n$ are both trivial. In this case, we claim that for every adversary A with binary output, for every $z \in \{0,1\}^*$, for sufficiently large n, and for every $C, C' \in \mathcal{C}_n$,

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\Big|$$

is negligible. If this is true, then it is easy to construct a simulator S for A: the simulator simply chooses a circuit $C \in \mathcal{C}_n$ arbitrarily, obfuscates it, and emulates an execution of A on the obfuscated circuit. It is straightforward to check that this simulator satisfies the virtual black-box property.

Suppose for the sake of contradiction that the claim is not true. Therefore, there exists an adversary A, a polynomial ρ, and auxiliary information $z \in \{0,1\}^*$ such that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ such that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right| > \frac{1}{\rho(n)}.$$

Now form an adversary A' (that is not constrained to binary output) as follows. Arbitrarily choose a circuit $D_1 \in \mathcal{D}_n$, and let D_0 be a circuit that is not equivalent to D_1, and hence not equivalent to any member of $\mathcal{D}_n$. The adversary A' nonuniformly hardcodes A, D_0, and D_1, and operates as follows upon receiving $\mathcal{O}(C)$ and z as input:

1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit b
2: **output** D_b

Let $\bar{E}$ be the relation that always accepts. As a result,

$$\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] = \Pr\left[A(\mathcal{O}(C), z) = 1\right]$$

because D_1 is equivalent to a member of $\mathcal{D}_n$ but D_0 is not. Hence, by our assumption above, it follows that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ and a polynomial ρ such that

$$\begin{aligned}\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] \\ - \Pr\left[P \leftarrow A'(\mathcal{O}(C'), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C', D) = 1\right]\Big| > \frac{1}{\rho(n)}.\end{aligned}$$

However, for any simulator S',

$$\begin{aligned}\Pr\left[Q \leftarrow (S')^C(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } \bar{E}(C, D) = 1\right] \\ = \Pr\left[Q \leftarrow (S')^{C'}(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^{C'} \text{ and } \bar{E}(C', D) = 1\right]\end{aligned}$$

information-theoretically because C and C' are functionally equivalent. Applying the triangle inequality to the previous two equations, it must be the case that either

$$\begin{aligned}\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] \\ - \Pr\left[Q \leftarrow (S')^C(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } \bar{E}(C, D) = 1\right]\Big| > \frac{1}{2\rho(n)},\end{aligned}$$

or the same is true of C'. As a result, S' fails to satisfy the tamper-proofing property. This is true for all possible S', so $\mathcal{O}$ is not tamper-proof over $\mathcal{D}$, which is a contradiction. □

It is simple to check that the proof goes through even if the auxiliary input z is constrained to be the empty string. As a result, tamper-proofing without auxiliary information implies obfuscation without auxiliary information. Similarly, the proof goes through if both properties are modified to require a negligible difference in the probabilities of success for the adversary and simulator.

5.2.3 Tamper-evident obfuscation

In this section, we develop the notion of verifiable obfuscation and use it to define another definition of non-malleability.

Definition 5.4. Given a pair of PPT algorithms $\mathcal{O}$ and V and a circuit family $\mathcal{C}$, we say that V is a *verifier for $\mathcal{O}$ applied to $\mathcal{C}$* if there exists a negligible function ε such that for all $n \in \mathbb{N}$ and $C \in \mathcal{C}_n$, $\Pr[V(\mathcal{O}(C)) = 1] > 1 - \varepsilon(n)$, where the probability is taken over the randomness of V, $\mathcal{O}$.

If $\mathcal{O}$ is an obfuscator for the family of circuits $\mathcal{C}$, then we say that the pair $(\mathcal{O}, V)$ constitutes a *verifiable obfuscator* for $\mathcal{C}$.

We do not place any restrictions on V when its input is not the result of the obfuscator applied to a circuit in the family. In particular, given any obfuscator $\mathcal{O}$, the pair $(\mathcal{O}, V_{\text{all}})$ is a verifiable obfuscator, where V_{all} is the algorithm that accepts all inputs. In many cases, however, we can create much better verification algorithms. For example, the (g, g^x) construction[8] of [23] can simply be verified by checking whether g and g^x are elements in the desired group G of prime order, because there is a unique discrete log of g^x so the program does implement a point circuit as desired. This results in a *perfect verifier* that accepts its input program if and only if it has the form of a program produced by the obfuscator.

Now we create a definition of non-malleability for verifiable obfuscators. As before, we consider an adversary that takes an obfuscated circuit as input and outputs a program. In this model, the adversary succeeds only if her output program passes the verification test and is related to the input program. Our definition of non-malleability requires that a simulator succeeds with approximately the same probability, so it must also output a program that passes the verification test. In particular, we no longer give an oracle to the program constructed by the simulator. A formal definition follows.

Definition 5.5. Let $\mathcal{C}$ and $\mathcal{D}$ be families of polynomial-size circuits such that $\mathcal{C} \subseteq \mathcal{D}$, and let $\mathcal{O}$ and V be PPT algorithms. We say that $(\mathcal{O}, V)$ is *an obfuscator for $\mathcal{C}$ that is tamper-evident over $\mathcal{D}$* if the following four conditions hold:

1. *Verification:* V is a verifier for $\mathcal{O}$ applied to $\mathcal{C}$. Additionally, for every n and every circuit P with n bits of input such that $V(P) = 1$, there exists $D \in \mathcal{D}_n$ such that $P \equiv D$.

2. *Almost exact functionality*: There exists a negligible function ε such that for every n and every circuit $C \in \mathcal{C}_n$,
$$\Pr[\mathcal{O}(C; r) \equiv C] > 1 - \varepsilon(n),$$
where the probability is taken over the randomness r.

3. *Polynomial slowdown*: There exists a polynomial ξ such that for every n, every circuit $C \in \mathcal{C}_n$, and every possible sequence of coin tosses r, the description length $|\mathcal{O}(C; r)| \leq \xi(n)$.

4. *Tamper-evidence*: for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$,
$$\Big|\Pr[P \leftarrow A(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1,\, \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P,\, E(C, D) = 1] - \Pr\left[Q \leftarrow S^C(1^n, z) : V(Q) = 1,\, \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q,\, E(C, D) = 1\right]\Big| < \frac{1}{\rho(n)},$$
where the runtime of A and S is polynomial in the length of their first input.

If $\mathcal{D} = \mathcal{C}$, we say that $(\mathcal{O}, V)$ is a *tamper-evident obfuscator for $\mathcal{C}$*.

[8] See Section 3.2.1 for a full explanation of this construction.

It is reasonable to relax the definition by not subjecting the simulator to the verification text. We choose not to do so because the current definition puts the adversary and simulator on more equal footing and because all of our constructions satisfy the stronger notion.

We also note that the definition requires that V only accept circuits that are equivalent to members of $\mathcal{D}$. The benefit of this restriction is that the adversary can efficiently test whether she outputs a circuit in $\mathcal{D}_n$, which is a requirement for her to succeed under this definition.

Additionally, the remarks pertaining to tamper-proof obfuscation also apply here:

1. The circuits in $\mathcal{C}$ and $\mathcal{D}$ are deterministic.

2. Restricting to univariate relations $E(D)$ results in an equivalent definition.

3. It is usually impossible to achieve tamper-evident obfuscation if $\mathcal{D}$ is the family of all circuits. The proof of this statement is identical to the proof of Theorem 5.2 in the tamper-proof obfuscation setting, so we do not repeat it here.

4. Tamper-evidence also implies the virtual black-box property.

Theorem 5.6. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families, and let $(\mathcal{O}, V)$ be an obfuscator for $\mathcal{C}$ that is tamper-evident over $\mathcal{D}$. Then, $\mathcal{O}$ is an obfuscator for $\mathcal{C}$ with dependent auxiliary information.*

This theorem also holds if neither definition allows auxiliary input, or if both properties require only negligible difference in the success probabilities of the adversary and simulator.[9]

This proof is very similar to the one in the tamper-proof setting, with two differences. First, the condition that $\mathcal{C} \subseteq \mathcal{D}$ makes the technical conditions from Theorem 5.3 unnecessary here: it is always true that either $\mathcal{D}_n$ is non-trivial or $\mathcal{C}_n$ is trivial. Second, in the previous proof, we created an adversary A' that outputs one of two circuits $D_0, D_1 \in \mathcal{D}_n$. In the tamper-evident setting, this no longer suffices because A' must output a circuit that passes the verification test in order to succeed. The only circuits that are guaranteed to pass the verification test with high probability are valid obfuscations of circuits in $\mathcal{C}_n$, so we set A' to output circuits of this form.

Proof. Let A be an adversary with binary output and let ρ be a polynomial. We need to construct a simulator S for A that satisfies the virtual black-box property. We prove this statement in two cases.

First, consider all n such that $\mathcal{D}_n$ is non-trivial, in the sense that there exist two circuits in $\mathcal{D}_n$ that do not have the same functionality. Choose a circuit $C_1 \in \mathcal{C}_n$ arbitrarily, and let $D_1 = \mathcal{O}(C_1)$ be an obfuscation of C_1 such that $D_1 \equiv C_1$ (this happens with overwhelming probability by almost exact functionality). By non-triviality, there exists a circuit $D_0 \in \mathcal{D}_n$ such that $D_0 \not\equiv D_1$. Hence, there exists at least one input value x and one bit position i such that $(D_0(x))_i \neq (D_1(x))_i$. Let $c = (D_1(x))_i$.

We use A to form an adversary A' that is not constrained to binary output in the following manner. The adversary A' nonuniformly hardcodes A, D_0, and D_1, and operates as follows upon receiving $\mathcal{O}(C)$ and z as input:

1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit b
2: **output** D_b

In particular, if A outputs 1, then A' outputs a well-formed obfuscation of a circuit in $\mathcal{C}_n$, whereas if A outputs 0, then A' outputs a circuit in $\mathcal{D}_n$ with different functionality.

[9] We thank Ronny Dakdouk for suggesting this theorem.

Given an adversary A' and polynomial 2ρ, tamper-evidence guarantees the existence of a simulator S' such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all PPT relations E,

$$\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } E(C, D) = 1\right]\Big| < \frac{1}{2\rho(n)}. \quad (5.3)$$

Finally, we construct a simulator S with binary output that hardcodes V, x, i, and c and operates as follows, when given auxiliary information z and access to an oracle C:

emulate an execution of $(S')^C(1^n, z)$ until it returns a string s
if s is not the valid encoding of a circuit **then**
  output 0
end if
let Q be the circuit that is represented by s
if $V(Q) = 0$ **or** $(Q(x))_i \neq c$ **then**
  output 0
else
  output 1
end if

We now show that this simulator S satisfies the virtual black-box property.

Let $\tilde{E}(C, D)$ be a relation that accepts if and only if $D(x)_i = c$. Note that $\tilde{E}$ has x and i nonuniformly hardcoded, and its output is independent of C. Then, for all $n \in \mathbb{N}$, $C \in \mathcal{C}_n$, and $z \in \{0,1\}^*$,

$$\Pr\left[A(\mathcal{O}(C), z) = 1\right] = \Pr\left[A'(\mathcal{O}(C), z) \text{ outputs an obfuscation of } C_1\right]$$

as an immediate consequence of the construction of A'. As a result, we claim that

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \tilde{E}(C, D) = 1\right]\Big|$$

is negligible. To see this, first note that A' always outputs a program that is equivalent to a member of $\mathcal{D}_n$. Furthermore, by definition the verifier V accepts D_1. There is only one small technical problem: in the case that $C \equiv C_1$, the output of A' might equal its input, which is not allowed in tamper-evident obfuscation. However, A' receives a random obfuscation of C_1, so it only receives D_1 with negligible probability.

Next, using (5.3) and the behavior of the relation $\tilde{E}$, it follows that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary information z,

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = c\right]\Big| < \frac{1}{2\rho(n)}. \quad (5.4)$$

Second, let $\hat{E}$ be the relation such that $\hat{E}(C, D) = 1$ if and only if $D(x)_i = 1 - c$. Using the

same analysis as above, it follows that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 0\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = 1 - c\right]\right| < \frac{1}{2\rho(n)},$$

where the only difference from before is that the behavior of $V(D_0)$ is unclear, whereas the behavior of $V(\mathcal{O}(C_1))$ is known. Applying the triangle inequality to the previous two inequalities, and using the fact that A always outputs either 0 or 1, it follows that

$$\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q\right] > 1 - \frac{1}{2\rho(n)}.$$

Finally, by the construction of S from S', it is easy to see that

$$\Pr\left[S^C(1^n, z) = 1\right] = \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } (Q(x))_i = c\right].$$

Furthermore, we showed that Q fails to be equivalent to a member of $\mathcal{D}_n$ with small probability, so

$$\begin{aligned}&\left|\Pr\left[S^C(1^n, z) = 1\right]\right.\\ &\quad\left. - \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = c\right]\right| < \frac{1}{2\rho(n)}.\end{aligned}$$

Applying the triangle inequality to this inequality and (5.4) yields

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

so S satisfies the virtual black-box property as desired.

Now we prove the theorem for all n such that $\mathcal{D}_n$ is trivial, meaning that all of the circuits in $\mathcal{D}_n$ have the same functionality. In this case, we claim that for every adversary A with binary output, for every $z \in \{0,1\}^*$, for sufficiently large n, and for every $C, C' \in \mathcal{C}_n$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right|$$

is negligible. If this is true, then it is easy to construct a simulator S for A: the simulator simply chooses a circuit $C \in \mathcal{C}_n$ arbitrarily, obfuscates it, and emulates an execution of A on the obfuscated circuit. It is straightforward to check that this simulator satisfies the virtual black-box property.

Suppose for the sake of contradiction that the claim is not true. Therefore, there exist an adversary A, a polynomial ρ, and auxiliary information $z \in \{0,1\}^*$ such that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ such that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right| > \frac{1}{\rho(n)}.$$

Now form an adversary A' (that is not constrained to binary output) as follows. Arbitrarily choose a circuit $C_1 \in \mathcal{C}_n$, and let D_0 be a circuit that is not equivalent to C_1, and hence not equivalent to any member of $\mathcal{D}_n$. The adversary A' hardcodes A, D_0, and D_1, and operates as follows upon receiving $\mathcal{O}(C)$ and z as input:

1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit b
2: **if** $b = 1$ **then**
3: **output** an obfuscation of C_1

4: **else**
5: **output** D_0
6: **end if**

Let $\bar{E}$ be the relation that always accepts. As a result,

$$\Big|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right]\Big|$$

is negligible, because C_1 is equivalent to a member of $\mathcal{D}_n$ and passes the verification test, whereas D_0 is not equivalent to a member of $\mathcal{D}_n$. The only problem is that the output of A' might equal its input, but this occurs with negligible probability.

Hence, by our assumption above, it follows that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ and a polynomial ρ such that

$$\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] - \Pr\left[P \leftarrow A'(\mathcal{O}(C'), z) : P \neq \mathcal{O}(C),\, V(P) = 1,\, \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P,\, \bar{E}(C', D) = 1\right]\Big| > \frac{1}{2\rho(n)}.$$

However, for any simulator S',

$$\Pr\left[Q \leftarrow (S')^{C}(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C, D) = 1\right] = \Pr\left[Q \leftarrow (S')^{C'}(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C', D) = 1\right]$$

information-theoretically because C and C' are functionally equivalent. Applying the triangle inequality to the previous two equations, it must be the case that either

$$\Big|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C),\, V(P) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] - \Pr\left[Q \leftarrow (S')^{C}(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C, D) = 1\right]\Big| > \frac{1}{4\rho(n)},$$

or the same is true of C'. As a result, S' fails to satisfy the tamper-evidence property. This is true for all possible S', so $\mathcal{O}$ is not tamper-evident over $\mathcal{D}$, which is a contradiction.

This proof also goes through if neither tamper-evidence nor the virtual black-box property allows auxiliary input, or if both properties require only negligible difference between the success probabilities of the adversary and simulator. □

5.2.4 Models with setup assumptions

In this section, we modify the non-malleability definitions to fit within the random oracle and common reference string models.

Random Oracle Model

In the random oracle model, every algorithm has access to a length-preserving permutation $R : \{0,1\}^n \to \{0,1\}^n$ chosen uniformly at random from the set of such permutations. We only consider a non-programmable model because the definitions lose much of their value if the simulator is allowed to program the random oracle; for example, in our motivating example from the Introduction, there

are "tamper-proof" obfuscators of three-point circuits that allow Alice to remove Bob's access to the computer.

Definition 5.7. A PPT algorithm $\mathcal{O}$ satisfies the tamper-proofing property in the random oracle model if for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$,

$$\Big|\Pr\left[P \leftarrow A^R(\mathcal{O}(C)^R, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P^R \text{ and } E^R(C,D) = 1\right] - \Pr\left[Q \leftarrow S^{C,R}(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^{C,R} \text{ and } E^R(C,D) = 1\right]\Big| < \frac{1}{\rho(n)},$$

where the probabilities are over the coin tosses of A, $\mathcal{O}$, and S, as well as the choice of $R : \{0,1\}^n \to \{0,1\}^n$ from the uniform distribution over all permutations. As before, we require that the runtime of A and S is polynomial in the length of their first input.

Definition 5.8. The pair $(\mathcal{O}, V)$ satisfies the tamper-evidence property in the random oracle model if for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{C}_n \to \{0,1\}$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\Big|\Pr\left[P \leftarrow A^R(\mathcal{O}(C)^R, z) : P \neq \mathcal{O}(C),\, V^R(P^R) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P^R,\, E^R(C,D) = 1\right] - \Pr\left[Q \leftarrow S^{C,R}(1^n, z) : V(Q^R) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^R \text{ and } E^R(C,D) = 1\right]\Big| < \frac{1}{\rho(n)},$$

where the runtime of A and S is polynomial in the length of their first input.

Note that the relation E and verifier V in these definitions receives the same random oracle that the adversary and simulator receive. As a result, the definitions impose a nonprogrammability restraint on the simulator because E and V collectively serve as an "environment" (in the universal composability sense [24]) that can try to detect any differences in the behavior of the adversary and simulator.

Common Reference String Model

In the common reference string model, a long string Σ is generated by a trusted party with each bit being chosen independently and uniformly at random. In this work, we only explore tamper-evident obfuscation in the CRS model, and we augment the definition to allow all algorithms in the real world access to the string Σ. In the simulated world, S is allowed to choose Σ.

Definition 5.9. The pair $(\mathcal{O}, V)$ satisfies the tamper-evidence property in the common reference string model if there exists a polynomial l such that for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{C}_n \to \{0,1\}$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\Big|\Pr\left[P \leftarrow A(\mathcal{O}(C), z, \Sigma) : P \neq \mathcal{O}(C),\, V(P,\Sigma) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P,\, E(C,D) = 1\right] - \Pr\left[(Q,\Sigma) \leftarrow S^C(1^n, z) : V(Q,\Sigma) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q,\, E(C,D) = 1\right]\Big| < \frac{1}{\rho(n)},$$

where the runtime of A, S, and V is polynomial in the length of their first input. The first probability is taken over the random coin tosses of A and $\mathcal{O}$ along with the random choice of $\Sigma \xleftarrow{U} \{0,1\}^{l(n)}$, and the second probability is taken over the random coin tosses of S.

Note that in the real execution of the adversary, the verifier receives the same common reference string as the adversary, but in the simulated world, S chooses the CRS for the verifier.

5.2.5 Comparison

We conclude this section by showing that the two non-malleability definitions are incomparable extensions of the virtual black-box property, at least in the random oracle model. Note that the virtual black-box property can be incorporated into the random oracle model in the same way as the non-malleability properties were, and both flavors of non-malleability still imply the virtual black-box property in the random oracle model.

Theorem 5.10. *Tamper-proof and tamper-evident obfuscation are incomparable in strength. This holds whether auxiliary input is permitted or not. Specifically,*

1. *In the random oracle model, there exists a circuit family $\mathcal{C}$ and a tamper-evident obfuscator $(\mathcal{O}, V)$ for $\mathcal{C}$ such that $\mathcal{O}$ is not a tamper-proof obfuscator for $\mathcal{C}$.*

2. *Suppose there exists a tamper-proof obfuscator $\mathcal{O}$ for an unlearnable family $\mathcal{C}$ (as defined in Theorem 5.2). Then, there exists another tamper-proof obfuscator $\mathcal{O}'$ for $\mathcal{C}$ such that no verifier V makes the pair $(\mathcal{O}', V)$ a tamper-evident obfuscator for $\mathcal{C}$.*
   Note that this direction does not rely on the random oracle model, and holds in the standard model as well. However, the only known constructions of tamper-proof obfuscators are in the random oracle model.

Later in this chapter, we construct counterexamples for the family of multi-point circuits that do not require auxiliary input, so this theorem follows immediately from Theorems 5.12, 5.15, and 5.17. Here, we focus on the high-level ideas behind the theorem.

Intuitively, tamper-proofing *prevents* attacks, whereas tamper-evidence has no prevention guarantee. This is reflected in the definition by the fact that an adversary attacking tamper-proof obfuscation does not have to pass a verification test, so the simulator must emulate more potential attacks. A concrete example of this separation is the tamper-evident obfuscator described in Algorithm 5.5 below, which does not prevent the attack described in the Introduction in which Alice removes Bob's password.

On the other hand, tamper-evidence must *detect* attacks that cannot hope to be prevented, which may not always be possible. This is reflected in the definitions by the fact that the program Q constructed by the simulator is not given oracle access to C in the verifiable definition. We provide a general separation in this direction below (which does not rely on the random oracle model) based on a modified form of Alice's April fools' prank described in the Introduction.

Proof. We defer the proof of part 1 to Theorem 5.17. Here, we prove part 2, which shows that tamper-proofing does not imply tamper-evidence. Let $\mathcal{C}$ be an unlearnable family of circuits, and let $\mathcal{O}$ be a tamper-proof obfuscator for $\mathcal{C}$. We form a slightly modified obfuscator $\mathcal{O}'$ that operates identically to $\mathcal{O}$, except that it adds a randomly-chosen bit of storage b to the circuit that it outputs. Hence, $\mathcal{O}'$ outputs circuits of the form $\langle \mathcal{O}(C), b \rangle$, where the bit b is not used in the operation of the circuit. Note that $\mathcal{O}'$ is a tamper-proof obfuscator because the extra bit of storage does not impact the functionality or runtime of the program that the obfuscator outputs, nor does it convey any

information about the underlying circuit, so it does not impact any of the three properties that define tamper-proof obfuscators.

On the other hand, $\mathcal{O}'$ is trivially vulnerable to a tamper-evidence attack. Let A be an adversary (without auxiliary input) that receives an obfuscated circuit $\langle \mathcal{O}(C), b \rangle$ and simply flips the storage bit. Because $\langle \mathcal{O}(C), 1-b \rangle$ is a valid output of the obfuscator $\mathcal{O}'$, any verifier for $\mathcal{O}'$ must accept it. By almost exact functionality, $\langle \mathcal{O}(C), 1-b \rangle \equiv C$ with overwhelming probability.

Consider the relation E that accepts if and only if its two circuits are identical. The adversary always satisfies this relation. As a result, a tamper-evident simulator S^C must output the description of the circuit C. However, we know that for every PPT simulator S (even with auxiliary input), there exists a sequence of circuits in $\mathcal{C}$ that S cannot learn, so S fails to satisfy the tamper-evidence property, as desired. □

5.3 Constructions of tamper-proof obfuscators

In this section, we present tamper-proof obfuscators for the family of multi-point circuits in the random oracle model. We begin with the single-point case.

In order to achieve exact functionality whenever possible in our constructions, we assume that $R : \{0,1\}^n \to \{0,1\}^n$ is a length-preserving permutation in most of this chapter. Using a random oracle function instead would suffice for security, but then our constructions would only achieve almost exact functionality.

5.3.1 Single-point circuits

Algorithm 5.1 describes an obfuscator $\mathcal{O}_{\mathcal{P}^1}$ for the family of point circuits $\mathcal{P}^1$ [63].

Algorithm 5.1 Obfuscator $\mathcal{O}_{\mathcal{P}^1}$ for the family of point circuits

Input: a circuit of the form I_w or $I_\emptyset$

1: **if** the input circuit is $I_\emptyset$ **then**
2: choose randomness $r \xleftarrow{U} \{0,1\}^{3|w|}$ and $t \xleftarrow{U} \{0,1\}^{4|w|}$
3: **else**
4: extract the point w and choose randomness $r \xleftarrow{U} \{0,1\}^{3|w|}$
5: compute $t = R(w \circ r)$, where $\circ$ denotes the string concatenation operation
6: **end if**

Output: the circuit $\Phi_{r,t}$ that stores r and t in some clearly identifiable manner, and on input a string x, outputs 1 if $R(x \circ r) = t$ and 0 otherwise

Theorem 5.11. *In the random oracle model, the algorithm $\mathcal{O}_{\mathcal{P}^1}$ is a tamper-proof obfuscator for the family of point circuits $\mathcal{P}^1$.*

Proof. The almost exact functionality of $\mathcal{O}_{\mathcal{P}^1}$ follows from the fact that R is a permutation (in fact, functionality is exact for all input circuits except $I_\emptyset$). Polynomial slowdown clearly holds because the runtime of $\Phi_{r,t}$ is linear in its input length. It remains to prove the tamper-proofing property.

To prove that $\mathcal{O}_{\mathcal{P}^1}$ is tamper-proof, we need to construct a simulator S for any adversary A. We informally explain S here and give a complete description of S in Algorithm 5.2.

The simulator has auxiliary information z and oracle access to the point circuit I_w (where for notational simplicity we allow for the possibility that $w = \emptyset$, rather than referring to $I_\emptyset$ explicitly each time). It makes a fake obfuscated circuit $\Phi_{r,t}$ by choosing r and t uniformly at random, and

Algorithm 5.2 Algorithm for the simulator $S^{R,I_w}(1^n, z)$ in the proof of Theorem 5.11

Input: inputs N and z, along with access to oracles R and C

set $n \leftarrow |N|$
choose random $r \xleftarrow{U} \{0,1\}^{3n}$ and $t \xleftarrow{U} \{0,1\}^{4n}$
repeat
  emulate one step of the execution of $A(\Phi_{r,t}, z)$
  if A queries its random oracle on input value q **then**
    if $R(q) = t$ **then**
      output $\perp$
    else if $|q| \neq 4n$ **or** the last $3n$ bits of q are not equal to r **then**
      respond to the query with $R(q)$
    else
      set y to the first n bits of q
      if $C(y) = 1$ **then**
        abort this emulation
        emulate $A(\Phi_{r,R(y \circ r)})$, responding to all oracle queries accurately, then goto step 21
      else
        respond to the query with $R(q)$
      end if
    end if
  end if
until A halts
set P to the output of the emulation $A(\Phi_{r,t}, z)$ from step 20, or the output of $A(\Phi_{r,R(y \circ r)})$ from step 14
choose a string $t' \xleftarrow{U} \{0,1\}^{4n}$
use P, r, t, and t' to develop the program Q, which for legibility is written below

Output: a description of Q

Algorithm 5.3 Algorithm for program Q^{R,I_w} in the proof of Theorem 5.11

Input: a string x, along with access to oracles R and C

repeat
  emulate one step of the execution of $P^R(x)$
  if A queries its random oracle on input value q **then**
    if $|q| \neq 4n$ **or** the last $3n$ bits of q are not equal to r **then**
      respond to the query with $R(q)$, unless this equals t, then respond with t'
    else
      set y to the first n bits of q
      if $C(y) = 1$ **then**
        respond to the query with t
      else
        respond to the query with $R(q)$, unless this equals t, then respond with t'
      end if
    end if
  end if
until P halts

Output: the value that the emulation of P outputs

then emulates an execution of $A^R(\Phi_{r,t}, z)$. In addition, S examines all of A's random oracle queries. There are three cases to consider:

1. If A queries R on the input $w \circ r$, then the adversary A found the hidden point w, but now the simulator has too. As a result, the simulator aborts this emulation of A, forms a valid obfuscation of I_w, runs A on it, and outputs the resulting program.

2. If A queries the value $s = R^{-1}(t)$, then A has discovered that our obfuscation is fake. In this case, the simulator fails. However, t is chosen uniformly at random and independently of A and z, so this case only occurs with negligible probability.

3. On any other query q, S responds honestly with $R(q)$.

Note that S either responds to oracle queries honestly or aborts the emulation of A.

Once the emulation of A terminates, it outputs a program P. Then, the simulator S has to output a program Q^{R,I_w}. We explain here how Q^{R,I_w} operates and provide a complete description in Algorithm 5.3.

The program Q^{R,I_w} runs P^R but continues to examine oracle calls. Once again, there are three cases to consider:

1. If P queries R on the input $w \circ r$, then Q answers the query with t.

2. If P queries the value $s = R^{-1}(t)$, then Q answers the query with a hardcoded response t' that is uniformly chosen from $\{0,1\}^{4n}$.

3. Otherwise, Q gives the correct response to P.

We emphasize here that Q is deterministic, as required. Now we analyze the above simulator and show that it satisfies the tamper-proofing property. We break the analysis of the simulator into cases based on the random oracle queries made by the emulated adversary. If the emulated adversary queries the correct password, or if the input circuit was $I_\emptyset$, then the emulation of the adversary is perfect. Additionally, the emulated adversary runs in polynomial time, so it can only query the random oracle on $s = R^{-1}(t)$ with negligible probability. Therefore, it suffices to restrict our attention to the case in which neither of these situations occurs.

Let E be any relation that only makes polynomially-many oracle queries, and define

$$\gamma_A = \Pr\left[C \leftarrow A^R(\mathcal{O}(I_w), z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv C^R \text{ and } E^R(I_w, I_{w'}) = 1\right]$$
$$\gamma_S = \Pr\left[D \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv D^{R,I_w} \text{ and } E^R(I_w, I_{w'}) = 1\right]$$

as the probabilities of success for the real adversary and simulator, respectively. We will show that $|\gamma_A - \gamma_S|$ is negligible in n. To do so, we perform the following "mental experiment": let R' be a different random oracle permutation in which

$$R'(w \circ r) = t, \qquad R'(s) = R(w \circ r),$$

and R' agrees with R on all other values (see Figure 5-1 for a diagram). If R' really were the random oracle, then the "fake" obfuscation $\Phi_{r,t}$ would actually be a valid obfuscation of I_w. Thus,

$$\gamma_A = \Pr\left[P \leftarrow A^{R'}(\Phi_{r,t}, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

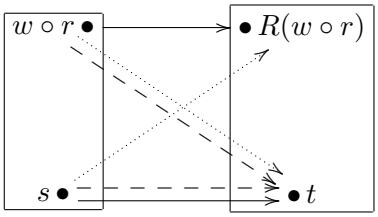

Figure 5-1: Pictorial representation of the actions of the actual random oracle R (solid line) and the two imaginary oracles R' (dotted line) and R'' (dashed line). The domain and range are both $\{0,1\}^{4|w|}$. The three random oracles operate identically on all other values.

Furthermore, by assumption A never queries the oracle at the two locations in which R and R' differ. Hence, A must act the same in both cases, so the circuit P returned by the emulation of A^R on the "fake" obfuscation is identical to the one returned by $A^{R'}$ when given a real obfuscation.

Consider a new simulator S' that operates just like S, except that after it finishes its emulation of the adversary, it simply returns the circuit P that the emulated adversary outputs. Then,

$$\gamma_A \approx \Pr\left[P \leftarrow (S')^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right],$$

where the symbol $\approx$ is used to denote the fact that the two probabilities may differ by the negligible probability that the emulated adversary queries its random oracle on $s = R^{-1}(t)$, forcing the simulator to abort.

Of course, the simulator S constructed above does not return P, but rather a related circuit Q that makes a few modifications to random oracle queries. These modifications are precisely the ones that make $P^{R'} \equiv Q^R$, except in the case that P queries the value $s = R^{-1}(t)$. Because this happens with negligible probability,

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Next, let R'' be an oracle in which $R''(w \circ r) = t$, and R'' agrees with R on all other values. (See Figure 5-1 for a diagram.) In other words, R'' and R' differ only on the input s. We consider the difference in functionality between $E^{R'}$ and $E^{R''}$. Because the adversary, simulator, and relation E all run in polynomial time, they only query the value s with negligible probability. Therefore,

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R''}(I_w, I_{w'}) = 1\right].$$

Finally, we claim that there is only a negligible difference in functionality between E^R and $E^{R''}$. This claim is information-theoretic: R and R'' only differ on input $w \circ r$, which has at least $3n$ bits of entropy, but the relation E only receives $2n$ bits of information as input (namely, the values w and w'). As a result, using any list encoding scheme, the adversary and simulator only have negligible probability of transmitting the value r to the relation E. If the relation does not query the value $w \circ r$, then the behavior of E^R and $E^{R''}$ is identical. As a result,

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R}(I_w, I_{w'}) = 1\right] = \gamma_S,$$

as desired.

Hence, $\mathcal{O}_{\mathcal{P}^1}$ is a tamper-proof obfuscator for $\mathcal{P}^1$, and in fact the definition is achieved in a strong sense in which the simulator S is independent of the choice of the polynomial ρ. □

Tamper-proofing over a larger family

In the preceding proof, we showed that $\mathcal{O}_{\mathcal{P}^1}$ is an obfuscator for $\mathcal{P}^1$ that is tamper-proof over $\mathcal{P}^1$. That is, the output family for the adversary is the same as the family being obfuscated. However, as suggested in Section 5.2.2, we may wish to prevent the adversary from creating bigger circuits like two-point circuits, and thus prevent attacks such as adding a password to the currently accepted one.

Our construction for $\mathcal{O}_{\mathcal{P}^1}$ can easily be modified to allow this. Note that the only place where we used the fact that the adversary outputs a point circuit is in the information-theoretic argument at the end of the proof, where we note that E receives $2n$ bits of input and thus cannot learn r which is $3n$ bits long. Hence, we can easily accommodate a larger output family like $\mathcal{P}^2$ by increasing the length of r (and thus t as well), and the argument will still go through.

More precisely, we can accommodate any output family $\mathcal{D}$ by setting the length of r to be $2n + \log(|\mathcal{D}|)$ and the length of t to be $3n + \log(|\mathcal{D}|)$ in Algorithm 5.1. In particular, we can accommodate any family of multi-point circuits $\mathcal{P}^m$ by setting $|r| = (m+2)n$ and $|t| = (m+3)n$. Note though that we must pick the output family that we wish to consider *before* obfuscating the circuit.

Removing tamper-evidence

As we will see later in Theorem 5.16, this obfuscator is also tamper-evident. However, we can apply the trick from Theorem 5.10 to form a new obfuscator that is not tamper-evident. Let $\mathcal{O}'_{\mathcal{P}^1}$ be an obfuscator that operates identically to Algorithm 5.1 except that it also hardcodes one randomly chosen bit b into the circuit that it outputs.

Theorem 5.12. *The obfuscator $\mathcal{O}'_{\mathcal{P}^1}$ is tamper-proof, but no verifier V makes $(\mathcal{O}'_{\mathcal{P}^1}, V)$ tamper-evident.*

Proof. This theorem is a specific instantiation of the general concepts from Theorem 5.10. The proof that $\mathcal{O}'_{\mathcal{P}^1}$ is tamper-proof is identical to the proof for $\mathcal{O}_{\mathcal{P}^1}$ above. Additionally, $\mathcal{O}'_{\mathcal{P}^1}$ is trivially vulnerable to a tamper-evidence attack by an adversary that simply flips the bit b stored in the obfuscated circuit that she receives. Specifically, the adversary receives an obfuscated circuit $\Phi_{r,t,b} = \mathcal{O}(I_w)$ and outputs $\Phi_{r,t,1-b}$. This circuit is equivalent in functionality to I_w, and it has the format of a well-formed circuit made by the obfuscator, so any verifier V must accept it. Consider the relation $E(I_w, I_{w'})$ that accepts if and only if $w = w'$. The adversary always satisfies this relation, so for tamper-evidence to hold, a PPT simulator must be able to satisfy it as well. However, a simulator $S^{I_w}(1^n)$ cannot output a circuit that is equivalent to I_w for any $w \in \{0,1\}^n$, so tamper-evidence fails. □

5.3.2 Multi-point circuits

Constructing a tamper-proof obfuscator for the family of multi-point circuits $\mathcal{P}^m$ is significantly more difficult. Roughly speaking, the principal issue is that the obfuscated program must "bundle together" the m points in a way that would prevent the adversary from changing any point in the bundle without applying the exact same change to all points in the bundle. For instance, simply concatenating m obfuscations of a single-point circuit (even obfuscations that are individually non-malleable) does not suffice because an adversary can change some of the points at will.

Instead, the code of the program must be a "house of cards" in the sense that an adversary cannot change the code without destroying information about all of the accepted points simultane-

ously. This is a difficult task to achieve, so we first present some warm-up examples in which the obfuscator receives special abilities.

First warm-up

In this example, we assume the existence of an m-to-1 random oracle, although this may easily be constructed from a permutation. Additionally, we augment the model by making two unrealistic assumptions.

1. Rather than receiving a list of accepted points $\{w_1, \ldots, w_m\}$, the obfuscator chooses these points on her own from the uniform distribution. An obfuscator of this type may still be useful in many situations, such as the motivating scenario from the Introduction in which several graduate students wish to share a computer. The students may allow the network administrator to choose passwords for them, as long as the passwords are chosen uniformly.

2. The obfuscator receives a special auxiliary input consisting of a random point t and all m preimages of t under the random oracle. Nobody else in the world receives this auxiliary input, which can be thought of as "trapdoor" information on the random oracle.

In this setting, the obfuscator can do the following.

1: choose a target point $t \leftarrow \{0,1\}^n$ uniformly at random
2: compute the m inverses of t, and denote them as $w_1, \ldots, w_m$
3: output the following circuit: "accept an input x if and only if $R(x) = t$"
4: distribute passwords to the appropriate entities

Note that the output of this algorithm is clearly an m-point circuit, since the random oracle is m-to-1. Furthermore, the only information stored by the circuit is t. This value "glues" together all m passwords, in the sense that an adversary cannot modify t without destroying all of the information about every password simultaneously.

Second warm-up

While the above protocol is clean and simple, it operates using unrealistic assumptions that we would like to remove. In the next warm-up, we return to the setting in which the obfuscator receives a set of accepted points $\{w_1, \ldots, w_m\}$, rather than being able to choose them on her own. However, the obfuscator is still able to invert the random oracle at a single point t.

Suppose the obfuscator starts as before by computing the m preimages of t under the random oracle, which we denote by $w'_1, w'_2, \ldots, w'_m$. It would be nice if the real accepted points w_i were equal to the w'_j, for then the obfuscator could use the first warm-up to complete her task, but sadly this is not the case.

Hence, the obfuscator needs a way to "link" the real accepted points to the fake ones. One way to do this is to use m obfuscated point circuits with multi-bit output [25] constructed such that someone who knows w_i is able to learn the string w'_i. Unfortunately, this method is insufficient in the tamper-proof obfuscation setting because an adversary can destroy some of the obfuscated links.

Instead, we use a polynomial. Specifically, view the strings w_i and w'_i as elements of a finite field of large ($\approx 2^n$) prime order, and let ξ be the unique degree $m-1$ polynomial over this field such that $\xi(w_i) = w'_i$ for all i. Then, the obfuscator creates a circuit that accepts its input x if and only if $R(\xi(x)) = t$. Because the w'_i are uniformly chosen, it follows that ξ is taken from the uniform distribution over all degree $m-1$ polynomials. We show the algorithm pictorially in Figure 5-2.

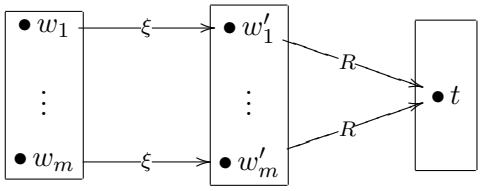

Figure 5-2: Pictorial representation of the tamper-proof m-point circuit in the second warm-up.

In this algorithm, the polynomial q "glues" together m passwords, in the sense that it is not possible for an adversary to change the polynomial without destroying information on all the points simultaneously, except for those points that the adversary already knows. There are two technical issues that prevent this algorithm from being an obfuscator.

1. There is no randomness in the obfuscation. That is, if the obfuscator receives the same set of passwords twice, then she will output the same circuit both times. As shown in [23], a deterministic algorithm inherently cannot be an obfuscator.

2. The obfuscated circuit does not preserve functionality. By construction, it accepts all of the w_i, but it accepts additional values too. In particular, it accepts all points x such that $\xi(x)$ equals any of the w'_i, and there are up to $m(m-1)$ such points.

Luckily, both of these issues can be resolved simultaneously with a simple change to the algorithm. The obfuscator needs to choose a long random string r, and apply the previous algorithm to the strings $\{w_1 \circ r, \ldots, w_m \circ r\}$ instead of just the passwords $\{w_1, \ldots, w_m\}$. As a result, the obfuscator outputs a circuit of the form "on input x, accept if and only if $R(\xi(x \circ r)) = t$." With overwhelming probability, the other inverses of the polynomial ξ will not be strings that end in r, so the obfuscation has exact functionality.

Recall that the obfuscator still requires special "trapdoor" information about the inverses of t under the random oracle. Hence, the construction operates in a "trapdoor random oracle model" with two oracles: one that computes a random m-to-1 function, and another one that takes no input and, when invoked, returns a uniformly random string t and all of the inverses of t under the random oracle. It can be proved that our construction yields a tamper-proof obfuscator for $\mathcal{P}^m$ in this model, although we do not do so here because the model is quite strong, and as such, still undesirable.

Full protocol for $m = 2$

Now we remove the second unrealistic assumption and return to the usual random oracle model, in which the obfuscator does not receive any special information about inverses to the random oracle.

In the second warm-up, R played two roles. In addition to its use as a random oracle, it also serves as a sort of trapdoor one-way function because the obfuscator receives a special inversion power. In this protocol, we separate the two duties into a random oracle function[10] and a special family of lossy one-way functions with second preimage resistance and a trapdoor-like property. Informally, we desire functions $f : \{0,1\}^{6n} \to \{0,1\}^{3n}$ with the following properties:

- The functions are 2^{3n}-to-1, but no adversary given $y \xleftarrow{U} \{0,1\}^{6n}$ can find y' such that $f(y') = f(y)$, even though there exist many such inputs.

[10]Note that in the rest of this chapter, the oracle is a random permutation, not just a random function. This is done to allow exact functionality whenever possible. However, as we will see, the construction in this section only permits almost exact functionality, so we simplify the analysis by merely assuming that R is a function.

- It must be possible to generate f together with a string y and a "collisions" circuit C that can enumerate all of the 2^{3n} strings that collide with y. Formally, $\forall i \in [3n]$, $f(C(i)) = f(y)$, and moreover $\forall i \neq j$, $C(i) \neq C(j)$.
- The collisions circuit C should have the property that if one chooses $i \in [3n]$ arbitrarily, the distribution on $C(i)$ is uniformly random over the set of strings that collide with y, even given f and y (but not C, so this property constrains the sampling algorithm that forms C).

In this construction, the trapdoor-like property of f takes over the corresponding responsibility of the trapdoor random oracle in the second warm-up. Additionally, having exponentially-many collisions and the circuit C will allow the simulator to produce consistent responses to random oracle queries. We will return to this issue and explain it in more detail after presenting our obfuscator.

First, we give a family of one-way functions that satisfy the desired properties based on the hardness of computing discrete logarithms. The following assumption uses the finite field $\mathbb{F}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$.

Assumption 5.13. *Let $\mathbb{G}$ be an algorithm that takes a security parameter n as input and outputs a tuple (p, G, g), where p is an n-bit prime, G is (the description of) a multiplicative group of order p, and g is a uniformly chosen generator of G. Next, choose $a \xleftarrow{U} \mathbb{F}_p$ and set $h = g^a$. The* discrete logarithm assumption *for $\mathbb{G}$ says that for any adversary A, $\Pr[A(p, G, g, h) = a]$ is negligible in n.*

We use the algorithm $\mathbb{G}$ to form a family of one-way functions as follows: sample $(p, G, g) \leftarrow \mathbb{G}(1^{3n})$, choose $a_1, a_2 \xleftarrow{U} \mathbb{F}_p$, and compute the group elements $g_1 = g^{a_1}$ and $g_2 = g^{a_2}$. Then, we let

$$f_{p,G,g_1,g_2}(y_1, y_2) = g_1^{y_1} \cdot g_2^{y_2}.$$

This is a function from two elements in $\mathbb{F}_p$ to one element in G, so it is clearly lossy. The function makes public p, G, g_1, and g_2, but not a_1 or a_2.

Furthermore, we can form a "trapdoor" of sorts using the two exponents a_1 and a_2. To explain this, it helps to switch to vector notation and let $\boldsymbol{y} = (y_1, y_2)$ and $\boldsymbol{a} = (a_1, a_2)$ be vectors in $\mathbb{F}_p^2$. If we choose a vector $\boldsymbol{y}$ uniformly at random, then $f(\boldsymbol{y}) = g^{\langle \boldsymbol{y}, \boldsymbol{a} \rangle}$ is a uniformly random group element which we call t. Furthermore, consider the vector $\bar{\boldsymbol{a}} = (a_2, -a_1)$. Because $\langle \boldsymbol{a}, \bar{\boldsymbol{a}} \rangle = 0$, it follows by linearity that for all $i \in \mathbb{F}_p$, $f(\boldsymbol{y} + i\bar{\boldsymbol{a}}) = t$. Hence, we can construct f together with $\boldsymbol{y}$, t, and a "collisions" circuit C such that $C(i)$ outputs $\boldsymbol{y} + i\bar{\boldsymbol{a}}$.

We note that this function is essentially the chameleon hash function of Krawczyk and Rabin [60], although our "input" equals both their message and randomness; that is, y_1 corresponds to the message to be hashed in their setting, and y_2 corresponds to the randomness used in the hash. The trapdoor used in their setting is similar as well. (See Section 6.6 for more on chameleon hashes.)

Theorem 5.14. *The family of functions $\{f_{p,G,g_1,g_2}\}_{(p,G,g) \leftarrow \mathbb{G}(1^{3n})}$ is one-way and second preimage resistant under Assumption 5.13.*

Proof. This theorem is a straightforward consequence of the fact that a chameleon hash function is collision-resistant [60], since collision-resistance in their framework corresponds to second preimage resistance in ours. Alternatively, the theorem follows from the fact that two copies of this construction (with the same p and G but different g's) can be used as the lossy branch of a lossy trapdoor function family [70, 77]. Hence, the function $f_{p,G,g_1,g_2} \circ f_{p,G,g_1',g_2'}$ is one-way, and it cannot be distinguished from an injective one, from which second preimage-resistance immediately follows.

These are simple statements to prove on their own though, so we do so for the sake of completeness.[11] First, we prove that f_{p,G,g_1,g_2} is one-way. Suppose for the sake of contradiction that it is not

[11] Additionally, the proof in [77] that the related family is lossy trapdoor uses the DDH assumption, whereas the self-contained proof (and the one in [60]) only needs the discrete logarithm assumption.

one-way, so there exists an adversary A that can invert it. We use this to form an adversary A' that breaks the discrete logarithm assumption. The algorithm A' receives $(p, G, g) \leftarrow \mathbb{G}$ and another group element $h = g^a$, and must compute the exponent a. Now, A' sets $g_1 = g$ and $g_2 = g^{a_2}$ for a uniformly chosen $a_2 \leftarrow \mathbb{F}_p$. Then, A' runs A to find an inverse of h under f_{p,G,g_1,g_2} (note that both h and f_{p,G,g_1,g_2} are distributed in precisely the manner that A expects). If A succeeds, then it outputs y_1, y_2 such that $h = g_1^{y_1} \cdot g_2^{y_2} = g^{y_1 + y_2 a_2}$. Finally, A' outputs $y_1 + y_2 a_2$ as the discrete log of h. It is clear that the success probability of A' is at least as high as that of A.

Second, we prove that f_{p,G,g_1,g_2} is second preimage resistant: given $(p, G, g) \leftarrow \mathbb{G}$, $g_1, g_2, t \xleftarrow{U} G$, $y_1 \xleftarrow{U} \mathbb{F}_p$, and y_2 such that $g_1^{y_1} g_2^{y_2} = t$, no adversary can find y_1', y_2' such that $g_1^{y_1'} g_2^{y_2'} = t$ as well except with negligible probability. Suppose for the sake of contradiction that this is not true, and there exists an adversary A that breaks it. Then, we form an adversary A' that breaks Assumption 5.13. As before, A' receives $(p, G, g) \leftarrow \mathbb{G}$ and another group element $h = g^a$, and sets $g_1 = g$, $g_2 = h$, and $t = g_1^{y_1} g_2^{y_2}$ for $y_1, y_2 \xleftarrow{U} \mathbb{F}_p$. Note that g_1, g_2, and t are uniformly distributed as required. Then, A' runs $A(f_{p,G,g_1,g_2}, t, y_1, y_2)$, which returns $y_1', y_2' \in \mathbb{F}_p$. If A succeeds, then $y_1 + y_2 a = y_1' + y_2' a$ with $y_1' \neq y_1$ and $y_2' \neq y_2$. Thus, A' outputs $\frac{y_1 - y_1'}{y_2' - y_2}$ and succeeds at least as often as A does. □

We provide a pictorial representation of the obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^2}$ in Figure 5-3. Note that we alter our notation slightly. In previous constructions we called the accepted points $w_1, \ldots, w_m$, but here we refer to the two accepted points as w and w', and instead use subscripts to denote vector components.

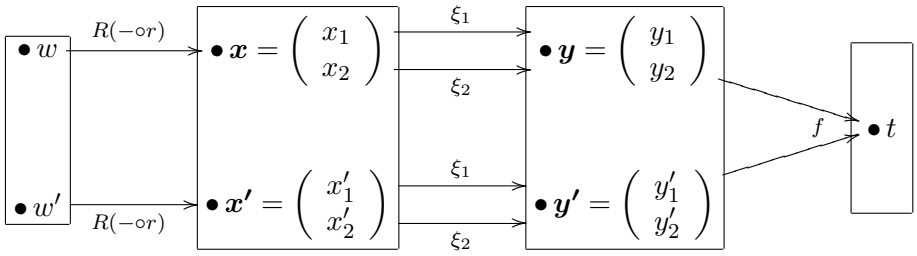

Figure 5-3: Pictorial representation of the action of an obfuscated two-point circuit created by $\bar{\mathcal{O}}_{\mathcal{P}^2}$. As before, t is the "glue" that binds the two accepted points together.

In this construction, we choose a random function f_{p,G,g_1,g_2} as specified above along with "trapdoor" information consisting of two vectors $\boldsymbol{y}$, $\boldsymbol{y'}$ that map to the same target element t. Then, an n-bit input string is concatenated with a randomly-chosen string $r \in \{0,1\}^{5n}$ and sent through the random oracle. The resulting output is a $6n$-bit long string, which we interpret as two elements in $\mathbb{F}_p$ in the natural way (remember from our construction of f that $|p| = 3n$). In particular, applying this procedure to the two accepted points yields two vectors $\boldsymbol{x} = R(w \circ r)$ and $\boldsymbol{x'} = R(w' \circ r)$.

Just like in the second warm-up, it would be nice if $\boldsymbol{x}$ and $\boldsymbol{x'}$ were equal to $\boldsymbol{y}$ and $\boldsymbol{y'}$, but sadly this is not the case. As a result, we link them together with two polynomials (which in the $m = 2$ case are just lines). Specifically, we form the line ξ_1 over $\mathbb{F}_p$ that is uniquely determined by the two constraints $\xi_1(x_1) = y_1$ and $\xi_1(x_1') = y_1'$. We form the line ξ_2 analogously. Note that each line can be identified by two group elements with overwhelming probability, since a (non-vertical) line has the form $y = mx + b$ for $m, b \in \mathbb{F}_p$.

Overall, on input u, the circuit checks whether $f(\boldsymbol{\xi}(R(u \circ r)))$ equals t, where all computations are done in the appropriate groups and $\boldsymbol{\xi}$ refers to the component-wise application of ξ_1 and ξ_2. The full construction is presented in Algorithm 5.4.

Algorithm 5.4 Obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^2}$ for the family of two-point circuits

Input: a circuit of the form $I_{\{w,w'\}}$ or $I_\emptyset$

sample $(p, G, g) \leftarrow \mathbb{G}(1^{3n})$, choose $a_1, a_2 \xleftarrow{U} \mathbb{F}_p$, and set $g_1 = g^{a_1}$ and $g_2 = g^{a_2}$
choose a random string $r \leftarrow \{0,1\}^{5n}$
if the input circuit is $I_\emptyset$ **then**
  choose a group element $t \xleftarrow{U} G$ and two lines ξ_1, ξ_2 uniformly over $\mathbb{F}_p$
else
  extract the points w, w' and randomize their order
  find $\boldsymbol{x} = R(w \circ r)$ and $\boldsymbol{x'} = R(w' \circ r)$, also choose $\boldsymbol{v} \xleftarrow{U} \mathbb{F}_p^2$ and set $t = g_1^{v_1} \cdot g_2^{v_2}$
  set $\boldsymbol{y} = \boldsymbol{v} + w \cdot \bar{\boldsymbol{a}}$, where $\bar{\boldsymbol{a}} = (a_2, -a_1)$ and we view w as an element of $\mathbb{F}_p$ in the natural way
  if $w = w'$ **then**
    in the finite field $\mathbb{F}_p$, choose lines ξ_1, ξ_2 uniformly subject to $\xi_i(x_i) = y_i$ for $i = 1, 2$
  else
    set $\boldsymbol{y'} = \boldsymbol{v} + w' \cdot \bar{\boldsymbol{a}}$
    in the finite field $\mathbb{F}_p$, find the line ξ_1 such that $\xi_1(x_1) = y_1$ and $\xi_1(x'_1) = y'_1$
    similarly, find the line ξ_2 such that $\xi_2(x_2) = y_2$ and $\xi_1(x'_2) = y'_2$
  end if
end if

Output: the circuit $\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}$ that stores all values in an easily identifiable manner and operates as follows: "on input u, compute $R(u \circ r)$, interpret this as a vector $\boldsymbol{u} \in \mathbb{F}_p^2$, compute $g_1^{\xi_1(u_1)} \cdot g_2^{\xi_2(u_2)}$ in G and accept iff this equals t"

Now we return to the issue of simulator consistency and the need for exponentially-many collisions. Recall that an important distinction between this construction and the one in the second warm-up is that the random oracle is now just a random function, rather than an m-to-1 function. As a result, the outputs to the two accepted points w and w' are different now, which causes some issues with our proof.

Just as in the one-point case (Theorem 5.11), we want the simulator to make a "fake" obfuscation that it modifies whenever it learns an accepted point, and output a circuit Q that continues to do the same. In the one-point case, this was simple: Q stores the desired target t of the random oracle, and when it receives the accepted point w as input, it changes the response of the random oracle query $R(w \circ r)$ to the target t. The same idea works in the warm-ups.

However, here the modification is not so simple, because Q must send $R(w \circ r)$ and $R(w' \circ r)$ to different outputs $\boldsymbol{x}$ and $\boldsymbol{x'}$. Moreover, choosing one of $\boldsymbol{x}$, $\boldsymbol{x'}$ at random is insufficient because if Q queries $R(w \circ r)$ many times, it should receive a consistent response. Thus, the mapping from w and w' to $\boldsymbol{x}$ and $\boldsymbol{x'}$ must be deterministic. However, the mapping cannot be made in advance because Q does not know w or w' ahead of time. Instead, Q simply learns one of the accepted points "on the fly" and must be able to send the two random oracle queries to different outputs anyway. In essence, we are asking Q to make a function $F : \{0,1\}^n \to \{0,1\}$ such that $F(w) \neq F(w')$ without knowing w or w', which cannot be done.

On a positive note, this problem can be solved if the range of F is large enough. This is where the lossiness of the one-way function comes in handy: instead of having just two different outputs, we actually have 2^{3n} possible values of $\boldsymbol{x}$! Hence, we can map *every* string to a different output of the random oracle, and then only use this mapping for the two accepted points (whatever they happen to be). We use this idea in the proof.

Theorem 5.15. *In the random oracle model, $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is a tamper-proof obfuscator for $\mathcal{P}^2$. However, $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is not tamper-evident.*

Many of the ideas in the proof follow that in the single-point case, and the common arguments are only sketched here in order to focus on the new ideas, so the reader is encouraged to examine the proof of Theorem 5.11 before continuing.

Proof. First, we show the almost exact functionality of $\bar{\mathcal{O}}_{\mathcal{P}^2}$. It is clear that the construction accepts the points w, w' that should be accepted, but it may cause "false positives": inputs that are accepted but shouldn't be. Given the target element t, there are 2^{3n} vectors $\boldsymbol{y}$ such that $f_{p,G,g_1,g_2}(\boldsymbol{y}) = t$. For each such vector, there exists a unique $\boldsymbol{x}$ such that $\boldsymbol{\xi}(\boldsymbol{x}) = \boldsymbol{y}$. Furthermore, the string $R^{-1}(\boldsymbol{x})$ ends in r with probability 2^{-5n} over the choice of the random oracle R. By a union bound, a false positive occurs with probability at most 2^{-2n}, which is negligible.

As always, the polynomial slowdown property is clear. It remains to prove tamper-proofing. Let A be an adversary, and we construct a simulator S for A. Given an input length n, auxiliary input z, and access to an oracle for a two-point circuit $I_{\{w_1,w_2\}}$, the simulator S constructs a "fake" obfuscated circuit by choosing $r \xleftarrow{U} \{0,1\}^{5n}$, $(p, G, g) \leftarrow \mathbb{G}(1^{3n})$, $\boldsymbol{a}, \boldsymbol{v} \xleftarrow{U} \mathbb{F}_p^2$, and two lines ξ_1, ξ_2 uniformly over $\mathbb{F}_p^2$. Then, it computes $g_1 = g^{a_1}$, $g_2 = g^{a_2}$, and $t = g_1^{v_1} g_2^{v_2}$. Finally, it outputs the circuit $\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}$. Note that with overwhelming probability, this circuit rejects all inputs.

Then, $S^{R,I_{\{w,w'\}}}$ emulates an execution of $A^R(\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}, z)$. When A makes an oracle query of the form $R(q)$, the simulator queries its oracle $I_{\{w,w'\}}$ on the first n bits of q.

1. If the oracle accepts, then S has found an accepted point (which we call w without loss of generality), so it aborts this emulation of the adversary, replaces the fake obfuscated circuit with a real obfuscation of the circuit $I_{\{w,w\}}$, and restarts the emulation. If the restarted emulation finds the second accepted point, then once again S aborts its emulation of the adversary, forms an obfuscation of the circuit $I_{\{w,w'\}}$, and emulates the adversary perfectly.

2. If $I_{\{w,w'\}}$ does not accept but $R(q)$ is a vector such that $f_{p,G,g_1,g_2}(\boldsymbol{\xi}(R(q))) = t$, then the adversary has detected that the obfuscation is fake. In this case, the simulator aborts.

3. Otherwise, S responds to the random oracle query honestly.

Note that S restarts the adversary at most twice. Furthermore, we claim that A only detects that the obfuscation is fake with negligible probability. This is an immediate consequence of one-wayness (if A has not yet found any accepted points) or second preimage resistance (if A has found one accepted point).[12]

Eventually, the emulation of A halts and outputs a circuit P. Then, S outputs the circuit Q that runs P and traps on its random oracle queries $R(q)$ in a slightly different way. Again, we start by passing the first n bits of q (which we denote by u) to the oracle $I_{\{w,w'\}}$

1. If $I_{\{w,w'\}}$ accepts, then Q has found an accepted point u. Then, Q returns $\boldsymbol{\xi}^{-1}(\boldsymbol{v} + u\bar{\boldsymbol{a}})$ as the response of the random oracle. Note that $\boldsymbol{\xi}$ consists of lines that are not horizontal with overwhelming probability, so it can easily be inverted.

2. Otherwise, Q responds to the random oracle query honestly.

[12]Note that if the adversary queries both accepted points, then it can learn $\bar{\boldsymbol{a}}$ and thus form exponentially-many inverses to t. That is, the function f is not third preimage resistant. However, in this case, the adversary has learned everything about the obfuscated circuit that we wished to hide anyway, so learning more about f is irrelevant.

In particular, note that the behavior of Q is deterministic, given that the behavior of P is so.

This concludes the explanation of the simulator S and its output circuit Q. Now we analyze these circuits and prove that they satisfy the tamper-proofing property for the adversary A. From now on, we assume that the final obfuscated circuit given to the adversary has no "false positive" points that are accepted but shouldn't be. As argued above, this holds with overwhelming probability. Under this assumption, we note that the simulation is trivially perfect in two cases: if the input circuit is $I_\emptyset$, or if the simulator S finds all of the accepted points in its emulation of the adversary.

Next, suppose that $w \neq w'$, and the simulator S found one accepted point w but not the other one w'. We claim that $\boldsymbol{y'} = \boldsymbol{\xi}(R(w' \circ r))$ still has the uniform distribution over all preimages of t, even to an adversary that knows f_{p,G,g_1,g_2}, t, w, w', and $\boldsymbol{y}$. To show this, we exploit the "redundancy" of $\boldsymbol{a}$; specifically, we use the fact that any multiple $c\boldsymbol{a}$ of $\boldsymbol{a}$ is consistent with the view of the adversary. Specifically, let $\boldsymbol{a'} = c\boldsymbol{a}$ for $c \in \mathbb{F}_p$ and $g' = g^{(c^{-1})}$. Then, $g'_1 = (g')^{a'_1} = (g^{c^{-1}})^{ca_1} = g_1$, and similarly $g'_2 = g_2$. Hence, if we choose $c \leftarrow \mathbb{F}_p$, sample $(p, G, g) \leftarrow \mathbb{G}(1^{3n})$, and set $g' = g^{(c^{-1})}$ and $\boldsymbol{a'} = c\boldsymbol{a}$, the adversary's view is unchanged because $f_{p,G,g'_1,g'_2} \equiv f_{p,G,g_1,g_2}$, and g is chosen uniformly from G so the sampling algorithm could've picked g' just as easily as g. Thus, from the adversary's point of view, $\boldsymbol{y'}$ is chosen from the distribution

$$\{\boldsymbol{y} + (w' - w)c\bar{\boldsymbol{a}} : c \leftarrow \mathbb{F}_p\},$$

which is easily seen to be uniform over all preimages of t (even for a fixed value of $w' - w$), as desired. Hence, knowledge of $\boldsymbol{x}$ and $\boldsymbol{y}$ provides no help in learning anything about w', $\boldsymbol{x'}$, or $\boldsymbol{y'}$, and as a result the proof techniques from Theorem 5.11 in the one-point case also work here.

The novel case to prove is the one in which there are two distinct accepted points $w \neq w'$ but the emulated adversary does not query either of them. In this case, we perform the following "mental experiment": suppose we replace the random oracle R with a fake oracle R' that differs only on the two values

$$R'(w \circ r) = \boldsymbol{\xi}^{-1}(\boldsymbol{v} + w \cdot \bar{\boldsymbol{a}}), \qquad R'(w' \circ r) = \boldsymbol{\xi}^{-1}(\boldsymbol{v} + w' \cdot \bar{\boldsymbol{a}}).$$

Then, R' is distributed as a uniformly random function, subject to the constraints that w, w' are accepted by the simulator's "fake" circuit when using oracle R'. This follows from the argument above that these two outputs are (jointly) uniformly random preimages of t over the choice of $\boldsymbol{v}$ and $\boldsymbol{a}$, and the fact that R' agrees with the random function R on all other inputs.

By assumption, the emulated adversary does not distinguish R from R', so its output circuit P is the same in either case. Furthermore, the random oracle query changes made by Q are precisely those that make $Q^R \equiv P^{R'}$. Finally, the relation E cannot distinguish between the real and fake random oracles by the same information-theoretic argument as before: its input has $4n$ bits of entropy, but it needs to learn r (a string with $5n$ bits of entropy) in order to distinguish R from R'. Putting everything together, we have shown that if the adversary does not query either w or w',

$$\begin{aligned}\Pr\Big[P \leftarrow A^{R'}(\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}, z) : \exists I_{\{w'',w'''\}} \equiv P^{R'} \text{ and } E^{R'}(I_{\{w,w'\}}, I_{\{w'',w'''\}}) = 1\Big] \\ \approx \Pr\Big[Q \leftarrow S^R(\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}, z) : \exists I_{\{w'',w'''\}} \equiv Q^R \text{ s.t. } E^R(I_{\{w,w'\}}, I_{\{w'',w'''\}}) = 1\Big]\end{aligned}$$

so $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is a tamper-proof obfuscator, as desired (in this formula, we allow w'' or w''' to equal the empty string $\emptyset$ if fewer than two points are accepted).

Next, we prove that $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is malleable in the tamper-evident sense, meaning that for every verifier $\bar{V}_{\mathcal{P}^2}$, the pair $(\bar{\mathcal{O}}_{\mathcal{P}^2}, \bar{V}_{\mathcal{P}^2})$ is not tamper-evident. We show this in the case that the auxiliary

information $z = \emptyset$ is the empty string. Just as in Theorem 5.12, we form an adversary A that modifies an obfuscated circuit into a different one that performs the same functionality, so it breaks tamper-evidence for the relation $E(I_{\{w,w'\}}, I_{\{w'',w'''\}})$ that accepts if and only if $\{w, w'\} = \{w'', w'''\}$.

Given an obfuscated circuit of the form $\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}$, suppose ξ_1 and ξ_2 are formed by the equations $y = mx + b$ and $y = m'x + b'$, respectively. Then, A forms new lines ξ_1' and ξ_2' according to the equations $y = \frac{m}{2}x + \frac{b}{2}$ and $y = \frac{m'}{2}x + \frac{b'}{2}$; that is, the output of the lines in $\boldsymbol{\xi'}$ are half those of $\boldsymbol{\xi}$.[13] Next, A computes $g_1' = g_1^2$ and $g_2' = g_2^2$ and outputs the circuit $\Psi_{r,t,\boldsymbol{\xi'},p,G,g_1',g_2'}$. Note that

$$f_{p,G,g_1',g_2'}(\boldsymbol{\xi'}(\boldsymbol{x})) = (g_1')^{\xi_1'(x_1)} \cdot (g_2')^{\xi_2'(x_2)} = g_1^{2\xi_1'(x_1)} \cdot g_2^{2\xi_2'(x_2)} = g_1^{\xi_1(x_1)} g_2^{\xi_2(x_2)} = f_{p,G,g_1,g_2}(\boldsymbol{\xi}(\boldsymbol{x})),$$

so the circuits $\Psi_{r,t,\boldsymbol{\xi},p,G,g_1,g_2}$ and $\Psi_{r,t,\boldsymbol{\xi'},p,G,g_1',g_2'}$ have the same functionality. Furthermore, the circuit $\Psi_{r,t,\boldsymbol{\xi'},p,G,g_1',g_2'}$ could have been generated by the obfuscator, so any verifier $\bar{V}_{\mathcal{P}^2}$ for $\bar{\mathcal{O}}_{\mathcal{P}^2}$ must accept it. Thus, for tamper-evidence to hold, there must exist a PPT simulator $S^{I_{\{w,w'\}}}$ that can output a circuit that is equivalent in functionality to $I_{\{w,w'\}}$ for any $w, w' \in \{0,1\}^n$, but this is clearly impossible so $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is not tamper-evident as desired. □

General m

The construction in $\bar{\mathcal{O}}_{\mathcal{P}^2}$ can be generalized to the m-point case for any constant m. Rather than describing a new obfuscator in detail, here we focus on the necessary changes from $\bar{\mathcal{O}}_{\mathcal{P}^2}$.

The main issue is with the one-way function. We used the function f_{p,G,g_1,g_2} above because it had second preimage resistance, and in fact satisfied it in a strong way such that we can find a collision pair $\boldsymbol{y}$ and $\boldsymbol{y'}$ such that $\boldsymbol{y'}$ has the uniform distribution over all collisions even given $\boldsymbol{y}$.

In the general case, we need a family $\mathcal{T}$ of m-to-1 one-way functions with a special property that we call "m^{th} preimage uninvertibility," whereby it is possible to choose a function $f \leftarrow \mathcal{T}$ and inputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}$ such that $f(\boldsymbol{y_1}) = \cdots = f(\boldsymbol{y_m})$, and moreover $\boldsymbol{y_m}$ is uniformly distributed over the set of preimages $f^{-1}(\boldsymbol{y_1})$ even given a description of f and the preimages $f(\boldsymbol{y_1}), \ldots, f(\boldsymbol{y_{m-1}})$.

One such function is the generalized chameleon hash[14]

$$f_{p,G,g_1,\ldots,g_m}(y_1, \ldots, y_m) = g_1^{y_1} \times \cdots \times g_m^{y_m},$$

where each group element g_i is chosen by taking a common generator g to the a_i power. Hence, the function computes $f(\boldsymbol{y}) = g^{\langle \boldsymbol{a}, \boldsymbol{y} \rangle}$, just as before. Note that the domain of f is now $\mathbb{F}_p^m$, and the range is still $\mathbb{F}_p$, which means that for a given target value $t = f(\boldsymbol{y})$, there is a dimension $m-1$ subspace of inputs $\boldsymbol{y'}$ such that $f(\boldsymbol{y'}) = t$. Furthermore, we can form a circuit C that enumerates over all of these preimages by choosing a basis $\boldsymbol{b_1}, \ldots, \boldsymbol{b_{m-1}}$ of vectors orthogonal to $\boldsymbol{a}$ and setting

$$C(i_1, \ldots, i_{m-1}) = \boldsymbol{y} + i_1 \boldsymbol{b_1} + \cdots i_{m-1} \boldsymbol{b_{m-1}}.$$

Note that the input to C is $(m-1)n$ bits long. This presents a technical problem, as we cannot simply index C using the accepted points like we did before with w and w'. Instead, we must associate accepted points $w \in \{0,1\}^n$ with "random" indices $i_1, \ldots, i_{m-1} \in \{0,1\}^{(m-1)n}$ in such a way that all of the indices depend on the choice of w. Luckily we have a random oracle around to solve this problem.

Essentially, the function $f_{p,G,g_1,\ldots,g_m}$ satisfies m^{th} preimage uninvertibility because if $\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}$ are chosen uniformly at random from the preimages of t, then they are linearly independent with

[13] Remember that $\mathbb{F}_p$ is a field, so we can divide by 2 (or in other words, multiply by $2^{-1} \bmod p$).

[14] We thank Nir Bitansky for pointing this out.

overwhelming probability. This can be seen iteratively, as each $\boldsymbol{y_i}$ has probability $\frac{1}{p}$ of being in the span of the previous vectors, and m is a constant. Now, consider an adversary with knowledge of the vectors $\boldsymbol{y_1}, \ldots, \boldsymbol{y_{m-1}}$. By essentially the same argument as used in Theorem 5.14, the adversary cannot find a linearly independent preimage[15] without breaking the discrete logarithm assumption. Hence, the adversary cannot find $\boldsymbol{y_m}$, so m^{th} preimage uninvertibility holds.

There are a few additional modifications to the construction in the $m = 2$ case. Note that the domains of the $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are $\mathbb{F}_p^m$ now, and there are m such vectors, so the lines ξ_1, ξ_2 in the above construction must be replaced by m polynomials of degree $m-1$ over $\mathbb{F}_p$. In our proof, the simulator needs to invert the ξ_i, so S has to find the roots of polynomials over a finite field, which can be done in polynomial time [90].

Finally, we need to increase the length of the string r. In order to maintain the information-theoretic argument about the relation E, we need $|r| \geq (2m+1)n$, and in order to maintain almost exact functionality, we need $|r| \geq (3n-2)n$. For simplicity, we choose $|r| = (3n-1)n$, so the prime p can remain $3n$ bits long and it is easy to embed strings of length $|w \circ r| = 3mn$ into $\mathbb{F}_p^m$.

Using the same arguments as in the proof of Theorem 5.15, it can be proved that these modifications yield a tamper-proof obfuscator for m-point circuits. We omit further details.

5.4 Constructions of tamper-evident obfuscators

In this section, we present tamper-evident obfuscators for the family of multi-point circuits in the random oracle and common reference string models.

5.4.1 Random oracle model

In the single-point case, the obfuscator $\mathcal{O}_{\mathcal{P}^1}$ from Algorithm 5.1 is also tamper-evident. Let $V_{\mathcal{P}^1}$ be the verification algorithm that accepts if and only if its input is a circuit of the form $\Phi_{r,t}$ for some r and t such that $|t| = \frac{4}{3}|r|$. It is clear from Algorithm 5.1 that $V_{\mathcal{P}^1}$ always accepts proper obfuscations of point circuits.

Theorem 5.16. *In the random oracle model, $(\mathcal{O}_{\mathcal{P}^1}, V_{\mathcal{P}^1})$ is a tamper-evident obfuscator for $\mathcal{P}^1$.*

Proof. The functionality and polynomial slowdown properties were proved in Theorem 5.11. Also, it is clear that $V_{\mathcal{P}^1}$ only accepts circuits that are equivalent to some member of $\mathcal{P}^1$, so the verification property holds. It remains to prove tamper-evidence. Let A be an adversary, and we construct the simulator S for A as follows. When given auxiliary information z and oracle access to the point circuit I_w, the simulator $S^{R,I_w}(1^n, z)$ forms a fake obfuscation by choosing $r \xleftarrow{U} \{0,1\}^{3n}$ and $t \xleftarrow{U} \{0,1\}^{4n}$. Then, S emulates an execution of $A(\Phi_{r,t}, z)$ while monitoring its random oracle calls. There are three cases to consider:

1. If A queries R on the input $w \circ r$, then the adversary A found the hidden point w, but now the simulator has too. As a result, the simulator can simply form an obfuscation of I_w and run A on it. In this case, the simulation is perfect.

2. If A queries the value $s = R^{-1}(t)$, then A has discovered that our obfuscation is fake. In this case, the simulator returns an abort symbol $\perp$.

[15] Note that the adversary can find many preimages of f by choosing any vector in the span of $\boldsymbol{y_1}, \ldots, \boldsymbol{y_{m-1}}$. Thus, the function violates m^{th} preimage resistance, and in fact it is not even third preimage-resistant, just as the function used before was not. However, it is uninvertible because the adversary cannot find a specific preimage.

3. On any other query q, S allows A to access the random oracle and continue its execution.

At the end of its execution, the emulated adversary outputs a circuit P. Then, the simulator outputs the same circuit $Q = P$.

Because the second case only occurs with negligible probability, and the simulation becomes perfect if the input circuit is $I_\emptyset$ or if the first case ever occurs, it remains to prove the theorem in the scenario that the input circuit is not $I_\emptyset$ and only the third case occurs. In this scenario, the adversary A never queries the oracle on the hidden point w, nor does it discover that the obfuscation is a forgery.

Let E be any relation that only makes polynomially-many oracle queries, and define

$$\gamma_A = \Pr\left[P \leftarrow A^R(\mathcal{O}(I_w), z) : P \neq \mathcal{O}(C),\, V(P) = 1,\, \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^R,\, E^R(I_w, I_{w'}) = 1\right]$$
$$\gamma_S = \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^{R,I_w} \text{ and } E^R(I_w, I_{w'}) = 1\right]$$

as the probabilities of success for the real adversary and simulator, respectively. We will show that $|\gamma_A - \gamma_S|$ is negligible in n. To do so, we perform the following "mental experiment": let R' be a different random oracle permutation in which

$$R'(w \circ r) = t, \qquad\qquad R'(s) = R(w \circ r),$$

and R' agrees with R on all other values (see Figure 5-1 for a diagram). If R' really were the random oracle, then the "fake" obfuscation $\Phi_{r,t}$ would actually be a valid obfuscation of I_w, so

$$\gamma_A = \Pr\left[P \leftarrow A^{R'}(\Phi_{r,t}, z) : P \neq \Phi_{r,t},\, V(P) = 1, \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^{R'},\, E^{R'}(I_w, I_{w'}) = 1\right].$$

Furthermore, by assumption A never queries the oracle at the two locations in which R and R' differ. Hence, A must act the same in both cases, so the circuit Q returned by the simulator after its emulation of A^R on the "fake" obfuscation is identical to the circuit P returned by $A^{R'}$ when given a real obfuscation. Therefore,

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : Q \neq \Phi_{r,t},\, V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^{R'},\, E^{R'}(I_w, I_{w'}) = 1\right],$$

where now the simulator itself chooses r and t (rather than before, where they were provided to the adversary).

Next, we claim that if $Q \neq \Phi_{r,t}$ $V(Q) = 1$, and $Q^{R'}$ is a point circuit, then Q^R and $Q^{R'}$ have the same functionality. The verification test ensures that Q has the form of a obfuscated circuit $\Phi_{r',t'}$ for some r' and t'. Thus, Q accepts the single point x such that $x \circ r'$, when applied to its oracle, yields t' (if such a point x exists), and the functionality of Q depends solely on the value of the random oracle at $x \circ r'$.

Recall that R and R' differ in only two input values, namely s and $w \circ r$. As a result, Q^R and $Q^{R'}$ have different functionalities if and only if $x \circ r' = s$ or $x \circ r' = w \circ r$.

1. The first case is easy to dismiss: it is infeasible for the simulator to find s, so it cannot output the final $3n$ bits of s as the value of r'.

2. In the second case, $x = w$ and $r' = r$. Because $\Phi_{r',t'} \neq \Phi_{r,t}$ but $r' = r$, it must be the case that $t' \neq t$. As a result, the circuit $Q^{R'}$ does not accept any points because $R'(w \circ r) \neq t'$. On the other hand, it can only be possible that $R(w \circ r) = t'$ with negligible probability, because the simulator never queries $w \circ r$ so it has only negligible probability of guessing its target t'. Hence, Q^R rejects all inputs with overwhelming probability as well.

As a result, Q^R and $Q^{R'}$ have different functionality with only negligible probability, so it follows that

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Finally, it remains to show that the relation E behaves similarly when given random oracle R or R'. This is an information-theoretic argument that proceeds in exactly the same manner as in the proof of Theorem 5.11.

Let R'' be another random oracle in which $R''(w \circ r) = t$, and R'' agrees with R on all other values. (See Figure 5-1 for a diagram.) In other words, R'' and R' differ only on the input s. Because the adversary, simulator, and relation E all run in polynomial time, they only query the value s with negligible probability, and therefore the difference in functionality between $E^{R'}$ and $E^{R''}$ is negligible. Furthermore, R and R'' only differ on input $w \circ r$, which has at least $3n$ bits of entropy, but the relation E only receives $2n$ bits of information as input (namely, the values w and w'). As a result, using any list encoding scheme, the adversary and simulator only have negligible probability of transmitting the value r to the relation E. If the relation does not query the value $w \circ r$, then the behavior of E^R and $E^{R''}$ is identical. As a result,

$$\gamma_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R}(I_w, I_{w'}) = 1\right] = \gamma_S,$$

as desired.

Hence, $(\mathcal{O}_{\mathcal{P}^1}, V_{\mathcal{P}^1})$ is a tamper-evident obfuscator for $\mathcal{P}^1$, and in fact the definition is achieved in a strong sense in which the simulator S is independent of the choice of the polynomial ρ. □

Multi-point circuits

In the multi-point setting, we can concatenate m copies of $\mathcal{O}_{\mathcal{P}^1}$ and "glue" them together using a self-signing technique[16] in order to construct the obfuscator $\mathcal{O}_{\mathcal{P}^m}$ described in Algorithm 5.5.

Algorithm 5.5 Obfuscator $\mathcal{O}_{\mathcal{P}^m}$ for the family of m-point circuits

Input: a circuit of the form $I_{\{w_1,\ldots,w_m\}}$ or $I_\emptyset$

1: let k be the number of distinct accepted points for the input circuit (possibly 0)
2: extract the k distinct points $w_1, \ldots, w_k$
3: choose randomness $r_1, \ldots, r_m \xleftarrow{U} \{0,1\}^{3mn}$
4: choose a key pair (s, v) for a one-time signature scheme with security parameter n
5: **for** $i = 1$ to k **do**
6: set $t_i = R(w_i \circ r_i \circ v)$
7: **end for**
8: **for** $i = k + 1$ to m **do**
9: choose $t_i \xleftarrow{U} \{0,1\}^{n+3mn+|v|}$
10: **end for**
11: choose a random permutation π on m elements, and permute the r_i and t_i by π
12: compute the signature $\sigma = \text{sign}_s(t_1, \ldots, t_m)$

Output: the circuit $\Gamma_{r_1,\ldots,r_m,t_1,\ldots,t_m,v,\sigma}$ that stores the r_i, t_i, v, and σ in a clearly identifiable manner, and on input x does the following: "for i from 1 to m, accept if $R(x \circ r_i \circ v) = t_i$"

[16] A signature scheme can be constructed from any one-way function [61] so in particular it can be constructed from the random oracle.

The associated verification algorithm $V_{\mathcal{P}^m}$ checks that its input circuit has the proper structure and validates the signature. The self-signing technique ensures that an adversary will be detected if she tries to re-use any of the $\mathcal{O}_{\mathcal{P}^1}$ obfuscations given to her, because she will not be able to forge the required signature.

To illustrate, let us return to the example from Section 1.2.4 in which Alice attacks an obfuscated three-point circuit that is constructed by concatenation. Alice can keep the pieces of the obfuscated circuit that accept Charles and herself but remove the part that accepts Bob. Thus, the construction is not tamper-proof. However, due to the unforgeability of the signature scheme, Alice cannot perform this attack in such a way that the verifier accepts her modified program, so the construction is tamper-evident.

Theorem 5.17. *In the random oracle model, $(\mathcal{O}_{\mathcal{P}^m}, V_{\mathcal{P}^m})$ is a tamper-evident obfuscator for $\mathcal{P}^m$. However, $\mathcal{O}_{\mathcal{P}^m}$ is not tamper-proof.*

Proof. It is easy to see that $\mathcal{O}_{\mathcal{P}^m}$ satisfies almost exact functionality and polynomial slowdown, and that $V_{\mathcal{P}^m}$ has the desired verification properties. We note that, as before with $\mathcal{O}_{\mathcal{P}^1}$, the obfuscator actually has exact functionality on all m-point circuits, and only errs with negligible probability when given a circuit with fewer than m accepted points.

To prove tamper-evidence, let A be an adversary and we construct a simulator S that is given auxiliary information z and oracle access to a circuit of the form $I_{\{w_1,\ldots,w_m\}}$. Let T be the set of accepted points found by the simulator, so at the start of the algorithm $T = \emptyset$. The simulator $S^{R,I_{\{w_1,\ldots,w_m\}}}(1^n, z)$ creates a random obfuscation of the circuit I_T. This produces a circuit of the form $\Gamma_{r_1,\ldots,r_m,t_1,\ldots,t_m,v,\sigma}$ along with the intermediate values k and π used in the obfuscation. Then, the simulator emulates an execution of $A^R(\Gamma_{r_1,\ldots,r_m,t_1,\ldots,t_m,v,\sigma}, z)$. Whenever A makes a query q to the random oracle R, S acts as follows:

1. If the query q is $n + 3mn + |v|$ bits long, then let x denote the first n bits of q. The simulator queries its oracle to find the value $I_{\{w_1,\ldots,w_m\}}(x)$. If the oracle returns 1, then the simulator has found an accepted point. The simulator aborts the current emulation of A, adds x to the set T, forms an obfuscation of the new I_T, and starts an emulation of A on the new obfuscated circuit.

2. If $R(q)$ equals any of the t_i but the substring consisting of the first n bits of q is not in T, then the simulator fails. However, these t_i are chosen uniformly at random and independently of A or z, so the adversary can only find the inverse of t_i under the random oracle with negligible probability.

3. Otherwise, S allows A to access the random oracle and continue its execution.

Note that the simulator S either responds to A's oracle queries honestly or aborts the emulation of A. Additionally, case 2 can happen at most m times, so the simulator will complete the emulation of A in polynomial time. Let P denote the circuit that the emulation of A outputs. Then, S outputs the same circuit P.

It remains to analyze the probability of success for this simulator. Although our emulation of A was on a "fake" obfuscation that only includes some of the accepted points, note that there is a random oracle on which the obfuscation is correct. In particular, let R' be a random oracle defined as follows: let $T' = \{w_1, \ldots, w_m\} \setminus T$ be the set of all accepted points that the simulator did not find, and let $J = \{1, \ldots, m\} \setminus \{\pi(1), \ldots, \pi(k)\}$ be the set of all indices i that do not correspond to r_i and t_i that are "meaningful" in the final obfuscation used by the simulator. For all $w \in T'$,

choose a distinct index $j \in J$ and set

$$R'(w \circ r_j \circ v) = t_j, \qquad R'(R^{-1}(t_j)) = R(w \circ r_j \circ v).$$

Finally, set R' equal to R on all other inputs. Note that if the "fake" obfuscation produced by the simulator were run with oracle R' instead of R, then it would accept all of the points in T' so it would in fact be a valid obfuscation of $I_{\{w_1,\ldots,w_m\}}$. Hence, the simulation would be perfect. Additionally, S never queries the locations on which R and R' differ, so the simulator is indifferent to the random oracle used. In particular, S^R and $S^{R'}$ output the same circuit P.

Next, we show that the functionality of P is also indifferent to the random oracle used. For P to pass the verification test, it must have the form $\Gamma_{r'_1,\ldots,r'_m,t'_1,\ldots,t'_m,v',\sigma'}$ and the signature test must pass. Because the adversary can only forge signatures with negligible probability, it follows that $v \neq v'$ with overwhelming probability. In this case, the differences between the random oracles R and R' are meaningless for the execution of P, so $P^R \equiv P^{R'}$.

Finally, we show that the relation E is indifferent to the random oracle used. The relation E receives $2mn$ bits of information, but the randomness values $r_1, \ldots, r_m$ each have $3mn$ bits of entropy. As a result, using any list encoding scheme, the adversary and simulator have only negligible probability of transmitting any of the r_i to the relation E. Without any of the r_i, the relation cannot distinguish between R and R'. Hence, the probability of success for the simulator changes only by a negligible amount when we substitute E^R for $E^{R'}$.

In conclusion, the simulation of the adversary would be perfect if the random oracle R' were used, and the behavior of S, P, and E do not depend on whether the real oracle R or the imaginary oracle R' is used. As a result, the simulation succeeds even with the random oracle R, so $\mathcal{O}_{\mathcal{P}^m}$ is a tamper-evident obfuscator for $\mathcal{P}^m$, as desired.

Next, we show that $\mathcal{O}_{\mathcal{P}^m}$ is not tamper-proof. We do so in the case that the auxiliary input $z = \emptyset$ is the empty string. Let A be the adversary that when given $\Gamma_{r_1,\ldots,r_m,t_1,\ldots,t_m,v,\sigma}$ as input, chooses a random value $w^* \xleftarrow{U} \{0,1\}^n$, sets $t_1^* = R(w^* \circ r_1 \circ v)$, and outputs the circuit $\Gamma_{r_1,\ldots,r_m,t_1^*,t_2,\ldots,t_m,v,\sigma}$. Note the following facts about A.

1. The signature σ is now incorrect, but in the tamper-proof setting this is okay because nobody checks the signature.

2. If the input $\Gamma_{r_1,\ldots,r_m,t_1,\ldots,t_m,v,\sigma}$ was the obfuscation of the m-point circuit $I_{\{w_1,\ldots,w_m\}}$, then the output of A accepts $m-1$ of the points in $\{w_1, \ldots, w_m\}$. However, one of the accepted values is removed and replaced by a random point w^*.

Let E be the polynomial time computable relation such that

$$E(I_{\{w_1,\ldots,w_m\}}, I_{\{w'_1,\ldots,w'_m\}}) = 1 \iff \{w_1, \ldots, w_m\} \cap \{w'_1, \ldots, w'_m\} = m - 1.$$

When the adversary is given a circuit that accepts exactly m points, it outputs a circuit that passes this relation with overwhelming probability: the only way that A fails is by choosing $t_1^* = t_1$ so its output circuit is equal to its input circuit, but t_1^* is chosen randomly so this only occurs with negligible probability. However, any simulator $S^{R_1,\ldots,R_m,I_{\{w_1,\ldots,w_m\}}}(1^n)$ only has oracle access to $I_{\{w_1,\ldots,w_m\}}$, so the only hope that the simulator has to create a circuit with the functionality of $I_{\{w^*,w_2,\ldots,w_m\}}$ is to guess points in the hope that it finds an accepted point w_i, which it can then replace. Therefore, for every simulator, there exist many circuits $I_{\{w_1,\ldots,w_m\}}$ such that the simulator only succeeds with negligible probability, so the adversary A breaks the tamper-proofing property as desired. □

The self-signing concept used in the above construction can also be applied to the tamper-proof obfuscator for the family of multi-point circuits described in Section 5.3.2, yielding an obfuscator that simultaneously satisfies both forms of non-malleability.

5.4.2 Common reference string model

In the common reference string (CRS) model, we provide tamper-evident obfuscators for the family $\mathcal{P}^{m+}$ of multi-point circuits that does not include $I_\emptyset$. This family is slightly different than the one used in the random oracle constructions, because the constructions in this section have the property that it is easy to tell that obfuscated circuits accept at least one input, which was not the case in the random oracle constructions.

Also, in this section we can only prove a slightly weaker form of tamper-evidence. We first present an obfuscator in the single-point setting, where we believe the weaker non-malleability property is meaningful. We can generalize the obfuscator to operate over multi-point circuits, but the weaker form of non-malleability is insufficient in this setting.

Single-point circuits

Our construction uses two building blocks. First, let $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ be any obfuscator for $\mathcal{P}^{1+}$ without auxiliary input that has a perfect verifier $\hat{V}_{\mathcal{P}^{1+}}$, such as the (g, g^x) construction of [23] described in Section 3.2.1.

Second, let Π be a non-malleable non-interactive zero-knowledge (NIZK) proof of knowledge system [83, 84]. This is a rather long and technical term, so we break it up into pieces.

- A *zero-knowledge proof system* allows a user Alice to prove to Bob that she knows that a statement is true without revealing any additional information.
- Such systems have a *proof of knowledge* property if Alice must possess a witness that the statement is true. Formally, there exists an extractor that determines the witness.
- *Non-malleability* means that the proof of knowledge property must hold even if Alice views proofs of other statements.
- *Non-interactivity* means that Alice only sends one message to Bob.

Thus, the proof system Π has the property that if an adversary views a proof π and then produces a different proof π', then there exists an extractor that extracts the witness to the proof of π'.

Using these building blocks, we form the obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)$ that outputs $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)$ along with a proof that it knows the point w. The verification algorithm $\tilde{V}_{\mathcal{P}^{1+}}$ associated to $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ runs $\hat{V}_{\mathcal{P}^{1+}}$ and the verification algorithm of the proof system Π to check the validity of the proof.[17]

More formally, we define an NP relation $R_{\hat{\mathcal{O}}_{\mathcal{P}^{1+}}}$ based on $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ as follows:

$$R_{\hat{\mathcal{O}}_{\mathcal{P}^{1+}}}(P, w) = 1 \text{ if and only if } \exists r \text{ s.t. } P = \hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w, r).$$

That is, the first input to the relation must be a valid output of the obfuscator $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ (which can be efficiently verified by $\hat{V}_{\mathcal{P}^{1+}}$), and the second input must be the unique point that is accepted by this circuit. The obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$, described in Algorithm 5.6, uses the NIZK Π on this NP relation [84].

[17] The verifier also needs to check that the common reference string has the desired form, in order to prevent the simulator from abusing its privilege to choose the CRS. Typically, we view Σ as being a uniformly chosen $l(n)$-bit string, so the verifier can check that it has the correct length.

Algorithm 5.6 Obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ for the family of point circuits in the CRS model

Input: a circuit of the form I_w and a common reference string Σ

1: produce the obfuscated circuit $\hat{I}_w = \hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)$

2: use Π to prove that the obfuscator knows the string w such that $R_{\hat{\mathcal{O}}_{\mathcal{P}^{1+}}}(\hat{I}_w, w) = 1$, and call the resulting proof π_w

Output: the circuit $\tilde{I}_w$ that is equal to $\hat{I}_w$ except that it also stores π_w in some clearly visible way

Intuitively, the non-malleability of the obfuscation follows from the non-malleability of the NIZK. Unfortunately, the proof turns out to be quite delicate, and we can only prove a weaker version of tamper-evidence.

Definition 5.18 (Weak tamper-evidence in the CRS model)**.** Let $\mathcal{C}$ be a family of circuits and $(\mathcal{O}, V)$ be a pair of PPT algorithms. We say that $(\mathcal{O}, V)$ is *weakly tamper-evident for relation E* if for every PPT adversary A and polynomial ρ, there exists a PPT simulator S such that for all sufficiently large n and for all circuits $C \in \mathcal{C}_n$,

$$\Big|\Pr\left[P \leftarrow A(\mathcal{O}(C), \Sigma) : P \neq \mathcal{O}(C),\, V(P, \Sigma) = 1, \text{ and } \exists D \in \mathcal{C}_n \text{ s.t. } D \equiv P,\, E(C, D) = 1\right] - \Pr\left[(Q, \Sigma) \leftarrow S^C(1^n) : V(Q, \Sigma) = 1,\, \exists D \in \mathcal{C}_n \text{ s.t. } D \equiv Q \text{ and } E(C, D) = 1\right]\Big| < \frac{1}{\rho(n)},$$

where the first probability is taken over the coin tosses of A and $\mathcal{O}$, along with the uniformly random choice of the common reference string Σ, and the second probability is taken over the coin tosses of S.

Note that this definition is weaker than 5.9 in two ways: the simulator S is allowed to depend on the relation E, and there is no auxiliary input in this definition. We can prove that our construction satisfies this weaker variant of non-malleability for many interesting relations E.

Definition 5.19. A bivariate relation E is *invertible* if there exists a polynomial time algorithm $\bar{E}$ such that for every y, $\bar{E}(y)$ returns a list of all x such that $E(x, y) = 1$.

In particular, because E is a polynomial time algorithm, it can only output a list that is polynomially long in length. Therefore, for every y, there can only be polynomially many x such that $E(x, y) = 1$.

Theorem 5.20. *Let E be an invertible relation. In the CRS model, $(\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}, \tilde{V}_{\mathcal{P}^{1+}})$ is a weakly tamper-evident obfuscator for E.*

To prove this theorem, we first show that $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ is an obfuscator. As usual, functionality and polynomial slowdown are clear, so it remains to prove the virtual black-box property. Note that we are only going to prove a weak form of tamper-evidence, and Theorem 5.6 does not apply for the weaker notion so we must prove the virtual black-box property directly.

Given an adversary $\tilde{A}$ in the common reference string model that receives a circuit from $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$, we construct an adversary $\hat{A}$ that receives a circuit from $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ as follows. The adversary $\hat{A}$ generates a CRS Σ along with a trapdoor that allows $\hat{A}$ to make false proofs, and then feeds $\tilde{A}$ its input circuit along with a fake proof that it knows the hidden acceptable point. When the emulation of $\tilde{A}$ halts and returns a bit, $\hat{A}$ outputs this bit too. It is easy to see that $\hat{A}$ successfully breaks the virtual black-box property if and only if $\tilde{A}$ does. By the virtual black-box property of $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$, there exists a simulator $\hat{S}$ corresponding to $\hat{A}$. It follows that $\hat{S}$ also satisfies the virtual black-box property for $\tilde{A}$.

It is also clear by construction that $\tilde{V}_{\mathcal{P}^{1+}}$ is a perfect verifier for $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$. Now we show that $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ satisfies the weaker tamper-evidence property for invertible relations E. The proof is inspired by [23] and proceeds in two stages. First, we describe an intermediate property and show that it implies weak tamper-evidence. Second, we prove the intermediate property using specific facts about $\mathcal{P}^{1+}$, $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$, Π, and the invertibility property of E.

Definition 5.21. Let $\mathcal{C}$ be a circuit family and $(\mathcal{O}, V)$ be a verifiable obfuscator for $\mathcal{C}$. The obfuscator is *almost everywhere non-malleable for relation E* if for every PPT adversary A and polynomial ρ, there exists a polynomial size family of sets $\{L_n\}_{n\in\mathbb{N}}$ (where $L_n \subseteq \mathcal{C}_n$) such that for sufficiently large n and for all $C \in \mathcal{C}_n \setminus L_n$,

$$\Pr\left[P \leftarrow A(\mathcal{O}(C)) : V(P) = 1 \text{ and } \exists D \in \mathcal{C}_n \text{ s.t. } P \equiv D \text{ and } E(C, D) = 1\right] < \frac{1}{\rho(n)}.$$

That is, any adversary that tries to modify an obfuscated program in accordance with the relation E can only succeed on a polynomial size set of circuits.

Now we prove that this property implies weak tamper-evidence for many interesting circuit families, such as the family of multi-point circuits $\mathcal{P}^{m+}$ or point circuits with multi-bit output $\mathcal{I}$.

Lemma 5.22. *Let $\mathcal{C}$ be a circuit family and $(\mathcal{O}, V)$ be a verifiable obfuscator for $\mathcal{C}$. Suppose $\mathcal{C}$ has the property that given a circuit $C \in \mathcal{C}$ and oracle access to a circuit $D \in \mathcal{C}$, one can test in polynomial time whether $C \equiv D$. Then, almost everywhere non-malleability for relation E implies weak tamper-evidence for relation E.*

The testability condition in the lemma is satisfied by $\mathcal{P}^{1+}$, but it is not satisfied by $\mathcal{P}^1$ because it does not hold for $C = I_\emptyset$.

Proof. Let E be a relation and assume that $(\mathcal{O}, V)$ is almost everywhere non-malleable for relation E. Given a polynomial ρ and an adversary A, we construct the simulator S necessary to prove weak tamper-evidence for relation E.

By almost everywhere non-malleability, there is a polynomial size family of sets $\{L_n\}$ associated to A. By non-uniformity, we construct a simulator S that knows this family. The simulator $S^D(1^n)$ tests whether there exists $C \in L_n$ such that $C \equiv D$. This is feasible because L_n is a polynomial size set and each test can be done in polynomial time by the assumption of the lemma. If one of the tests succeeds, say when testing $C \in L_n$, then S emulates an execution of $A(\mathcal{O}(C))$. Otherwise, the simulator halts and outputs a failure symbol $\perp$.

It remains to analyze the performance of the simulator S. We consider two cases.

1. If S^D finds some $C \in L_n$ such that $C \equiv D$, then it emulates a real execution of A, so the simulation is perfect in this case.

2. Otherwise, then the simulator succeeds with probability zero. However, in this case the real adversary runs $A(\mathcal{O}(D))$ for some D such that $D \notin L_n$. Therefore, the almost everywhere non-malleability property says that the adversary succeeds with at most $\frac{1}{\rho(n)}$ probability.

Hence, in both cases the difference in success probabilities between the adversary and simulator is less than $\frac{1}{\rho(n)}$, as desired. □

Now we demonstrate that the obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ satisfies the almost everywhere non-malleability property. Whereas the above proof was generic in nature, now we focus on our specific construction. It is given that $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ is an obfuscator for the family of point circuits. As a result, [23] shows that $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ has the following property.

Definition 5.23 (Oracle indistinguishability[18])**.** For any polynomial ρ and any PPT distinguisher D that outputs a single bit, there exists a polynomial size family of sets $\{L_n\}_{n\in\mathbb{N}}$ such that for sufficiently large n and for all $I_w, I_{w'} \in \mathcal{P}^{1+}_n \setminus L_n$,

$$\left|\Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)) = 1\right] - \Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_{w'})) = 1\right]\right| < \frac{1}{\rho(n)}.$$

We use this property to demonstrate that almost everywhere non-malleability holds.

Lemma 5.24. *Let E be an invertible relation. Then, $(\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}, \tilde{V}_{\mathcal{P}^{1+}})$ is almost everywhere non-malleable for relation E.*

Proof. Let E be an invertible relation and $\bar{E}$ be the extraction algorithm associated to E. Assume for the sake of contradiction that $(\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}, \tilde{V}_{\mathcal{P}^{1+}})$ is *not* almost everywhere non-malleable for relation E, and we will show a distinguisher that breaks the oracle indistinguishability property of $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$. Since $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ is assumed to be an obfuscator for the family of point circuits, and [23] shows that all such obfuscators obey oracle indistinguishability, we establish a contradiction.

Suppose the adversary A breaks almost everywhere non-malleability for relation E. We use A to construct a distinguisher D. Note that A outputs long strings, but D is only allowed to output a single bit. Distinguisher D operates as follows.

on input a circuit P, form a CRS-trapdoor pair (Σ, τ)
use the trapdoor τ to create a false proof π claiming you know a witness w s.t. $R_{\hat{\mathcal{O}}_{\mathcal{P}^{1+}}}(P, w) = 1$
set P' to be the circuit that is equal to P except that it also stores π in some clearly visible way (the same way that the obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ does in Algorithm 5.6)
emulate $A(P')$ with common reference string Σ, and set Q to be the output of the emulation
if $\tilde{V}_{\mathcal{P}^{1+}}(Q) = 1$ **then**
  let π' be the proof embedded in Q (there is such a proof because the verifier checks for its existence)
  use the proof system's knowledge extractor to extract the witness w' in the proof π'
  set $S \leftarrow \bar{E}(w')$ to be the set of all x such that $E(x, w) = 1$
  for all $x \in S$ **do**
    if $P(x) = 1$ **then**
      output the first bit of x
    end if
  end for
end if
output a uniformly random bit

This distinguisher runs in polynomial time because A, $\tilde{V}_{\mathcal{P}^{1+}}$, $\bar{E}$, and the proof system's knowledge extractor are all PPT algorithms. Now we analyze the distinguisher's output when it is given $P = \hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)$ as input. We consider two cases:

1. If the adversary A succeeds in creating a circuit P' such that there exists w' where P' looks like a valid obfuscation of $I_{w'}$ and $E(I_w, I_{w'}) = 1$, then the distinguisher correctly finds w and outputs the first bit of w.

2. Otherwise, the distinguisher outputs a random bit.

[18] We have seen this property before (in Definition 4.9) as it applies to point circuits with multi-bit output. Here, we present the definition in the case of point circuits.

By assumption, A breaks almost everywhere non-malleability for relation E, so there exists a polynomial ρ such that for every family of polynomial size sets $\{L_n\}$, there exist infinitely many n and a circuit $I_w \in \mathcal{P}_n^{1+} \setminus L_n$ such that

$$\Pr\left[P' \leftarrow A(\tilde{\mathcal{O}}_{\mathcal{P}^1}(I_w)) : V(P') = 1 \text{ and } \exists I_{w'} \text{ s.t. } P' \equiv I_{w'} \text{ and } E(I_w, I_{w'}) = 1\right] > \frac{1}{\rho(n)}.$$

By our analysis, if the distinguisher is given an obfuscation of this special circuit I_w, case 1 occurs with probability at least $\frac{1}{\rho(n)}$, so the distinguisher correctly outputs the first bit of w with probability at least $\frac{1}{2} + \frac{\rho(n)}{2}$.

Alternatively, suppose the distinguisher is given the obfuscation of any circuit $I_v \in \mathcal{P}_n^{1+} \setminus L_n$ such that the first bit of v is different from the first bit of w. We no longer know how often case 1 occurs and how often case 2 occurs, but in either case the distinguisher will output the first bit of v with probability at least $\frac{1}{2}$. This is enough to complete the proof.

Set the polynomial $\rho' = \frac{\rho}{2}$, and we have shown the following: for every family of polynomial size sets $\{L_n\}$, there exist infinitely many n and some $I_w, I_v \in \mathcal{P}_n^{1+} \setminus L_n$ such that

$$\Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)) = \text{first bit of } w\right] \geq \frac{1}{2} + \rho'(n)$$

and

$$\Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_v)) = \text{first bit of } v\right] \geq \frac{1}{2}.$$

But the first bits of w and v are different, so it follows that

$$\left|\Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)) = 1\right] - \Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_v)) = 1\right]\right| \geq \frac{1}{\rho'(n)},$$

so the distinguisher D and polynomial ρ' break oracle indistinguishability. □

Finally, the composition of Lemmas 5.22 and 5.24 yield Theorem 5.20, as desired.

Multi-point circuits

The obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ can be generalized to the multi-point setting, as follows. Let $\tilde{\mathcal{O}}_{\mathcal{P}^{m+}}$ be the obfuscator that, when given $I_{\{w_1,\ldots,w_m\}}$ as input, outputs the concatenation of m single-point obfuscations $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_{w_1})$, …, $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_{w_m})$ followed by a non-malleable NIZK proof of knowledge that it knows all of the accepted points $w_1, \ldots, w_m$. As before, let $\tilde{V}_{\mathcal{P}^{m+}}$ be the verification algorithm that checks the structure of the program and the validity of the proof.

To show that $\tilde{\mathcal{O}}_{\mathcal{P}^{m+}}$ is an obfuscator, one can either assume that $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ is an m-composable obfuscator [25] or, for certain constructions of $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$, prove the composability of the obfuscator from a number-theoretic assumption like a strengthening of Assumption 3.20 [12].[19] We show that this obfuscator is weakly tamper-evident for invertible relations using a proof similar to that in Theorem 5.20.

Theorem 5.25. *Let E be an invertible relation. In the CRS model, $(\tilde{\mathcal{O}}_{\mathcal{P}^{m+}}, \tilde{V}_{\mathcal{P}^{m+}})$ is a weakly tamper-evident obfuscator for E.*

[19] See Sections 3.2.1 and 3.3.3 for details.

Proof. The conditions of Lemma 5.22 hold in the multi-point setting, so it suffices to prove that $(\tilde{\mathcal{O}}_{\mathcal{P}^{m+}}, \tilde{V}_{\mathcal{P}^{m+}})$ is almost everywhere non-malleable for E. Suppose for the sake of contradiction that the adversary A breaks almost everywhere non-malleability for E. Then, we construct a distinguisher that breaks the oracle indistinguishability property for $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$. The distinguisher D uses the proof system's knowledge extractor K and the extraction algorithm $\bar{E}$ associated with E. The distinguisher operates as follows, when it receives the obfuscation of a single-point circuit as input.

on input a circuit P, set n to be the input length of P
form a CRS-trapdoor pair (Σ, τ), and choose $w_1, \ldots, w_{m-1} \xleftarrow{U} \{0,1\}^n$
for $i = 1$ to $m-1$ **do**
  form the obfuscation $\hat{O}_{\mathcal{P}^{1+}}(I_{w_i})$ using fresh randomness
end for
let P' be the concatenation of P with the $m-1$ obfuscations $\hat{O}_{\mathcal{P}^{1+}}(I_{w_i})$ in a random order
create a false proof π claiming you know the m points that P' accepts, and append this proof to P'
emulate $A(P')$ with common reference string Σ, and set Q to be the output of the emulation of A
if $\tilde{V}_{\mathcal{P}^{m+}}(Q) = 1$ **then**
  set π' to be the proof embedded in Q
  run $K(\pi')$ to obtain the witness $w'_1, \ldots, w'_m$ consisting of the m accepted points in Q
  initialize S to be the empty set, and compute $\bar{E}(w'_1, \ldots, w'_m)$
  for all $I_{w_1,\ldots,w_m}$ such that $E(I_{w_1,\ldots,w_m}, I_{w'_1,\ldots,w'_m}) = 1$, as found by $\bar{E}$ **do**
    insert $w_1, \ldots, w_m \in S$
  end for
  for all $x \in S$ **do**
    if $P(x) = 1$ **then**
      output the first bit of x
    end if
  end for
end if
output a uniformly random bit

The adversary succeeds with non-negligible probability, so using the same analysis as in the proof of Lemma 5.24, it follows that the distinguisher $D(\hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_x))$ has a noticeable advantage in finding the first bit of x for super-polynomially many x, so D breaks oracle indistinguishability as desired. □

Unfortunately, in the multi-point setting, the set of invertible relations is too small. For example, the simple relation $E(I_{\{w_1,\ldots,w_m\}}, I_{\{w'_1,\ldots,w'_m\}})$ that accepts if any of the w_i equal any of the w'_j is not invertible. As a result, Theorem 5.25 is a promising result but still unsatisfactory. Future research is needed to find an obfuscator that is tamper-evident for a wider class of relations.

Chapter 6

Obfuscation of Hyperplane Membership

This chapter is based on joint work with Ran Canetti and Guy Rothblum [30].

6.1 Introduction

In the previous two chapters, we have connected program obfuscation to the problems of symmetric key encryption and tamper-proof hardware. These connections are elegant on a theoretical level, but the simple fact remains that we do not have many positive results in the field of program obfuscation. In order to expand our knowledge of program obfuscation, we have to find new constructions and proof techniques.

Under the Barak *et al.* definition, we can only construct obfuscators for very simple functionalities like login programs and its generalizations like multiple-user login programs or digital lockers (as described in Chapter 3). These programs have a simple structure: their behavior is identical on every possible input except for a polynomial-sized "exception list." The circuit is hardcoded with both the exception list and the behavior of the circuit when given an input from this list. This fact is exploited heavily in the proofs of their obfuscators.

The goal of this chapter is to construct an obfuscator for a new family of circuits that does not have this structure. Given a positive integer d and prime $p \approx 2^n$, let $\mathbb{F}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$ be the finite field of order p. Consider a hyperplane through the origin in the d-dimensional vector space $\mathbb{F}_p^d$, which can be identified by a vector $\boldsymbol{a}$ that is orthogonal to every point in the plane. Let $H_{\boldsymbol{a}}$ be a circuit that nonuniformly stores $\boldsymbol{a}$ in a readily identifiable manner and computes

$$H_{\boldsymbol{a}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We wish to obfuscate this family[1] of circuits for all hyperplanes $\boldsymbol{a}$. Note that $H_{\boldsymbol{a}}$ accepts and rejects exponentially many inputs (because p is exponential in the security parameter) so it cannot have the "exception list" structure. Instead, the accepted points can only be detected by an arithmetic computation, which an obfuscator for this family must hide. Due to these differences, obfuscating these programs is a challenging and novel goal.

[1] It turns out that this family is also a generalization of login programs, but in a different way than prior generalizations. In the $d = 2$ case, the "hyperplane testing programs" turn out to be equivalent to login programs, but for higher dimensions d, the new programs are more complicated.

Unfortunately, it is so challenging that we are not able to do it under any standard cryptographic assumptions, or even Canetti's strong DDH assumption (Assumption 3.20).[2] We need a generalization of Assumption 3.20 that considers more group elements and more operations that can be performed in the exponent (not just multiplication). Specifically, given a tuple of group elements $\langle g^{a_1}, g^{a_2}, \ldots, g^{a_d} \rangle$ from a group G, where the a_i are chosen from some joint distribution, our assumption limits the polynomials ξ for which $g^{\xi(a_1,\ldots,a_d)}$ can be distinguished from uniform. Of course, if ξ is linear, or even "close" to linear, then $g^{\xi(a_1,\ldots,a_d)}$ can easily be distinguished (and in fact computed). We want indistinguishability to hold in all other cases.

However, it is not clear how to define what it means for a polynomial to be "close" to linear. This issue is even more complex over the finite field $\mathbb{F}_p$ because very high degree polynomials can be close to linear ones. For instance, $x^p = x$ by Fermat's little theorem. Additionally, every function over $\mathbb{F}_p$ can be written as a polynomial, even slight modifications of linear functions. Identifying all of the ways that a polynomial can be "close" to linear is rather tedious. Instead, we we form a distributional assumption on G that is conceptually simple and provides the same guarantee. We describe our assumption in more detail in Section 6.3.

Although our assumption is stronger than Canetti's DDH assumption, we provide evidence of its feasibility by proving that it holds in the generic group model. Intuitively, this is not surprising because our assumption basically states that the only computations that can be performed in the exponent are linear ones. In other words, the assumption states that the group G is pseudo-free [67, 81] except that the order of the group is known.

Our construction itself is rather simple. Given a circuit $H_{\boldsymbol{a}}$, where $\boldsymbol{a} = (a_1, \ldots, a_d)$, the obfuscator chooses a random generator $g \xleftarrow{U} G$ and outputs g^{a_i} for all i. Then, a user can evaluate $H_{\boldsymbol{a}}(\boldsymbol{x})$ by testing whether

$$(g^{a_1})^{x_1} \times \cdots \times (g^{a_d})^{x_d} = g^{\langle \boldsymbol{a}, \boldsymbol{x} \rangle}$$

equals the identity element of G.[3]

Finally, we describe an application of our obfuscator to leakage-resilient one-time signature schemes. We use our obfuscator for hyperplane membership testing programs with dimension 3 to construct a signature scheme that satisfies a weak form of existential unforgeability, and use techniques from [56] to transform the weak scheme into an ordinary one-time signature scheme. The virtual black-box property immediately gives a leakage resilience guarantee, just as it did with encryption schemes in Chapter 4. Our scheme remains unforgeable even when a function of the secret key is leaked whose output length is up to half as long as the secret key, which matches the leakage bound of [58], albeit under much stronger assumptions.

Organization

In Section 6.2, we review the linear algebra that is required in this chapter. Section 6.3 describes our assumption in detail, comparing it to previous assumptions and showing that it holds in the generic group model. We present our obfuscator and prove its security in Section 6.4, and extend our construction to the multi-bit setting in Section 6.5. Finally, our one-time signature scheme is discussed in Section 6.6.

[2] We consider the strong DDH assumption often in this chapter, especially to compare it to our new assumption. We strongly encourage the reader to examine Section 3.2.1 for a formal treatment of this assumption.

[3] We note that in the work of Shen, Shi and Waters on private inner-product predicate encryption schemes [86], their construction also tests whether an inner product is 0 by running it in the exponent of a group where CDH is hard. Otherwise the settings, constructions, and assumptions are quite different. In particular, a user who wants to check whether a vector $\boldsymbol{x}$ has inner product 0 with a hidden vector $\boldsymbol{v}$ needs to first encrypt $\boldsymbol{v}$ using a secret key, so their predicate encryption scheme does not directly yield an obfuscation.

6.2 Finite vector spaces

In this section, we define the vector spaces over which our constructions operate. Let $d \in \mathbb{N}$ and p be a prime number. Then, $\mathbb{F}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$ is a finite field and $\mathbb{F}_p^d$ is a vector space over $\mathbb{F}_p$. We denote a vector in the vector space by $\boldsymbol{x} = (x_1, \ldots, x_d)$, where each $x_i \in \mathbb{F}_p$, and we have an inner product-style operation given by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{d} x_i y_i$.

Definition 6.1. Let $V \subseteq \mathbb{F}_p^d$ be a set.

1. Two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_p^d$ are *orthogonal* if their inner product is zero, so $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$. Note that the set of all vectors orthogonal to $\boldsymbol{x}$ forms a $(d-1)$ dimensional hyperplane.

2. The *closure* of V, written $\overline{V}$, is the subspace of all linear combinations of vectors in V.

3. The *orthogonal complement* of V, written $V^\perp$, is the subspace of all vectors that are orthogonal to every vector in V. That is,

$$V^\perp = \{\boldsymbol{x} \in \mathbb{F}_p^d : \langle \boldsymbol{x}, \boldsymbol{v} \rangle = 0 \ \forall \boldsymbol{v} \in V\}.$$

We caution that $\mathbb{F}_p^d$ does not satisfy all of the axioms of an inner product space, and many of our intuitions about vector spaces from $\mathbb{R}^d$ break down over finite fields. For instance, a non-zero vector can be orthogonal to itself! Hence, the concept of orthogonality loses its geometric meaning over finite fields. Nevertheless, the following theorem about inner product spaces, which we need in the proof of our main theorem, does hold over $\mathbb{F}_p^d$.

Theorem 6.2. *Let $V \subseteq \mathbb{F}_p^d$ be a set. Then, $(V^\perp)^\perp = \overline{V}$.*

Proof sketch. The theorem follows from three simple facts. First, $V^\perp$ and $V^{\perp\perp}$ are subspaces of $\mathbb{F}_p^d$ because the conditions imposed on them are linear. Second, $\bar{V} \subseteq V^{\perp\perp}$ because the vectors in $\overline{V}$ are orthogonal to those in $V^\perp$, so they are in $V^{\perp\perp}$. Third, $\dim(\overline{V}) = \dim(V^{\perp\perp})$ because both of them are equal to $d - \dim(V^\perp)$.

Therefore, $\overline{V}$ and $V^{\perp\perp}$ are subspaces of $\mathbb{F}_p^d$ of the same dimension such that one is included in the other, so they are equal. □

Note that the vector space $\mathbb{F}_p^d$ is a bit redundant for our needs. We wish to identify a hyperplane with a vector $\boldsymbol{a}$ that is orthogonal to every vector in the hyperplane. However, the vector $\boldsymbol{a}$ is not unique: indeed, for any $c \in \mathbb{F}_p \backslash 0$, the vector $c\,\boldsymbol{a}$ is also orthogonal to every vector in the hyperplane. Thus, the normal vector to the hyperplane is only unique up to scalar multiplication.

As a result, there are only $d-1$ degrees of freedom when choosing a normal vector, which is why the $d = 2$ case corresponds to point circuits. One canonical representation of the normal vector, which we will use when convenient throughout the paper, is to consider all of the vectors in $\mathbb{F}_p^d$ whose first non-zero coordinate equals 1.

In fact, from a mathematical point of view, the appropriate ambient space from which to consider the normal vectors is the projective $(d-1)$-dimensional space over $\mathbb{F}_p$, rather than $\mathbb{F}_p^d$ which has the unfortunate redundancy described above. However, from a computer science point of view, we need a concrete instantiation of the projective space that allows us describe vectors, which is why we use its embedding in $\mathbb{F}_p^d$.

Finally, we let $[k]$ denote the set of the first k natural numbers. That is, $[k] = \{1, 2, \ldots, k\}$.

6.3 Assumption

In this section, we define the main assumption, relate it to the strong DDH assumption (Assumption 3.20), and consider our assumption in the generic group model. The assumption uses groups of increasing prime order. We use the following definition to encapsulate the order requirement.

Definition 6.3. A function $\nu(n)$ is called a *prime sequence* if for every $n \in \mathbb{N}$, $\nu(n)$ is a prime number in the range $(2^{n-1}, 2^n]$.

Our assumption is parametrized by $d \in \mathbb{N}$. We abuse notation a bit and denote $\mathbb{F}_\nu^d = \{\mathbb{F}_{\nu(n)}^d\}_{n\in\mathbb{N}}$.

Assumption 6.4. *Given $d \in \mathbb{N}$, there exists a family of groups $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$ (written multiplicatively) such that the following three conditions hold:*

1. *The orders of the groups form a prime sequence $\nu(n) = |G_n|$.*
2. *There are efficient algorithms to perform the group and inverse operations, to test for equality with the identity element, and to sample uniformly from $\mathcal{G}$.*
3. *For every PPT adversary A and for all families of distributions $\mathcal{L} = \{\mathcal{L}_n\}_{n\in\mathbb{N}}$ and $\mathcal{R} = \{\mathcal{R}_n\}_{n\in\mathbb{N}}$ over $\mathbb{F}_\nu^d$, there exists a polynomial ξ such that for all n,*

$$\left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L}_n, g \overset{U}{\leftarrow} G_n : A(g^{l_1}, \ldots, g^{l_d}) = 1\right] - \Pr\left[\boldsymbol{r} \leftarrow \mathcal{R}_n, g \overset{U}{\leftarrow} G_n : A(g^{r_1}, \ldots, g^{r_d}) = 1\right]\right|$$
$$\leq \xi(n) \cdot \max_{\boldsymbol{x}\in\mathbb{F}_{\nu(n)}^d} \left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L}_n : \langle \boldsymbol{l}, \boldsymbol{x}\rangle = 0\right] - \Pr\left[\boldsymbol{r} \leftarrow \mathcal{R}_n : \langle \boldsymbol{r}, \boldsymbol{x}\rangle = 0\right]\right|. \quad (6.1)$$

In words, this assumption states that an adversary can distinguish two distributions of vectors if and only if orthogonality tests (i.e., choosing a vector and checking if it is orthogonal to the given one) can do so as well.

Discussion

We make several remarks.

1. The right-hand side of (6.1) depends on ν but not on any other property of $\mathcal{G}$.
2. The adversary is allowed to distinguish $\mathcal{L}$ and $\mathcal{R}$ better than any single orthogonality test does. For example, the adversary may try many orthogonality tests. The assumption merely states that the left-hand side of (6.1) is negligible whenever the right-hand side is.
3. Assumption 6.4 for dimension 1 trivially holds, even by groups that do not satisfy DDH.
4. Given an adversary A, let $A_{\boldsymbol{l}} = \Pr\left[g \overset{U}{\leftarrow} G_n : A(g^{l_1}, \ldots, g^{l_d}) = 1\right]$ and $A_{\mathcal{L}} = \mathbb{E}[\boldsymbol{l} \leftarrow \mathcal{L}_n : A_{\boldsymbol{l}}]$. We say that $\mathcal{L}$ and $\mathcal{R}$ are *indistinguishable by orthogonality tests* if

$$\varepsilon(n) = \max_{\boldsymbol{x}\in\mathbb{F}_{\nu(n)}^d} \left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L}_n : \langle \boldsymbol{l}, \boldsymbol{x}\rangle = 0\right] - \Pr\left[\boldsymbol{r} \leftarrow \mathcal{R}_n : \langle \boldsymbol{r}, \boldsymbol{x}\rangle = 0\right]\right|$$

   is a negligible function of n. Thus, the assumption states that for all $\mathcal{L}$ and $\mathcal{R}$ that are indistinguishable by orthogonality tests, $|A_{\mathcal{L}} - A_{\mathcal{R}}|$ is negligible as well for all PPT A.

5. This assumption is computationally falsifiable, though perhaps inefficiently. There are two possible obstructions to efficiency. First, the descriptions of $\mathcal{L}$ and $\mathcal{R}$ may be inefficient, although this is not a problem for the distributions constructed in our proof. Second, it may not be efficient to determine which orthogonality test performs the best. An interesting question is whether this computation can be performed efficiently, which would yield an efficient falsification procedure.

Hardness of the assumption

The following theorem exemplifies the strength of our assumption by relating it to Assumption 3.20, which is already considered by the cryptographic community to be quite strong.

Theorem 6.5. *Assumption 3.20 implies Assumption 6.4 for dimension 2. For higher dimensions, our assumption may be stronger because Assumption 6.4 for dimension $d+1$ implies Assumption 6.4 for dimension d.*

On the other hand, we provide evidence that our assumption is feasible by showing that it holds in the generic group model.

Theorem 6.6. *For all $d \in \mathbb{N}$, Assumption 6.4 for dimension d holds in the generic group model.*

Finally, note that we only consider constant dimensions d in this chapter. We justify this choice by showing that if d is allowed to depend on n, then the assumption quickly becomes unachievable. Loosely speaking, the problem is that the adversary can make queries adaptively.

Theorem 6.7. *If d is any super-logarithmic function of n, then Assumption 6.4 is false (even in the generic group model).*

This theorem is due to Nir Bitansky. Note that there is a gap in between the last two theorems: for super-constant but $O(\log(n))$ dimension, the status of our assumption and construction are unknown. However, the proof that the assumption holds in the generic group model and the proof of security for our construction require d to be a constant. Therefore, if the assumption and construction were secure for larger dimensions, new proof techniques would be necessary.

Proofs of these theorems are found in the next three subsections (Sections 6.3.1 through 6.3.3).

6.3.1 Hierarchy of assumptions

Here we prove Theorem 6.5, which we split into the following two lemmas.

Lemma 6.8. *Assumption 6.4 for dimension $d+1$ implies Assumption 6.4 for dimension d.*

Lemma 6.9. *Assumption 3.20 implies Assumption 6.4 for dimension 2.*

Proof of Lemma 6.8. Assume that Assumption 6.4 for dimension d is false, so for every prime sequence ν and every set of groups $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$, there exists a PPT adversary A and two distributions $\mathcal{L}$, $\mathcal{R}$ over vectors in $\mathbb{F}_\nu^d$ that are indistinguishable by orthogonality tests but such that $|A_\mathcal{L} - A_\mathcal{R}|$ is noticeable.

Now construct distributions $\mathcal{L}'$ and $\mathcal{R}'$ over vectors in $\mathbb{F}_\nu^{d+1}$ that sample $\mathcal{L}$ and $\mathcal{R}$, respectively, to obtain the first d components of the vector, and then sample the final component uniformly over $\mathbb{F}_\nu$. We claim that orthogonality tests do not distinguish $\mathcal{L}'$ from $\mathcal{R}'$.

Any orthogonality test $\boldsymbol{x'_n} \in \mathbb{F}_{\nu(n)}^{d+1}$ that has a non-zero final component will not distinguish $\mathcal{L}'_n$ from $\mathcal{R}'_n$ because the final component of these two distributions is uniform, so the inner product

will have the uniform distribution in both cases as well. Furthermore, if there exists a sequence of orthogonality tests $\{\boldsymbol{x'_n}\} \in \mathbb{F}_\nu^{d+1}$ that have zero for the final component and distinguish $\mathcal{L}'$ from $\mathcal{R}'$, then the sequence $\{\boldsymbol{x_n}\} \in \mathbb{F}_\nu^d$ formed by deleting the final component from $\boldsymbol{x'_n}$ distinguishes $\mathcal{L}$ and $\mathcal{R}$, contradicting our assumption that $\mathcal{L}$ and $\mathcal{R}$ are indistinguishable by orthogonality tests.

Finally, let A' be the adversary that drops its final component and feeds the rest to A. It is clear that $A'_{\mathcal{L}'} = A_{\mathcal{L}}$ and $A'_{\mathcal{R}'} = A_{\mathcal{R}}$, so $|A'_{\mathcal{L}'} - A'_{\mathcal{R}'}|$ is noticeable. Therefore, Assumption 6.4 for dimension $d+1$ is false as well. □

Proof of Lemma 6.9. Suppose that Assumption 3.20 holds. For every n, the assumption holds for a randomly chosen safe prime p, and thus for every n there exists some safe prime $p_n = 2q_n + 1$ for which it holds. Let G_n be the subgroup of quadratic residues in $\mathbb{F}_{p_n}^*$, and let $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$. We claim that Assumption 6.4 for dimension 2 holds for the family $\mathcal{G}$ and prime sequence $\nu(n) = q_n$.

It is clear that the first two properties of Assumption 6.4 for dimension 2 hold. Also, using our convention that the first non-zero coordinate of a vector is fixed to be 1, we may assume without loss of generality that every vector in $\mathbb{F}_\nu^2$ has the form $(1, x)$ for $x \in \mathbb{F}_{q_n}$ except for the vector $(0, 1)$, which is easy to test for. Thus, a "vector" is really just a group element. Furthermore, an "orthogonality test" is just an equality check because the vector $(y, -1)$ has an inner product of zero with the vector $(1, x)$ if and only if $y = x$.

Hence, it remains to prove the following: for every PPT adversary A and for all families of distributions $\mathcal{L}$ and $\mathcal{R}$ over $\{\mathbb{F}_{q_n}\}_{n\in\mathbb{N}}$ such that

$$\max_{x\in\mathbb{F}_{q_n}} \left|\Pr\left[l \leftarrow \mathcal{L} : l = x\right] - \Pr\left[r \leftarrow \mathcal{R} : r = x\right]\right|$$

is negligible, the quantity

$$\left|\Pr\left[l \leftarrow \mathcal{L}_n, g \xleftarrow{U} G_n : A(g, g^l) = 1\right] - \Pr\left[r \leftarrow \mathcal{R}_n, g \xleftarrow{U} G_n : A(g, g^r) = 1\right]\right|$$

is negligible as well.

First, we prove that the statement holds when $\mathcal{L}$ is well-spread and $\mathcal{R}$ is the uniform distribution. Hence, we wish to show that for all PPT A,

$$\left|\Pr\left[l \leftarrow \mathcal{L}_n, g \xleftarrow{U} G_n : A(g, g^x) = 1\right] - \Pr\left[r \xleftarrow{U} \mathbb{F}_{q_n}, g \xleftarrow{U} G_n : A(g, g^r) = 1\right]\right|$$

is negligible. The proof of this statement closely follows the proofs in [23], so we only sketch the details here. If this statement is not true, then the probability $P_x = A_{(1,x)} = \Pr\left[A(g, g^x) = 1\right]$ is noticeably different from the mean value $\bar{P} = A_{\mathcal{R}}$ for super-polynomially many values of x. Without loss of generality, there exist super-polynomially many values a for which P_a is noticeably larger than $\bar{P}$. Let X_{q_n} be the uniform distribution over all such a. Then, the ensembles $\langle g, g^a, g^b, g^c\rangle$ and $\langle g, g^a, g^b, g^{ab}\rangle$ are distinguishable when $a \leftarrow X_{q_n}$ by running A on the final two components of the ensemble. In the first case, A outputs 1 with probability $\bar{P}$, and in the second case, A outputs 1 with noticeably higher probability. This contradicts Assumption 3.20.

Next, we note that the statement immediately extends to the setting where both $\mathcal{L}$ and $\mathcal{R}$ are well-spread by a simple hybrid argument.

Finally, we consider arbitrarily distributions $\mathcal{L}$ and $\mathcal{R}$ such that

$$\max_{x\in\mathbb{F}_{q_n}} \left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L} : l = x\right] - \Pr\left[r \leftarrow \mathcal{R} : r = x\right]\right|$$

is negligible. In words, this equation means that for every x that occurs with noticeable probability

in $\mathcal{L}$, it occurs with the same probability in $\mathcal{R}$ as well up to a negligible difference. Thus, the distributions $\mathcal{L}$ and $\mathcal{R}$ can only differ on outcomes that occur with negligible probability. Therefore, it suffices to consider $\mathcal{L}$ and $\mathcal{R}$ that are well-spread, and in this case we showed that for every PPT A,

$$\left|\Pr\left[l \leftarrow \mathcal{L}_n, g \xleftarrow{U} G_n : A(g, g^l) = 1\right] - \Pr\left[r \leftarrow \mathcal{R}_n, g \xleftarrow{U} G_n : A(g, g^r) = 1\right]\right|$$

is negligible, so Assumption 6.4 for dimension 2 holds as desired. □

We note that a literal converse to this lemma does not quite make sense because Assumption 3.20 is specific to the group of quadratic residues modulo $\mathbb{F}_p^*$ for a safe prime p, whereas Assumption 6.4 makes the more general claim that there exists some family of groups that satisfy a certain condition (potentially quite different from the groups used in Assumption 3.20).

6.3.2 Generic group model

In this section, we prove Theorem 6.6, which states that Assumption 6.4 holds in the generic group model. We caution that this section is rather technical, and it may be safely skipped by the uninterested reader who is willing to take the hardness of Assumption 6.4 on faith.

To begin, we explain the ideas behind the generic group model and its formalizations.

Binary encodings of group elements. Recall that in our assumption, the adversary receives several group elements as input. More precisely, A receives binary encodings that represent the group elements in some way. For instance, if G_n is the group of quadratic residues $Q \leq \mathbb{F}_p^*$ (as in Assumption 3.20), then we can represent group elements as strings using the natural embeddings[4]

$$Q \hookrightarrow \mathbb{F}_p \hookrightarrow \mathbb{Z} \hookrightarrow \{0,1\}^*,$$

where in the final step we form a string using the binary representation of integers. By making strings in this way, it is simple for an adversary to compute the group operation, inverse operation, and test elements for equality in Q.

Similarly, one could consider elliptic curve groups over finite fields, which also have a natural binary encoding that allows the adversary to observe the group structure. It is believed that the Decisional Diffie-Hellman problem is hard for certain elliptic curve groups.

It is possible that an adversary A might be able to break the discrete logarithm or DDH assumptions using only the group structure of Q. Such an adversary would be "generic" in the sense that it would break the corresponding assumption for elliptic curve groups as well. However, the adversary may instead be able to exploit extra properties of Q beyond its group structure. For example, index calculus algorithms can use the prime factorization of the integer corresponding to a group element in order to compute discrete logarithms [74]. An algorithm of this type does not immediately translate into a corresponding algorithm on elliptic curve groups. In fact, for certain elliptic curve groups, generic algorithms provide essentially the best known attacks on the discrete logarithm and DDH problems [14, 57].

The model. The *generic group model* is an abstract model in which the structure of the group is "hidden" from the adversary. Specifically, the embedding between group elements and their binary representations is chosen uniformly at random. Additionally, we give the adversary oracles that compute the embedding and perform the group's multiplication and inverse operations on the

[4]We stress that these are embeddings of sets, not groups.

binary representations of elements. Thus, an adversary can access the group structure but cannot exploit any properties of the representation of the group elements, aside from the fact that each element has a distinct, unique representation.

The generic group model is commonly used in three ways:

1. Demonstrate the hardness of solving number-theoretic problems like factoring or computing discrete logarithms [2, 35, 62, 65, 73, 87]

2. Relate the hardness of different problems [4, 19, 64, 66]

3. Provide evidence for the feasibility of new assumptions or the security of new cryptographic protocols [15, 20, 37, 85, 88]

In this section, we do the latter. There is a word of caution here: proving that an assumption holds in the generic group model does not guarantee that the assumption is true for *any* concrete group [36, 40, 59]. Nevertheless, it does show that any algorithm attacking the assumption must be non-generic in nature and make use of the underlying features of group elements. Finding such algorithms for certain groups (such as elliptic curve groups over finite fields) would shed new light on their structure. Therefore, resolving the status of our assumption is interesting either way.

Formalizations. Now we construct a "generic group" with the structure of a given family of groups $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$. There are two ways to formalize the generic group model. The first formalization is due to Shoup [87] and the second one is due to Maurer [65].

First, we can choose a random encoding $\sigma : \mathcal{G} \hookrightarrow \{0,1\}^*$ in advance. Then, rather than giving group elements as input to an adversary A, we provide the encodings of group elements under σ. In our problem, the adversary receives

$$A(\sigma(g_1), \sigma(g_2), \ldots, \sigma(g_d))$$

as input. For notational simplicity, we write this instead as $A(\sigma; g_1, \ldots, g_d)$.

To allow A to make intelligent use of these encodings, we provide two oracles: one that computes σ, and another oracle Add_σ that performs the group operation on encoded strings. Specifically, $\mathrm{Add}_\sigma(s, s', b)$ takes as input two strings s, s' and a bit b, finds group elements g, g' such that $s = \sigma(g)$ and $s' = \sigma(g')$, and outputs $\sigma(g + (-1)^b g')$.

The second formalization does not choose σ in advance but rather builds it "on the fly." That is, whenever σ or Add_σ must output the embedding of a new group element, the oracle chooses a new string uniformly at random. Hence, σ and Add_σ jointly maintain a table T of all the group elements they have seen so far together with their encodings. This table is required to respond consistently to the encodings of group elements that have been seen before.

Jager and Schwenk showed that the two formalizations are equivalent [57]. For simplicity, we use the second formalization in our proof.

With the required background complete, we can now present a formal statement of the theorem.

Theorem 6.10 (Precise statement of Theorem 6.6)**.** *Let $d \in \mathbb{N}$ and ν be a prime sequence. Then, Assumption 6.4 holds in the generic group model over the family of groups $\mathbb{F}_\nu$. Specifically, there exists an oracle σ that creates an embedding $\mathbb{F}_{\nu(n)} \hookrightarrow \{0,1\}^n$ on the fly, as well as an addition oracle Add_σ that operates as described above, such that*

1. *There are efficient algorithms to perform the group and inverse operations, to test for equality with the identity element, and to sample uniformly on encodings of group elements.*

2. *Let $\mathcal{L}$ and $\mathcal{R}$ be two distributions over $\mathbb{F}_\nu^d$ that are indistinguishable by orthogonality tests. Then, for any PPT adversary A,*

$$\left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L}_n, A^{\sigma,\mathrm{Add}_\sigma}(\sigma; l_0, \ldots, l_d) = 1\right] - \Pr\left[\boldsymbol{r} \leftarrow \mathcal{R}_n : A^{\sigma,\mathrm{Add}_\sigma}(\sigma; r_0, \ldots, r_d) = 1\right]\right|$$

is negligible, where the probabilities are taken over the random coin tosses of A and the uniform choice of σ.

The rest of this section is devoted to a proof of the theorem. The first part of the theorem is easy to see: the oracle Add_σ can be used to perform the group and inverse operations, testing for equality with the identity element can be done with a string comparison to $\sigma(1)$, and one can sample uniformly on encodings by choosing $a \xleftarrow{U} \mathbb{F}_{\nu(n)}$ and returning $\sigma(a)$.

Proving the second part of the theorem is significantly more complicated. Suppose for the sake of contradiction that there exist A, $\mathcal{L}$, $\mathcal{R}$, and a polynomial ρ such that

$$\left|\Pr\left[\boldsymbol{l} \leftarrow \mathcal{L}_n, A^{\sigma,\mathrm{Add}_\sigma}(\sigma; l_0, \ldots, l_d) = 1\right] - \Pr\left[\boldsymbol{r} \leftarrow \mathcal{R}_n : A^{\sigma,\mathrm{Add}_\sigma}(\sigma; r_0, \ldots, r_d) = 1\right]\right| > \frac{1}{\rho(n)}. \quad (6.2)$$

Then, we show that there exists a polynomial ρ' and a sequence of orthogonality tests $\{\boldsymbol{x_n}\}_{n\in\mathbb{N}}$ such that $\boldsymbol{x_n}$ distinguishes[5] $\mathcal{L}_n$ from $\mathcal{R}_n$ with probability at least ρ'.

Simplifications. We start by making four simplifications. First, we can assume without loss of generality that A is deterministic. This is done by a straightforward argument: there must exist some setting of A's random tape such that (6.2) holds, or else it cannot be true on average, and by non-uniformity we hardcode this setting of the random tape into A.

Second, we can remove the oracle σ, leaving A with only one oracle to Add_σ. This is possible by our convention that the first non-zero coordinate of a vector equals 1. Therefore, A can compute $\sigma(0)$ by subtracting any encoding from itself, and the first input to A that isn't equal to $\sigma(0)$ must be $\sigma(1)$. Then, any oracle query $\sigma(a)$ can be simulated with oracle queries to Add_σ by making the simple observation that

$$a = \underbrace{1 + 1 + \cdots + 1}_{a \text{ times}}.$$

Hence, A can compute $\sigma(a)$ efficiently by adding $\sigma(1)$ to itself many times using the technique of repeated doubling.

Recall that the oracle Add_σ stores a table T of group elements and their corresponding encodings for every distinct group element that A queries. In our third observation, we note that this table is polynomially-bounded in length. To see why this is true, note that there are three ways that an entry can be added to T in Maurer's model.

1. An encoding must be assigned for each of A's input elements.
2. If A makes an oracle query $\mathrm{Add}_\sigma(s, s', b)$ such that one or both of the strings s, s' are not in T, then they must be added to T as the encoding of a new group element.
3. If an oracle query $\mathrm{Add}_\sigma(s, s', b)$ is made such that $\sigma^{-1}(s) + (-1)^b \cdot \sigma^{-1}(s')$ is not already in the table, it is added to the table and assigned a new encoding.

For now, assume that case 2 never occurs; we will justify this assumption at the end of the proof. Because A runs in polynomial time, there exists a polynomial q such that A receives at most q

[5] It is not necessary to *find* the distinguishing test efficiently; we merely have to show that such a test exists.

encoded strings from its inputs and oracle query responses. A simple bound is $q = d + \text{size}(A)$. By making additional queries if necessary, we may assume without loss of generality that A makes exactly $q - d$ entries.

Fourth, we make a slight modification to Maurer's model [65]. Rather than choosing an encoded string "on the fly" for every new group element encountered, we will "pre-process" this step. That is, we'll choose q strings $\sigma_1, \ldots, \sigma_q \xleftarrow{U} \{0,1\}^n$ in advance. Then, σ_1 through σ_d are the input strings to A (unless some of the input elements coincide), and whenever Add_σ has to respond with a new string, it simply chooses the first unused string in this list. Just as we can assume without loss of generality that A is deterministic, we can fix the strings $\sigma_1, \ldots, \sigma_q$ because there must exist some setting of the strings under which (6.2) holds.[6]

Analyzing oracle query responses. With these simplifications in place, the behavior of A is "consistent" in the sense that if we run A on the same input many times, the output is always the same. More precisely, for any $\boldsymbol{v} \in \mathbb{F}_{\nu(n)}$, the behavior of $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$ is deterministic because A and σ are both deterministic, so the input strings of A are uniquely determined. From this, it is easy to see inductively that A's oracle query inputs and outputs are the same in every execution, so the output of A is also uniquely determined. Incidentally, note that the table T is constructed in the same way every time.

This observation leads to a stronger statement. Suppose $\boldsymbol{v}$ and $\boldsymbol{v'}$ are two vectors such that the executions of $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$ and $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v'})$ never result in any oracle query "collisions": that is, the d components of each vector are distinct, and each output of the oracle Add_σ represents a new group element (not one that was encountered before). Then, the view of A is identical in the two executions, because A receives $\sigma_1, \ldots, \sigma_d$ as inputs and $\sigma_{d+1}, \ldots, \sigma_q$ as the responses to its oracle queries in both cases. In other words, the table of encodings T is the same in both executions. Since A is deterministic, it follows that its output must be identical in both cases.

This property extends to any two vectors $\boldsymbol{v}$ and $\boldsymbol{v'}$ that have the same "response pattern" to their oracle queries, even if they are not always distinct. To codify this statement, we must consider all of the possible responses to the oracle queries made by $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$. The first input coordinate v_1 is always encoded as σ_1. The second input coordinate v_2 is then encoded as σ_2, unless $v_2 = v_1$ in which case the encoding of v_2 is also σ_1. By continuing this process, we can make a tree Q of all the possible responses of the oracle queries. Figure 6-1 shows the first three levels of this tree.

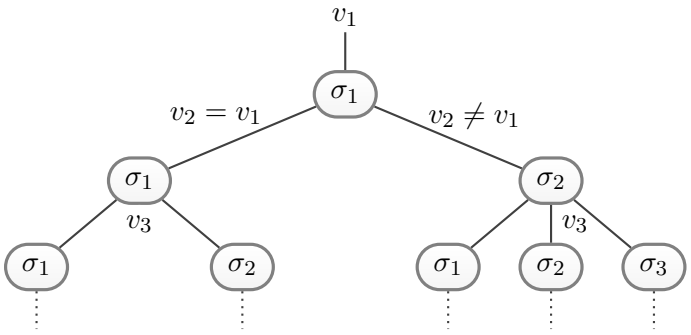

Figure 6-1: Query response tree. Edges denote group elements and nodes denote responses.

[6] By using the same strings $\sigma_1, \ldots, \sigma_q$ for different inputs, the embedding σ that we are constructing "on the fly" is different for every input to A. If our proof used Shoup's model [87] instead, we could make a similar argument but would have to describe a permutation between different embeddings under which the view of A remains constant.

The depth of this tree is q, and each node in the tree is defined by the query response pattern up to that point. Note that the i^{th} oracle query has only i possible responses: the strings $\sigma_1, \sigma_2, \ldots, \sigma_i$. Hence, each node is uniquely identified by a tuple of at most q integers such that the first coordinate equals 1 and every coordinate is at most 1 more than the maximum of all the previous coordinates. For instance, Figure 6-1 shows that there are five nodes at the third level of the tree: $Q(1,1,1)$, $Q(1,1,2)$, $Q(1,2,1)$, $Q(1,2,2)$, and $Q(1,2,3)$.

We are interested in the vectors that reach any given node. Given a vector $\boldsymbol{v} \in \mathbb{F}^d_{\nu(n)}$, we know that $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$ has one of five possible query response patterns after three queries are made, so it "belongs" to one of the five nodes shown at the bottom of Figure 6-1. Continue this procedure down to the leaves of the tree. This creates a partition of the vectors in $\mathbb{F}^d_{\nu(n)}$ into several sets, one for each leaf, where the set $V(Q(c_1, \ldots, c_q))$ contains every vector $\boldsymbol{v}$ such that:

- In an execution of A, component v_1 is mapped to the string σ_{c_1}, v_2 is mapped to σ_{c_2}, and so on. Hence, A receives $(\sigma_{c_1}, \ldots, \sigma_{c_d})$ as input.
- For $i \in [q-d]$, the response to the i^{th} oracle query of $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$ is the string $\sigma_{c_{d+i}}$.

Now we can formalize the statement made above. For any two vectors $\boldsymbol{v}$ and $\boldsymbol{v'}$ that have the same "response pattern," in the sense that there exist $c_1, \ldots, c_q$ such that $\boldsymbol{v}$ and $\boldsymbol{v'}$ belong to the same set $V(Q(c_1, \ldots, c_q))$, the view of A is identical in the executions of $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$ and $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v'})$, so the output of A must be the same in both instances. Hence, A cannot distinguish $\boldsymbol{v}$ from $\boldsymbol{v'}$.

Learning from collisions. We showed above that if the components of $\boldsymbol{v}$ and the oracle query responses to Add_σ are distinct in an execution of $A^{\text{Add}_\sigma}(\sigma; \boldsymbol{v})$, then A does not "learn anything" about $\boldsymbol{v}$, in the sense that its view is indistinguishable from that of every other input vector with these properties. Conversely, when an oracle query response is a string that has been seen before, then we can learn information about $\boldsymbol{v}$.

To do so, we can add a third column to the table T that keeps track of the relationship of each group element to the input elements. We associate each input element v_i with an indicator vector $\boldsymbol{y_{v_i}} = (0, \ldots, 0, 1, 0, \ldots, 0)$ that has a 1 in the i^{th} position. Then, for each oracle query $\text{Add}_\sigma(s, s', b)$, we find the group elements $g, g' \in \mathbb{F}_{\nu(n)}$ and vectors $\boldsymbol{y_g}, \boldsymbol{y_{g'}} \in \mathbb{F}^d_{\nu(n)}$ in T that correspond to the strings, and add a new entry to T with the group element $g + (-1)^b g'$, vector $\boldsymbol{y_g} + (-1)^b \boldsymbol{y_{g'}}$, and the appropriate encoded string (based on our pre-processing technique).

This procedure has an important property for every entry in T: the group element g and its corresponding vector $\boldsymbol{y_g}$ are related by $g = \langle \boldsymbol{v}, \boldsymbol{y_g} \rangle$.

Now suppose an oracle query results in a group element $g'' = g + (-1)^b g'$ has already been seen before, and thus is already in the table T along with a vector $\boldsymbol{y_{g''}}$. Then, it follows that

$$\langle \boldsymbol{v}, \boldsymbol{y_g} + (-1)^b \boldsymbol{y_{g'}} - \boldsymbol{y_{g''}} \rangle = 0.$$

Hence, finding a collision in an oracle query yields one linear constraint on the input vector $\boldsymbol{v}$.

Limits to learning. While the tree Q is super-polynomial in size, we claim that there are only polynomially many leaves $\bar{c} = (c_1, \ldots, c_q)$ such that $V(Q(\bar{c}))$ is non-empty. To prove this claim, we combine the analysis of oracle query responses with the learning from collisions property to "prune" nodes in Q that are never reached. It helps to define sets $V(Q(c_1, \ldots, c_k))$ on non-leaf nodes as the union of all the sets $V(Q(c_1, \ldots, c_k, c'_{k+1}, \ldots, c'_q))$ for the leaves in the subtree rooted at $(c_1, \ldots, c_k)$.

Each time an oracle query collides with a prior one, a linear constraint on $\boldsymbol{v}$ is revealed, and we "move to a left child" in Q (that is, rather than moving to the rightmost leaf, we follow a different path). Once we "move left" enough times to learn $d-1$ linearly independent constraints[7] on $\boldsymbol{v}$, the vector can be identified uniquely. Hence, only one vector is in the set corresponding to this node, so there is only one non-empty leaf in the subtree rooted at this node, and we can prune all the other leaves.

Of course, it is not clear how many times one must "move left" in order to learn linearly independent constraints. There may be linearly dependent constraints along the way, but the tree can be pruned in this case too. Given an input vector $\boldsymbol{v}$, suppose A's first k oracle responses yield the pattern $(c_1, \ldots, c_k)$ with j "left" moves. As a result, we learn that $\boldsymbol{v}$ satisfies j linear constraints $\langle \boldsymbol{v}, \boldsymbol{x_1} \rangle = \cdots = \langle \boldsymbol{v}, \boldsymbol{x_j} \rangle = 0$.

Suppose A's next oracle query reveals a new constraint $\langle \boldsymbol{v}, \boldsymbol{x_{j+1}} \rangle = 0$ for a vector $\boldsymbol{x_{j+1}}$ that is linearly dependent on $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_j}\}$, and suppose this causes A to follow path c_{k+1}. We claim that all the children of $Q(c_1, \ldots, c_k)$ except c_{k+1} can be pruned. Every vector $\boldsymbol{v'}$ that reaches node $Q(c_1, \ldots, c_k)$ also satisfies the constraints $\langle \boldsymbol{v'}, \boldsymbol{x_1} \rangle = \cdots = \langle \boldsymbol{v'}, \boldsymbol{x_j} \rangle = 0$, and thus also satisfies $\langle \boldsymbol{v'}, \boldsymbol{x_{j+1}} \rangle = 0$ by linear dependence. Hence, every vector in $V(Q(c_1, \ldots, c_k))$ is also in $V(Q(c_1, \ldots, c_k, c_{k+1}))$ and we can prune all of the other children.

After pruning the empty leaves, the remaining tree has the following two properties.

1. It has depth q and each non-leaf node has at most q children.
2. In every path down the tree, a child aside from the rightmost one is taken at most $d-1$ times. (If a node has only one child, it is denoted the rightmost one.)

We can bound the number of leaves in such a tree by a simple combinatorial argument. First, we choose a number $0 \leq i \leq d-1$ of levels in the tree to "go left." There are $\binom{q}{i}$ ways to make this decision. For each such level, there are $q-1$ children to pick. Therefore, Q has at most

$$\sum_{i=1}^{d-1} \binom{q}{i} \cdot (q-1)^i = \mathrm{O}(q^{2d+1})$$

leaves whose vector sets are non-empty.

Finding an orthogonal test. At each edge on the pruned tree, A makes one oracle query to Add_σ so it learns at most one linear constraint on $\boldsymbol{v}$. We pool all of these linear constraints together into a polynomial-sized set of vectors $X_n \subseteq \mathbb{F}_{\nu(n)}^d$ such that A effectively computes all the orthogonality tests in X_n; the adversary A does not learn any more information than this because A's view is identical for all the vectors in the same leaf node. Additionally, the size of X_n is bounded by $\mathrm{O}(q^{2d+2})$, which is polynomial in n because d is a constant[8] and q is polynomial in n. Concretely, let $s(n)$ be a polynomial such that $|X_n| \leq s(n)$.

Recall that the adversary A distinguishes $\mathcal{L}_n$ from $\mathcal{R}_n$ with probability at least $\frac{1}{\rho(n)}$. By a standard union bound argument, there exists an orthogonality test $\boldsymbol{x_n} \in X_n$ that distinguishes $\mathcal{L}_n$ from $\mathcal{R}_n$ with probability at least $\frac{1}{\rho(n)s(n)}$. This can be done for every n, yielding a sequence of orthogonality tests $\{\boldsymbol{x_n}\}$ that distinguishes $\mathcal{L}$ from $\mathcal{R}$ with noticeable probability $\frac{1}{\rho'}$, where $\rho' = \rho \cdot s$. This completes the proof of Theorem 6.6.

[7] Following our convention that the first non-zero coordinate of $\boldsymbol{v}$ equals 1 gives one linear constraint on $\boldsymbol{v}$. Hence, only $d-1$ more are required to identify $\boldsymbol{v}$ uniquely.

[8] This step depends crucially on the fact that d is constant. It should not be surprising that this proof is dependent on the size of d, since the theorem is false if d is a super-logarithmic function of n (see Theorem 6.7).

Oracle queries in the dark. The only remaining step in the proof is to justify the assumption above that A never makes a "case 2" oracle query of the form $\mathrm{Add}_\sigma(s, s', b)$ such that the string s is not in T. (The same may be true of s' as well, but is not required.) Intuitively, these oracle queries can be answered in a manner that is unrelated to the rest of A's computation, so they do not help A to learn any useful information.

Because the string s is not yet in T, we must add it along with a group element g and vector $\boldsymbol{y_g}$. The idea is to treat g has a "free variable" that does not depend on any of the prior known information, and then "quarantine" this variable so A doesn't learn anything about it.

Specifically, we increase the dimension of the vectors in T (which is initially d) by one and set the new coordinate of this vector to 0 for every vector in T. Then, we set $\boldsymbol{y_g} = (0, \ldots, 0, 1)$. Recall that A only learns useful information about a group element through Add_σ oracle collisions, and in each such collision A learns one linear constraint. With the new free variable in place, this corresponds to a vector $\boldsymbol{x} \in \mathbb{F}_{\nu(n)}^{d+1}$ such that $\langle (v_1, \ldots, v_d, g), \boldsymbol{x} \rangle = 0$. If the final coordinate of $\boldsymbol{x}$ is 0, then this constraint does not depend on the free variable.

Finally, we choose $g \in \mathbb{F}_{\nu(n)}$ in such a way that future linear constraints are independent of it. This can be done (inefficiently) by enumeration: once g is set, the adversary A can be emulated to completion. Suppose a linear constraint

$$\langle (v_1, \ldots, v_d, g), \boldsymbol{x} \rangle = 0 \tag{6.3}$$

is found with $x_{d+1} \neq 0$. This constraint is dependent on g, but changing its value to any other element of $\mathbb{F}_{\nu(n)}$ eliminates (6.3). The adversary makes polynomially many queries, and thus there are only polynomially many constraints to avoid, whereas there are exponentially many choices of g, so it is possible to find one that works. Furthermore, this technique easily generalizes to the case of multiple "case 2" queries.

6.3.3 Adaptivity

In this section, we prove Theorem 6.7, which says that Assumption 6.4 is false when d is a super-logarithmic function of n. This proof does *not* exploit the ability of an adversary to do anything more sophisticated than performing orthogonality tests. Instead, the proof uses the fact that an adversary can choose orthogonality tests adaptively based on the results of prior tests. Also, the proof does not exploit any property of the family of groups $\mathcal{G}$ other than its group structure, so the impossibility result holds in the generic group model as well.

In the proof, we construct a PPT adversary A and families of distributions $\mathcal{L}$, $\mathcal{R}$ with disjoint supports that satisfy the following three properties.

1. Given any vector $\boldsymbol{a} \in \mathrm{Supp}(\mathcal{L}) \cup \mathrm{Supp}(\mathcal{R})$, there exists a "special" orthogonality test that can determine whether $\boldsymbol{a}$ is in $\mathcal{L}$ or $\mathcal{R}$.

2. The adversary can find the special test adaptively based on the outcomes of other tests.

3. The special test is different for every $\boldsymbol{a}$, so any particular test fails most of the time.

These three properties break Assumption 6.4 because there exists a super-polynomial relationship between the success probability of the adversary (which is 1) and that of the best orthogonality test (which is negligible). A formal proof follows.

Proof of Theorem 6.7. Let $f \in \omega(\log(n))$ and $\mathcal{G} = \{G_n\}_{n \in \mathbb{N}}$ be a family of groups, and we show that Assumption 6.4 is false for $d = f(n)$ and $\mathcal{G}$. Without loss of generality, we may assume that

orders of the groups form a prime sequence ν, and there are efficient algorithms to perform the group and inverse operations, to test for equality with the identity element, and to sample uniformly from $\mathcal{G}$ (otherwise the assumption is discernibly false).

Given $n \in \mathbb{N}$, we construct a distribution $\mathcal{L}_n$ as follows.

choose an integer $B \xleftarrow{U} \{0, 1, \ldots, 2^{d-2} - 1\}$
for $i = 1$ to $d - 2$ **do**
  if $b_i = 0$, where b_i denotes the i^{th} bit in the binary representation of B **then**
    set $a_i \leftarrow 0$
  else
    choose $a_i \xleftarrow{U} \mathbb{F}_{\nu(n)} \setminus 0$
  end if
end for
output the vector[9] $\boldsymbol{a} = (a_1, \ldots, a_{d-2}, 1, B)$ in $\mathbb{F}^d_{\nu(n)}$

We construct distribution $\mathcal{R}_n$ by an identical procedure except that we output the vector $\boldsymbol{a} = (a_1, \ldots, a_{d-2}, 1, B + 1)$; that is, the final coordinate of $\boldsymbol{a}$ is different. We do this for all n to build families of distributions $\mathcal{L}$ and $\mathcal{R}$.

Next, we construct a (uniform) adversary A distinguishes $\mathcal{L}$ from $\mathcal{R}$ with probability 1. The adversary A receives as input $g^{a_1}, g^{a_2}, \ldots, g^{a_{d-2}}, g, g^{a_d}$, where a_d equals B or $B + 1$. We know that there exists an algorithm that A can use to determine whether each g^{a_i} equals the identity element of G_n. If it does, A knows that $b_i = 0$, and if not, then A knows that $b_i = 1$. In this way, A recovers B perfectly. Finally, A tests whether g^{a_d} equals g^B: if so, then the vector comes from $\mathcal{L}$, and if not, then the vector comes from $\mathcal{R}$. It is straightforward to show that A always succeeds.

To show that the assumption is false, it suffices to prove that every orthogonality test $\boldsymbol{x} \in \mathbb{F}^d_{\nu(n)}$ only succeeds with negligible probability. Note that if $x_d = 0$, then the test $\boldsymbol{x}$ cannot distinguish between $\mathcal{L}$ and $\mathcal{R}$ at all because their constructions only differ in the final component. Hence, from now on it suffices to consider the case in which $x_d \neq 0$. Next, we condition on the choice of B. There are two cases to consider.

Case 1: Suppose $\boldsymbol{x}$ and B have the property that for every $i \in [d - 2]$ such that $x_i \neq 0$, $b_i = 0$. As a result, $a_i = 0$ as well for these indices, so it follows that

$$\langle \boldsymbol{a}, \boldsymbol{x} \rangle = x_{d-1} + a_d x_d,$$

which only equals zero when $a_d = \frac{-x_{d-1}}{x_d}$ (recall that $x_d \neq 0$ so this is well-defined). Because a_d always equals B or $B + 1$, it follows that $\boldsymbol{x}$ only distinguishes $\mathcal{L}$ from $\mathcal{R}$ for two choices of B. Hence, $\boldsymbol{x}$ succeeds with probability $\frac{2}{2^{d-2}}$ over the random choice of B.

Case 2: Suppose there exists an index i such that both x_i and b_i are non-zero, and thus a_i is non-zero as well. Then, we can consider what happens when we vary a_i over all non-zero values in $\mathbb{F}^d_{\nu(n)}$ but keep B and all of the other components of $\boldsymbol{a}$ fixed. Notice that

$$\langle \boldsymbol{a}, \boldsymbol{x} \rangle = c + a_i x_i,$$

where c is a constant that only depends on whether $\boldsymbol{a}$ comes from $\mathcal{L}$ or $\mathcal{R}$. In each case, there is a unique value $a_i = \frac{-c}{x_i}$ such that the inner product equals zero. Hence, $\boldsymbol{x}$ only distinguishes $\mathcal{L}$ from $\mathcal{R}$ for two choices of $a_i \in \mathbb{F}_{\nu(n)}$.

[9] Recall that the vectors in $\mathcal{L}$ and $\mathcal{R}$ are only defined up to constant multiplication. We usually enforce this rule by setting the first non-zero coordinate of the vector to 1. Here, we set $a_{d-1} = 1$ instead for notational simplicity.

In summary, the orthogonality test $\boldsymbol{x}$ succeeds with probability

$$\begin{aligned}
&\Pr[\boldsymbol{x} \text{ distinguishes } \mathcal{L} \text{ from } \mathcal{R}] \\
&\quad = \Pr[\boldsymbol{x} \text{ distinguishes } \mathcal{L}, \mathcal{R} \text{ and in Case 1}] + \Pr[\boldsymbol{x} \text{ distinguishes } \mathcal{L}, \mathcal{R} \text{ and in Case 2}] \\
&\quad = \frac{2}{2^{d-2}} + \Pr[\boldsymbol{x} \text{ distinguishes } \mathcal{L}, \mathcal{R} \mid \boldsymbol{x}, B \text{ in Case 2}] \cdot \Pr[\boldsymbol{x}, B \text{ in Case 2}] \\
&\quad \leq \frac{2}{2^{d-2}} + \frac{2}{\nu(n)} \cdot 1 < \frac{1}{2^{f(n)-3}} + \frac{1}{2^{n-2}},
\end{aligned}$$

which is negligible in n because $f \in \omega(\log(n))$ and $\nu(n) > 2^{n-1}$. Therefore, no polynomial ξ can relate the success probability of adversary A to the success probability of any linear test, so Assumption 6.4 is false as desired. □

6.4 Construction

In this section, we define the family of programs that we obfuscate, present the obfuscator, and prove its security under Assumption 6.4.

Let d be an integer and ν be a prime sequence. Given a vector $\boldsymbol{a} \in \mathbb{F}_{\nu(n)}^d$, let $H_{\boldsymbol{a}}$ be the circuit that has $\boldsymbol{a}$ hardwired, and on input $\boldsymbol{x} \in \mathbb{F}_{\nu(n)}^d$, computes $\langle \boldsymbol{a}, \boldsymbol{x} \rangle$ in the obvious way and accepts if and only if the inner product equals 0. Let $\mathcal{F}_\nu^d = \{H_{\boldsymbol{a}} : n \in \mathbb{N}, \boldsymbol{a} \in \mathbb{F}_{\nu(n)}^d\}$ be the family of all such circuits.

We show how to obfuscate the family $\mathcal{F}_\nu^d$ for any $d \in \mathbb{N}$, prime sequence ν, and set of groups $\mathcal{G}$ (written multiplicatively) that satisfy Assumption 6.4 for dimension d. The obfuscator $\mathcal{O}_{\mathcal{G},d}$ operates as follows.

Algorithm 6.1 Obfuscator $\mathcal{O}_{\mathcal{G},d}$ for the family of hyperplane membership testing programs $\mathcal{F}_\nu^d$

Input: vector $\boldsymbol{a} = (a_1, \ldots, a_d)$ in $\mathbb{F}_{\nu(n)}^d$

1: choose a generator $g \xleftarrow{U} G_n \setminus \{1_{G_n}\}$ uniformly at random
2: compute $g_i \leftarrow g^{a_i}$ for $i = 1, \ldots, d$

Output: circuit that has $g_1, \ldots, g_d$ hardwired, and on input a vector $\boldsymbol{x}$, accepts if $\prod_{i=1}^d g_i^{x_i} = 1_{G_n}$

We stress that the generator g is not made public in addition to the g_i. However, recall that the vector $\boldsymbol{a}$ is only defined uniquely up to scalar multiplication, and that one way to enforce this requirement is to assume that the first non-zero coordinate of $\boldsymbol{a}$ equals 1. With this convention, the generator g is revealed.

Also, this convention makes it clear that in the $d = 2$ case, this construction is the same as that of Canetti [23], and it can be based on the same DDH assumption by Theorem 6.5. Hence, our construction subsumes the one in [23], which was described in Section 3.2.1.

We now show that $\mathcal{O}_{\mathcal{G},d}$ is an obfuscator, based on Assumption 6.4.

Theorem 6.11. *Let $d \in \mathbb{N}$ and $\mathcal{G}$ be a set of groups satisfying Assumption 6.4. Then, the algorithm $\mathcal{O}_{\mathcal{G},d}$ is an obfuscator for the family $\mathcal{F}_\nu^d$ with exact functionality.*

It is clear that $\mathcal{O}_{\mathcal{G},d}$ satisfies the exact functionality and polynomial slowdown properties required of an obfuscator, so it remains to prove the virtual black-box property. Before doing so, we present a definition that will be useful throughout the proof and an intermediate lemma.

Definition 6.12. Let $d \in \mathbb{N}$ and p be a prime number. We say that the set $V \subseteq \mathbb{F}_p^d$ *distinguishes* two vectors $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{F}_p^d$ if there exists $\boldsymbol{x} \in V$ such that exactly one of the inner products $\langle \boldsymbol{l}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{r}, \boldsymbol{x} \rangle$ equals 0. Otherwise, we say that $\boldsymbol{l}$ *and* $\boldsymbol{r}$ *are indistinguishable by* V, which means that for all $\boldsymbol{x} \in V$, $\langle \boldsymbol{l}, \boldsymbol{x} \rangle = 0$ if and only if $\langle \boldsymbol{r}, \boldsymbol{x} \rangle = 0$.

At a high level, this lemma states that for every adversary A, there exists a set V that can distinguish vectors in $\mathbb{F}_{\nu(n)}^d$ as well as A can.

Lemma 6.13. *Suppose $(\mathcal{G}, d)$ satisfy Assumption 6.4. For every PPT adversary A and polynomial ρ, there exists a polynomial s (that can depend on A) such that for every $n \in \mathbb{N}$, there exists a set $V \subseteq \mathbb{F}_{\nu(n)}^d$ of size at most $s(n)$, such that for every pair of vectors $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{F}_{\nu(n)}^d$ that are indistinguishable by V,*[10]

$$|A_{\boldsymbol{l}} - A_{\boldsymbol{r}}| = \left| \Pr\left[g \xleftarrow{U} G_n : A(g^{l_0}, \ldots, g^{l_d}) = 1 \right] - \Pr\left[g \xleftarrow{U} G_n : A(g^{r_0}, \ldots, g^{r_d}) = 1 \right] \right| < \frac{1}{\rho(n)}.$$

Using standard techniques found in [23] and other papers (which we have used before in the proofs of Lemmas 4.12, 4.16, and 5.22), we show that the lemma implies that $\mathcal{O}_{\mathcal{G},d}$ is an obfuscator.

Proof that Lemma 6.13 implies Theorem 6.11. Let A be an adversary and ρ be a polynomial, and we must construct a simulator S such that for every $n \in \mathbb{N}$ and every vector $\boldsymbol{r} \in \mathbb{F}_{\nu(n)}^d$,

$$\left| \Pr\left[A(\mathcal{O}_{\mathcal{G},d}(H_{\boldsymbol{r}})) = 1 \right] - \Pr\left[S^{H_{\boldsymbol{r}}}(1^n) = 1 \right] \right| < \frac{1}{\rho(n)}.$$

By Lemma 6.13, there exists a polynomial s such that for every $n \in \mathbb{N}$, there exists a set $V \subseteq \mathbb{F}_{\nu(n)}^d$ of size at most $s(n)$ such that the property in the lemma holds. Let $S^{H_{\boldsymbol{r}}}(1^n)$ be the non-uniform circuit that receives V as advice and does the following:

1: **for all** $\boldsymbol{x} \in V$ **do**
2: query the oracle on input $\boldsymbol{x}$ and record the response
3: **end for**
4: choose a vector $\boldsymbol{l} \in \mathbb{F}_{\nu(n)}^d$ such that $\forall \boldsymbol{x} \in V$, $\langle \boldsymbol{l}, \boldsymbol{x} \rangle = 0$ iff $H(\boldsymbol{x})$ accepts
5: **output** $A(\mathcal{O}_{\mathcal{G},d}(H_{\boldsymbol{l}}))$

Since $\Pr[A(\mathcal{O}_{\mathcal{G},d}(H_{\boldsymbol{r}})) = 1] = A_{\boldsymbol{r}}$ by definition and $\Pr[S^{H_{\boldsymbol{r}}}(1^n) = 1] = A_{\boldsymbol{l}}$ by construction, Lemma 6.13 ensures that S satisfies the virtual black-box condition.

It remains to show that S runs in polynomial time. The only step that could cause a problem is Step 4, so we describe a method to find $\boldsymbol{l}$ in polynomial time. First, we solve the system of linear equations $\langle \boldsymbol{x}, \boldsymbol{v} \rangle = 0$ for all $\boldsymbol{x} \in V$ such that $H(\boldsymbol{x})$ accepts. Let $\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}$ be a basis for the set of such solutions (we know $k \geq 1$ because $\boldsymbol{r}$ is a non-zero solution). Choose $\boldsymbol{l}$ by sampling uniformly at random from the space of all such solutions until a candidate is found such that $\langle \boldsymbol{l}, \boldsymbol{x} \rangle \neq 0$ for all $\boldsymbol{x} \in V$ such that $H(\boldsymbol{x})$ rejects. Since a random $\boldsymbol{l}$ satisfies these constraints with overwhelming probability, this algorithm terminates in expected polynomial time, as desired. □

Next, we provide some high-level intuition about why the lemma is true. Suppose there is an adversary A that breaks the obfuscation (and thus the lemma as well). We build a new adversary

[10] If d is a super-logarithmic function of n, we proved in Theorem 6.7 that Assumption 6.4 is false. A similar argument can be used to prove that Lemma 6.13 is false in this setting as well. Hence, while having d depend on n does not immediately seem to rule out our construction, we would need a different assumption and a new proof technique that does not go through the lemma.

A^* that runs A many times. Also, we construct two distributions $\mathcal{L}$ and $\mathcal{R}$. These distributions will be uniform over their support, so we can really just think of them as sets.

The construction of $\mathcal{L}$ and $\mathcal{R}$ proceeds iteratively, subject to two invariant conditions: first, A^* must be able to distinguish these distributions, and second, no orthogonality test should do so. These constraints together violate Assumption 6.4.

We achieve the first invariant using the negation of Lemma 6.13, which continually gives us a pair of vectors $(\boldsymbol{l_i}, \boldsymbol{r_i})$ that A (and thus A^*) can distinguish. We add $\boldsymbol{l_i}$ to the support of $\mathcal{L}$ and $\boldsymbol{r_i}$ to the support of $\mathcal{R}$.

The second invariant is achieved by continually monitoring $\mathcal{L}$ and $\mathcal{R}$ as they grow. Our goal is to "trap" any orthogonality test $\boldsymbol{x}$ once it can distinguish d of the pairs $(\boldsymbol{l_i}, \boldsymbol{r_i})$, but unfortunately there are too many such tests to list. Instead, we do the next best thing: find a basis for the space in which all of these tests live. Then, we ensure that subsequent pairs of vectors that we add to $\mathcal{L}$ and $\mathcal{R}$ are indistinguishable by these basis vectors (and as a result, any vector in the space that the basis vectors span). Therefore, any orthogonality test can only distinguish a constant number of the pairs, so by making the distributions $\mathcal{L}$ and $\mathcal{R}$ well-spread, we ensure that all orthogonality tests succeed with negligible probability. The only downside to the proof is that the "trapping" procedure requires a simulator whose runtime is exponential in d, so the proof only holds for constant dimension.

The rest of this section is devoted to a formal proof of the lemma, which uses some techniques from the proofs in [23], some novel proof concepts, and some linear algebra.

Proof that Assumption 6.4 implies Lemma 6.13. Given $\mathcal{G}$ and d, assume for the sake of contradiction that the obfuscator $\mathcal{O}_{\mathcal{G},d}$ does not satisfy Lemma 6.13. Hence, there exists an adversary A and polynomial ρ such that for all polynomials s, there exist infinitely many $n \in \mathbb{N}$ such that for every set $V \subseteq \mathbb{F}^d_{\nu(n)}$ of size at most $s(n)$, there exist vectors $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{F}^d_{\nu(n)}$ with the property that $\langle \boldsymbol{l}, \boldsymbol{x} \rangle = 0$ if and only if $\langle \boldsymbol{r}, \boldsymbol{x} \rangle = 0$ for all $\boldsymbol{x} \in V$, such that $|A_{\boldsymbol{r}} - A_{\boldsymbol{l}}| \geq \frac{1}{\rho(n)}$.

Because these probabilities are separated by a noticeable amount, an efficient algorithm is able to determine which of $A_{\boldsymbol{l}}$ and $A_{\boldsymbol{r}}$ is larger by taking n samples of each one (using independent randomness for A and the choice of $g \xleftarrow{U} G_n$ each time) and observing which sample probability is greater. By a Chernoff bound, this algorithm succeeds with overwhelming probability. Thus, from now on we assume without loss of generality that $A_{\boldsymbol{r}} > A_{\boldsymbol{l}}$ and drop the absolute value.

Given a constant c, apply this statement to the polynomial $s_c(n) = n^c$ and the resulting $n \in \mathbb{N}$ in order to build two large sets $\hat{L}^c_n$ and $\hat{R}^c_n$ iteratively as follows.

initialize $V \leftarrow \emptyset$ and $i \leftarrow 1$
while $|V| \leq n^c$ **do**
  given the set V, let $\boldsymbol{l_i}$ and $\boldsymbol{r_i}$ be vectors that violate Lemma 6.13
  insert $\boldsymbol{l_i} \in \hat{L}^c_n$ and $\boldsymbol{r_i} \in \hat{R}^c_n$
  for all subsets $T \subseteq \hat{L}^c_n \cup \hat{R}^c_n$ of size at most $d-2$ **do**
    add to V random bases of $(T \cup \{\boldsymbol{l_i}\})^\perp$ and $(T \cup \{\boldsymbol{r_i}\})^\perp$
  end for
  increment $i \leftarrow i + 1$
end while

This algorithm iteratively finds pairs of vectors that the adversary A can distinguish but the set V cannot. Then, it adds many points to V. We now describe in detail how these additional points affect future iterations of the loop.

When $T = \emptyset$ in the for loop, the algorithm adds to V a basis of vectors orthogonal to $\boldsymbol{l_i}$. Since $\boldsymbol{l_i}$ is the only vector (up to scalar multiplication) that is orthogonal to every vector in this basis, it

follows that in all future iterations $i' > i$ of the loop, $\boldsymbol{l_{i'}}$ and $\boldsymbol{r_{i'}}$ are linearly independent from $\boldsymbol{l_i}$, because $\boldsymbol{l_{i'}}$ and $\boldsymbol{r_{i'}}$ must be indistinguishable by V. The same is true for $\boldsymbol{r_i}$, so the sets $\hat{L}_n^c$ and $\hat{R}_n^c$ are continually increasing in size.

When T is not equal to the empty set, the additional points added to V ensure that orthogonality tests cannot distinguish $\hat{L}_n^c$ from $\hat{R}_n^c$. Specifically, we claim that for every vector $\boldsymbol{x} \in \mathbb{F}_{\nu(n)}^d$, there are at most d indices such that $\langle \boldsymbol{x}, \boldsymbol{l_i} \rangle = 0$ but $\langle \boldsymbol{x}, \boldsymbol{r_i} \rangle \neq 0$, or vice-versa.

To see this, suppose without loss of generality that there exists a vector $\boldsymbol{x} \in \mathbb{F}_{\nu(n)}^d$ and J indices $i_1 < i_2 < \cdots < i_J$ such that $\langle \boldsymbol{x}, \boldsymbol{l_{i_j}} \rangle = 0$ but $\langle \boldsymbol{x}, \boldsymbol{r_{i_j}} \rangle \neq 0 \ \forall j \in [J]$. We show by induction that the vectors $\boldsymbol{l_{i_1}}, \ldots, \boldsymbol{l_{i_J}}$ are linearly independent. As the base case, we showed above that any two vectors from $\hat{L}_n^c \cup \hat{R}_n^c$ are linearly independent. Now, for $j \geq 2$ suppose that $S_j = \{\boldsymbol{l_{i_1}}, \ldots, \boldsymbol{l_{i_j}}\}$ contains linearly independent vectors. At iteration i_j of the loop, a basis $\{\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}\}$ of the space $S_j^\perp$ is added to V. By definition, the basis vectors are linearly independent. If $\boldsymbol{l_{i_{j+1}}}$ were linearly dependent on S_j, say $\boldsymbol{l_{i_{j+1}}} = \alpha_1 \boldsymbol{l_{i_1}} + \cdots + \alpha_j \boldsymbol{l_{i_j}}$, then

$$\langle \boldsymbol{l_{i_{j+1}}}, \boldsymbol{b_{i'}} \rangle = \langle \alpha_1 \boldsymbol{l_{i_1}} + \cdots + \alpha_j \boldsymbol{l_{i_j}}, \boldsymbol{b_{i'}} \rangle = \alpha_1 \langle \boldsymbol{l_{i_1}}, \boldsymbol{b_{i'}} \rangle + \cdots + \alpha_j \langle \boldsymbol{l_{i_j}}, \boldsymbol{b_{i'}} \rangle = 0$$

for all $i' \in [k]$. Because $\boldsymbol{l_{i_{j+1}}}$ and $\boldsymbol{r_{i_{j+1}}}$ are indistinguishable by V, it follows that $\langle \boldsymbol{r_{i_{j+1}}}, \boldsymbol{b_{i'}} \rangle = 0$ for all $i' \in [k]$ as well, so

$$\boldsymbol{r_{i_{j+1}}} \in \overline{\{\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}\}}^\perp = (S_j^\perp)^\perp = \overline{S_j}$$

by Theorem 6.2, which means that $\boldsymbol{r_{i_{j+1}}}$ is linearly dependent on the vectors in S_j so $\langle \boldsymbol{x}, \boldsymbol{r_{i_{j+1}}} \rangle = 0$. This contradicts the assumption that $\boldsymbol{x}$ distinguishes $\boldsymbol{l_{i_{j+1}}}$ from $\boldsymbol{r_{i_{j+1}}}$, so the vectors $\boldsymbol{l_{i_1}}, \ldots, \boldsymbol{l_{i_J}}$ must be linearly independent, which completes the induction. The vectors come from a space with dimension d, so there can only be d linearly independent vectors, so $J \leq d$ as desired.

Next, we find a lower bound on the size of the sets $\hat{L}_n^c$ and $\hat{R}_n^c$. The loop condition is to stop when $|V| > n^c$. On each iteration of the loop, $|\hat{L}_n^c|$ and $|\hat{R}_n^c|$ each increase by 1 and $|V|$ increases by at most

$$2d \times \left[\sum_{k=0}^{d-2} \binom{|\hat{L}_n^c \cup \hat{R}_n^c|}{k} \right] \leq 2d^2 \binom{|\hat{L}_n^c \cup \hat{R}_n^c|}{d-2} \leq \mathrm{O}(|\hat{L}_n^c \cup \hat{R}_n^c|^{d-2}),$$

which means that the size of V is

$$|V| = \mathrm{O}(2^{d-2}) + \mathrm{O}(4^{d-2}) + \cdots + \mathrm{O}(|\hat{L}_n^c \cup \hat{R}_n^c|^{d-2}) = \mathrm{O}(|\hat{L}_n^c \cup \hat{R}_n^c|^{d-1}).$$

We also know that $|V| \leq n^c$, so it follows that $|\hat{L}_n^c|$ and $|\hat{R}_n^c|$ are $\Omega(n^{c/d})$.

Consider the $2\rho(n)$ intervals

$$\left[0, \frac{1}{2\rho(n)}\right], \ \left[\frac{1}{2\rho(n)}, \frac{1}{\rho(n)}\right], \ \ldots, \ \left[1 - \frac{1}{2\rho(n)}, 1\right]$$

that partition the unit interval. We say that an interval $[\beta, \delta]$ *separates* an $\boldsymbol{l_i}$, $\boldsymbol{r_i}$ pair if $A_{\boldsymbol{l_i}} < \beta$ and $A_{\boldsymbol{r_i}} > \delta$. Since $A_{\boldsymbol{r_i}} - A_{\boldsymbol{l_i}} > \frac{1}{\rho(n)}$, each pair is separated by at least one of the $2\rho(n)$ intervals. Hence, by the pigeonhole principle, there exists one interval that separates a $\frac{1}{2\rho(n)}$ fraction of the pairs. Call this interval $[\beta_c^*, \delta_c^*]$. Let L_n^c and R_n^c be subsets of $\hat{L}_n^c$ and $\hat{R}_n^c$, respectively, consisting only of the $\boldsymbol{l_i}$, $\boldsymbol{r_i}$ pairs that are separated by $[\beta_c^*, \delta_c^*]$. Note that $|L_n^c|$ and $|R_n^c|$ are $\Omega(\frac{n^{c/d}}{\rho(n)})$.

Furthermore, there is an algorithm A_c^* that distinguishes L_n^c from R_n^c. It is nonuniformly hard-coded with the value $\mu_c^* = \frac{1}{2}(\beta_c^* + \delta_c^*)$, and operates as follows.

Input: a vector $\boldsymbol{v} \in \mathbb{F}_{\nu(n)}^d$

1: run $A(\mathcal{O}_{\mathcal{G},d}(H_{\boldsymbol{v}}))$ a total of $32n \cdot \rho(n)^2$ times using fresh randomness for A and $\mathcal{O}$ each time
2: let τ denote the fraction of iterations that A accepts
Output: "L_n^c" if $\tau \leq \mu_c^*$ and "R_n^c" otherwise

If the input to this algorithm is a vector $\boldsymbol{l} \in L_n^c$, then we know that $A_{\boldsymbol{l}} \leq \beta_c^*$. By a Chernoff bound, the probability that the empirical acceptance rate τ is greater than $\mu_c^* = \beta_c^* + \frac{1}{4\rho(n)}$ is at most e^{-n}. The same is true for vectors in R_n^c, so this algorithm succeeds with probability $1 - e^{-n}$. On the other hand, we argued above that orthogonality tests distinguish L_n^c from R_n^c with probability at most $\frac{d}{|L_n^c|} = \mathrm{O}(\frac{\rho(n)}{n^{c/d}})$.

Finally, we construct the distributions $\mathcal{L}$ and $\mathcal{R}$ that break Assumption 6.4. Recall that the negation of Lemma 6.13 yields a function $n(c)$ as follows: for every polynomial s_c, the lemma provides some value n of the security parameter where there is a counterexample to the lemma. Furthermore, we note that as $c \to \infty$, the sequence $\{n(c)\}_{c \in \mathbb{N}} \to \infty$ as well. This is due to the fact that if $c > nd$, then the lemma considers sets V of size up to $n^{nd} > \nu(n)^d$, so the entire collection of vectors in $\mathbb{F}_{\nu(n)}^d$ can fit in V and the lemma is obviously true in this case.

We form a sort of inverse to this function as follows: given n, let c_n be the biggest value of c such that the counterexample with c applies to n. Note that c_n is not well-defined for all values of n, but it is defined for infinitely large set of values which we will denote by $N \subseteq \mathbb{N}$. It follows from the above argument that as $n \to \infty$, the sequence $\{c_n\}_{n \in N} \to \infty$ as well. Hence, there exists an infinitely large subset $N' \subseteq N$ such that $\{c_n\}_{n \in N'}$ is monotonically increasing. We form the families of distributions $\mathcal{L}$ and $\mathcal{R}$ such that $\mathcal{L}_n$ and $\mathcal{R}_n$ are uniform over the sets $L_n^{c_n}$ and $R_n^{c_n}$, respectively, for all $n \in N'$. We set $\mathcal{L}_n = \mathcal{R}_n$ arbitrarily for all $n \notin N'$.

Consider the following unified adversary A^* that is nonuniformly hardcoded with the values $\mu_{c_n}^* = \frac{1}{2}(\beta_{c_n}^* + \delta_{c_n}^*)$ for all $n \in N'$ (and arbitrarily values of $\mu_{c_n}^*$ for $n \notin N'$).

Input: a vector $\boldsymbol{v}$
1: run $A(\mathcal{O}_{\mathcal{G},d}(H_{\boldsymbol{v}}))$ a total of $32n \cdot \rho(n)^2$ times using fresh randomness for A and $\mathcal{O}$ each time
2: let τ^* denote the fraction of iterations that A accepts
Output: "L_n" if $\tau^* \leq \mu_{c_n}^*$ and "R_n" otherwise

This adversary will succeed at distinguishing $\mathcal{L}$ from $\mathcal{R}$ with overwhelming probability $1 - e^{-n}$ for all $n \in N'$ (and of course the adversary will fail on all $n \notin N'$). On the other hand, any sequence of orthogonality tests only succeeds with probability $\mathrm{O}(\frac{\rho(n)}{n^{c_n/d}})$ which is negligible since $c_n \to \infty$ as $n \to \infty$. Hence, there is no polynomial $\xi(n)$ that bounds the ratio of success probabilities for the infinitely many $n \in N'$, so Assumption 6.4 is false as desired. □

6.5 Multi-bit output

Given an obfuscator for the family of point circuits, Canetti and Dakdouk show how to construct an obfuscator for the family of point circuits with multi-bit output [25].[11] This family also accepts a single point, but instead of just having a yes or no output, it returns a hidden message on the correct input value. Such an obfuscator can be used to create a strong symmetric-key encryption scheme that satisfies leakage resilience and circular security, as we saw in Chapter 4.

Their construction applies in our case too, so we can obfuscate the family of "hyperplane testing programs with multi-bit output," with the nice property that the message is not revealed when the input is the zero vector (the one vector that is known to be in every hyperplane).

[11] See Section 3.2.1 for an explanation of the construction.

Formally, let $H_{\boldsymbol{a},m}$ be the circuit that has the vector $\boldsymbol{a} \in \mathbb{F}^d_{\nu(n)}$ hardwired, and on input a vector $\boldsymbol{x} \in \mathbb{F}^d_{\nu(n)}$, outputs m if $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0$ but $\boldsymbol{x} \neq \boldsymbol{0}$, and outputs $\perp$ otherwise. Let

$$\mathcal{M}^d_{\nu,\ell} = \{H_{\boldsymbol{a},m} : \boldsymbol{a} \in \mathbb{F}^d_{\nu(n)}, m \in \{0,1\}^{\ell(n)}\}$$

be the family of all such circuits. In particular, we can think of $\mathcal{F}^d_\nu$ as a special case of this family where $\ell = 0$ (i.e., there is only one possible message).

We define an obfuscator $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$ for the family $\mathcal{M}^d_{\nu,\ell}$ in Algorithm 6.2, given any $d \in \mathbb{N}$ and obfuscator $\mathcal{O}_{\mathcal{G},d}$ for hyperplane testing programs that is $(\ell+1)$-composable (in the sense of Definition 2.5). We emphasize that this construction is generic in the sense that it does not require the obfuscator described above in Section 6.4 but rather can use any obfuscator for $\mathcal{F}^d_\nu$ (although the only candidate that we are aware of is the construction in this work). Unfortunately, we do not know how to prove from Assumption 6.4 that our construction is even 2-composable. All we can show is that the composability of $\mathcal{O}_{\mathcal{G},d}$ is related to the length of messages that we can obfuscate.

Algorithm 6.2 Obfuscator $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$ for the family $\mathcal{M}^d_{\nu,\ell}$

Input: vector $\boldsymbol{a} \in \mathbb{F}^d_{\nu(n)}$

let $C_0 = \mathcal{O}_{\mathcal{G},d}(\boldsymbol{a})$
for $i = 1$ to ℓ **do**
  if $m_i = 1$ **then**
    set $C_i = \mathcal{O}_{\mathcal{G},d}(\boldsymbol{a})$
  else
    choose $\boldsymbol{a'} \xleftarrow{U} \mathbb{F}^d_{\nu(n)}$ and set $C_i = \mathcal{O}_{\mathcal{G},d}(\boldsymbol{a'})$
  end if
end for

Output: circuit that hardwires $C_0, \ldots, C_\ell$ and operates as follows on input $\boldsymbol{x} \in \mathbb{F}^d_{\nu(n)}$: output $\perp$ if $C_0(\boldsymbol{x})$ rejects or if $\boldsymbol{x} = \boldsymbol{0}$, otherwise output the string s formed by $s_i = C_i(\boldsymbol{x})$ for $i = 1, \ldots, \ell$

Theorem 6.14. *Suppose that the obfuscator $\mathcal{O}_{\mathcal{G},d}$ is $(\ell+1)$-composable for some $\ell = \mathrm{poly}(n)$, and let $\nu(n) = |G_n|$. Then, $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$ is an obfuscator for $\mathcal{M}^d_{\nu,\ell}$ with approximate functionality.*

Proof. The proof of this theorem is similar to the one in [25]. First, we prove the approximate functionality of $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$. Note that the construction has two parts: first, circuit C_0 is used to determine which inputs should be accepted, and then circuits C_1, ..., C_ℓ are used to determine the message. By the exact functionality of $\mathcal{O}_{\mathcal{G},d}$, circuit C_0 always accepts or rejects inputs accurately, and the same is true for all of the circuits that encode a message bit of 1.

However, the computed message may be inaccurate because the encoding of 0 is a randomly chosen hyperplane testing program, which is supposed to reject the input but may end up accepting it. A randomly chosen circuit accepts a $\frac{1}{\nu(n)}$ fraction of its inputs, so all of the circuits representing a 0 value reject the input with probability at least $(1 - \frac{1}{\nu(n)})^{\ell(n)}$, which is overwhelming because ν is exponential in n and ℓ is only polynomial in n.

The polynomial slowdown of $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$ is clear from the construction and the corresponding properties of $\mathcal{O}_{\mathcal{G},d}$. It remains to prove the virtual black-box property. For $i = 1$ to ℓ, let $\boldsymbol{a_i}$ be the vector such that $\boldsymbol{a_i} = \boldsymbol{a}$ if $m_i = 1$ or $\boldsymbol{a_i}$ is uniformly chosen otherwise. By the $(\ell+1)$-composable virtual black-box property, we know that there exists a simulator S such that the output of

$$A(\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}(\boldsymbol{a})) = A(\mathcal{O}_{\mathcal{G},d}(\boldsymbol{a_1}), \ldots, \mathcal{O}_{\mathcal{G},d}(\boldsymbol{a_d}))$$

can be simulated by $S^{H_{\boldsymbol{a_1}},\ldots,H_{\boldsymbol{a_d}}}$. Furthermore, the oracles $H_{\boldsymbol{a_1}},\ldots,H_{\boldsymbol{a_d}}$ can be simulated by the oracle $H_{\boldsymbol{a},m}$ up to a negligible simulation error in the following manner: if $H_{\boldsymbol{a},m}(\boldsymbol{x}) = \perp$, then we say that $H_{\boldsymbol{a_i}}(\boldsymbol{x}) = 0$ for all i. Otherwise $H_{\boldsymbol{a},m}(\boldsymbol{x}) = m$, in which case we say that $H_{\boldsymbol{a_i}}(\boldsymbol{x}) = m_i$. Hence, the simulator $T^{H_{\boldsymbol{a},m}}$ that runs $S^{H_{\boldsymbol{a_1}},\ldots,H_{\boldsymbol{a_d}}}$ and emulates the oracle queries in this manner satisfies the virtual black-box property for $\tilde{\mathcal{O}}_{\mathcal{G},\ell,d}$. □

6.6 One-time signature scheme

We can use an obfuscator for the family of planes in three-dimensional space to form a one-time signature scheme. Informally, the secret and public keys are a hidden plane and an obfuscation of the plane membership testing program, respectively. A signature of a message is a point on the plane that is related to the message, and the verification procedure runs the obfuscated plane testing circuit to verify signatures.

More formally, let ν be a prime sequence, and $\mathcal{O}$ be an obfuscator for the family of planes over $\mathbb{F}_\nu^3$ (such as the one in Section 6.4). Consider the following three algorithms.

KeyGen(1^n): Choose field elements $sk_1, sk_2, c \xleftarrow{U} \mathbb{F}_{\nu(n)} \setminus 0$. Form the vector $\boldsymbol{sk} = (sk_1, sk_2, 1)$ and the obfuscated plane $P = \mathcal{O}(H_{\boldsymbol{sk}})$. The secret key is (sk_1, sk_2), and the public key is (P, c).

Sign($m \in \mathbb{F}_{\nu(n)}$): Let σ_2 be the unique field element such that the inner product $\langle \boldsymbol{sk}, (cm, \sigma_2, 1)\rangle$ equals zero. The signature is (cm, σ_2).

Verify($m, (\sigma_1, \sigma_2)$): Accept if and only if $\sigma_1 = cm$ and $P(\sigma_1, \sigma_2, 1)$ accepts.

This signature scheme is unforgeable in a weak sense, described in [56] and other works, in which the forger must choose the message on which she requests a signature before being shown the public key. (However, she does not need to commit to the message whose signature she intends to forge.) A more formal statement of this model follows.

Definition 6.15. A one-time signature scheme (KeyGen, Sign, Verify) satisfies *weak $l(n)$-resistant unforgeability* if for all forgers $F = (F_1, F_2)$, the probability that F_2 forges a signature is negligible, where the forger plays the following game:

1. $F_1(1^n)$ outputs a message m and leakage circuit L with output length at most $l(n)$.
2. We obtain a key pair $(sk, pk) \leftarrow \text{KeyGen}(1^n)$. Then, F_2 receives as input the message m, a signature $\sigma = \text{Sign}(m)$, the public key, and the leakage $L(sk)$, and F_2 must output a valid message-signature pair that is not (m, σ).[12]

The techniques in [56] allow us to transform this scheme into one that is existentially unforgeable under chosen message attacks (the standard security notion for signature schemes). The transformation preserves leakage resilience, and it requires a chameleon hash function whose seed can be chosen with public coins [60]. Such a function can be constructed assuming the hardness of computing discrete logarithms (Assumption 5.13), and the construction is very similar to the one-way function described in Section 5.3.2.

Theorem 6.16. *Let ν be a prime sequence, and $\mathcal{O}$ be an obfuscator for the family of hyperplane testing programs over the vector space $\mathbb{F}_\nu^3$. Then, the above algorithm leads to an existentially*

[12] In general, F_2 is allowed to find a new signature for the same message m. However, in our construction, the signature is uniquely determined by m and the public key, so the forger must sign a different message.

unforgeable one-time signature scheme under chosen message attacks. Furthermore, the signature scheme is resilient to any leakage function whose output length is bounded by

$$l(n) = n - \omega(\log(n)).$$

In particular, leakage of $l(n) = \gamma n$ for any $\gamma < 1$ is permitted.

We make several remarks about this theorem and then prove it.

1. The secret key consists of two elements of $\mathbb{F}_{\nu(n)}$, so it is $2n$ bits long. Thus, our signature scheme permits leakage of up to half of the length of the secret key. This matches the leakage bound attained by Katz and Vaikuntanathan [58] for schemes that do not use general non-interactive zero-knowledge proofs (albeit under a much stronger assumption).

2. The leakage bound in the theorem is tight. Consider the following leakage function that has a message m hardcoded: use the secret key to form a signature associated to m, and output σ_2. This leakage function has n bits of output, and permits a forgery of the message m by the signature (cm, σ_2).

3. There are two slightly different concepts of "leakage" that one can consider: either the leakage function can depend on the secret key, or on the randomness used in the key generation algorithm. Both concepts of leakage are discussed in [58]. The latter concept is stronger because the secret key can be derived from the randomness. The converse is not always true, but it is true for our signature scheme. Therefore, our scheme trivially satisfies the stronger notion of leakage.

4. This scheme is rather efficient. Signing requires just two modular multiplications over $\mathbb{F}_{\nu(n)}$, which translates to approximately 6 multiplications over $2n$-bit integers using the Montgomery method [69]. Verification is slower, and its speed depends upon the implementation of $\mathcal{O}$. Using our construction in Section 6.4, two exponentiations and three multiplications over G_n are required.

Proof. It is proved in [56, Appendix A] that the existential unforgeability of the final scheme reduces to the weak unforgeability of the original scheme. While their proof applies to multiple-use signature schemes, is straightforward to check that the reduction preserves the number of signatures required, so it applies to one-time signatures as well. Additionally, their proof preserves leakage resilience because the transformation only adds a chameleon hash function that is public coins, and the trapdoor information to the hash is not stored in the secret key during honest usage of the signature scheme.

Therefore, it suffices to prove the weak unforgeability of the construction above. Suppose for the sake of contradiction that there exists a function $l(n) \in n - \omega(\log(n))$ and a forger $F = (F_1, F_2)$ that breaks weak $l(n)$-resilient unforgeability. We will use the forger to construct an adversary A that breaks the virtual black-box property of $\mathcal{O}$.

When given a random tape r, $F_1(1^n; r)$ outputs a message m and leakage function L. We know that the forger succeeds with noticeable probability, which means that there exists a fixed value r^* of the random tape of F_1 for which F succeeds with noticeable probability. We only consider $r = r^*$ from now on, and the message m and leakage function L that result from r^*.

We construct several adversaries $A_{\boldsymbol{\sigma},s,\pi}$ that are parametrized by a vector $\boldsymbol{\sigma} \in \mathbb{F}_\nu^3$, a string $s \in \{0,1\}^{l(n)}$, and a binary predicate π. In addition, all of the adversaries are hardcoded with m. It suffices to prove that there exists one such adversary that breaks the virtual black-box property.

The adversary $A_{\boldsymbol{\sigma},s,\pi}$ receives an obfuscated program $P = \mathcal{O}(H_{\boldsymbol{a}})$ as input for an unknown plane $\boldsymbol{a} = (a_1, a_2, 1)$. The adversary interacts with the forger in a way such that $\boldsymbol{a}$ is consistent with the secret key in the signature scheme. We describe this interaction in detail.

1. $A_{\boldsymbol{\sigma},s,\pi}$ computes $c = \frac{\sigma_1}{m}$. It then considers (P, c) to be the public key of the signature scheme and s to be the leakage on the secret key, so it runs $F_2(m, \boldsymbol{\sigma}, (P, c), s)$.

2. Eventually F_2 outputs a forgery $(m', \boldsymbol{\sigma'})$. Then, $A_{\boldsymbol{\sigma},s,\pi}$ can find the hidden plane $\boldsymbol{a}$ by solving the linear system $\langle \boldsymbol{a}, \boldsymbol{\sigma} \rangle = 0$ and $\langle \boldsymbol{a}, \boldsymbol{\sigma'} \rangle = 0$. Finally, the adversary outputs $\pi(\boldsymbol{a})$.

We now analyze the success probability of $A_{\boldsymbol{\sigma},s,\pi}$. It is useful in the analysis to consider the distribution $\mathcal{D}_{\boldsymbol{\sigma},s}$ that is uniform over all vectors $\boldsymbol{v} \in \mathbb{F}_\nu^3$ such that $\langle \boldsymbol{v}, \boldsymbol{\sigma} \rangle = 0$ and $L(\boldsymbol{v}) = s$. Recall that the forger F initially believes that the secret key $\boldsymbol{a}$ is initially chosen from the uniform distribution, and it is later told that $\langle \boldsymbol{a}, \boldsymbol{\sigma} \rangle = 0$ and $L(\boldsymbol{a}) = s$, so from F's point of view the secret key is chosen from the distribution $\mathcal{D}_{\boldsymbol{\sigma},s}$.

We compute the entropy of distribution $\mathcal{D}_{\boldsymbol{\sigma},s}$ for a random choice of $\boldsymbol{\sigma}$ and s. Note that the secret key $\boldsymbol{a}$ initially has $2n$ bits of entropy, and the leakage function L fixes $n - \omega(\log(n))$ bits of entropy. Furthermore, $\boldsymbol{\sigma}$ leaks at most n bits of information about $\boldsymbol{a}$, because σ_1 can be chosen uniformly and independently of $\boldsymbol{a}$, and then σ_2 is chosen to be the unique element of $\mathbb{F}_{\nu(n)}$ such that $\langle \boldsymbol{a}, \boldsymbol{\sigma} \rangle = 0$ so only σ_2 depends on $\boldsymbol{a}$. As a result, there exists some $\boldsymbol{\sigma^*}$ and s^* such that the distribution $\mathcal{D}_{\boldsymbol{\sigma^*},s^*}$ has at least $\omega(\log(n))$ bits of min-entropy.

Next, we form a polynomial time computable predicate π^* that equals 1 with probability $\frac{1}{2}$ when given a plane from distribution $\mathcal{D}_{\boldsymbol{\sigma^*},s^*}$. To do so, we list all of the planes in the support of $\mathcal{D}_{\boldsymbol{\sigma^*},s^*}$ lexicographically. Then, π^* that hardcodes the plane in the middle of this list, and tests whether its input comes lexicographically after the hardcoded plane.

We showed that for all planes $\boldsymbol{a}$ in the support of $\mathcal{D}_{\boldsymbol{\sigma^*},s^*}$, the adversary $A_{\boldsymbol{\sigma^*},s^*,\pi^*}$ is able to compute $\boldsymbol{a}$ and thus $\pi^*(\boldsymbol{a})$. On the other hand, we claim that any PPT simulator $S^{H_{\boldsymbol{a}}}$ is unable to find the hidden plane $\boldsymbol{a}$ when it is chosen from distribution $\mathcal{D}_{\boldsymbol{\sigma^*},s^*}$. The simulator knows that $\boldsymbol{\sigma^*}$ is on the plane, but it cannot find another point on the plane because it takes n bits of information to describe a point and the leakage is limited in length to $n - \omega(\log(n))$ bits of output. Thus, information-theoretically there are at least $2^{\omega(\log(n))}$ points consistent with the leakage, and the simulator does not have time to query all of them. Furthermore, unless it happens to query the correct point, it cannot learn any more information about $\boldsymbol{a}$, since querying one more accepted point reveals all of $\boldsymbol{a}$. Thus, S is unable to compute $\pi^*(\boldsymbol{a})$ with probability noticeably different from $\frac{1}{2}$.

Therefore, we have shown that $A_{\boldsymbol{\sigma^*},s^*,\pi^*}(H_{\boldsymbol{a}})$ outputs $\pi^*(\boldsymbol{a})$ when $\boldsymbol{a}$ is chosen from the well-spread distribution $D_{\boldsymbol{\sigma^*},s^*}$, whereas no simulator $S^{H_{\boldsymbol{a}}}$ can do so. This breaks the virtual black-box property, as desired. □

Chapter 7

Future work

Obfuscation is an intriguing problem with the ability to solve many outstanding cryptographic problems, as well as providing benefits to the computer science community at large. The field is still in its infancy, however. While there are many positive results with useful applications, much more research is required before obfuscation can be used generally to provide rigorous security guarantees against reverse engineering and tampering for arbitrary programs (such as the example mentioned at the beginning of the thesis concerning an RSA-breaking program that uses a hidden factoring algorithm as a subroutine).

As a result, the goal of this thesis is two-fold: find new results that advance our knowledge on obfuscation, and describe the state of the field in order to encourage future work on the problem. Chapter 3 surveyed the known results so far, and in this chapter, we examine some of the possible future directions of the field. First, we discuss three "holy grail" problems that, if solved, could have enormous benefits for cryptography as a whole. Then, we discuss some (hopefully more manageable) extensions of the three specific results in this thesis.

Complexity classes. Positive results in obfuscation have only been found for circuit families that perform a specific task, such as point circuits and their extensions. If an obfuscator were constructed for a complexity class instead, then it may have more applications. The general impossibility results from Section 3.3.1 are greatly troubling here, as they show the impossibility of obfuscating low-complexity classes like TC^0 (for virtual black-box security) or AC^0 (for best-possible security). At the other extreme, incredibly low complexity classes are learnable, and thus trivially obfuscatable but uninteresting (Theorem 3.4). Even so, finding a generic obfuscator for a complexity class in between these bounds would be very useful as it would include many interesting cryptographic functionalities [7].

Connection to homomorphic encryption. A recent breakthrough in cryptography is in the study of (fully) homomorphic encryption, which allows messages to be manipulated in a meaningful way while in a hidden state. The illuminating work of Gentry [41] shows that this problem is feasible, and subsequent work has reduced the assumptions required [89]. The relationship between this problem and obfuscation seems promising but is not fully understood. If an obfuscator existed for the "decrypt, add or multiply, then re-encrypt" functionality, then it could be used to construct a new homomorphic encryption scheme. In the other direction, homomorphic encryption intuitively seems very helpful in obfuscation: one can obfuscate a circuit by "encrypting" it under the fully homomorphic encryption scheme and running the circuit on ciphertexts. However, this naïve plan fails because the result is an encryption of the output, not the output itself. More research is

required to see if homomorphic encryption can help program obfuscation (perhaps simply in the restricted setting of low-complexity circuits, as described above).

Cryptographic hash functions. One question of interest to many cryptographers is the design of secure hash functions. This is a topical issue, as NIST is currently running a competition to choose a new hash function standard [78, 80]. One challenging issue regarding the construction of a secure hash function is to determine the properties that it must satisfy. Clearly one-wayness, collision-resistance, and second-preimage resistance are a good start, but there are applications of hash functions in practice that require stronger properties. For instance, the login program application described in Section 1.2.4 requires that one-wayness hold when the input is chosen from a structured distribution, rather than the uniform one.

The NIST competition's request for comments initially stated that candidates would be judged on "the extent to which the algorithm output is indistinguishable from a random oracle [72]." On the face of it, this criterion does not quite make sense, as any concrete, instantiated algorithm is trivially distinguishable from a random oracle [29]. On the other hand, obfuscation provides a framework in which this statement does make sense. A pseudorandom function ensemble has the property that it cannot be distinguished from a random oracle in a black-box way. Therefore, the same must be true for an adversary that receives the code of an obfuscated pseudorandom function. As a result, obfuscation of pseudorandom functions would be of enormous value. This goal is sadly unachievable under the average-case security definition (Theorem 3.15), but it remains feasible under the virtual black-box property.

Obfuscation and encryption. In Chapter 4, we connected obfuscation to the problem of symmetric key encryption with leakage resilience and key-dependent message security. Subsequent to our work, Bitanksi and Canetti [12] the connection to public key encryption schemes, and to the multiple KDM setting in which an adversary receives the encryption of multiple messages under multiple keys that may be arbitrarily correlated. Future research along these lines may find new relationships between the two problems, or find more implications of the connection (such as in Section 4.6).

Non-malleable obfuscation. In Chapter 4, we added two anti-tampering guarantees to obfuscated programs. However, we are only able to construct obfuscators that satisfy the stronger requirements in models with trusted setup. Finding such a construction in the standard model would have practical uses in settings such as the login program example described in Section 1.3.2. Additionally, our proof of security in the common reference string model only applies to relations of a certain type. While relations of this form suffices for many applications, proving the security of this proof against all relations (or finding another construction that achieves this goal) would be helpful as well.

Hyperplane membership testing. In Chapter 4, we constructed an obfuscator for a new family of programs. There are several directions of improvement here. First, we can only prove the security of our construction for constant dimension d. While this suffices for applications such as the digital signature scheme, it is still an open problem to determine the security of our construction when d depends on the security parameter. (By Theorem 6.7, our assumption and proof technique are invalid if $d = \omega(\log(n))$, so novel techniques are required.) A second line of research is to explore the properties of our construction such as its composability or non-malleability. Third, we wish to find more applications for this obfuscator.

List of Symbols

$\leftarrow$ The assignment operator. Given a variable x and distribution D, $x \leftarrow D$ means that one should take a sample from D and assign it to x. Sometimes we abuse notation and write $x \leftarrow S$ where S is a set, in which case we use the uniform distribution on S. (We often write $x \xleftarrow{U} S$ to emphasize the choice of the uniform distribution.) Abusing notation even further, if x and y are both variables, the notation $x \leftarrow y$ is used to denote $x \leftarrow \{y\}$; that is, we set x to the value of y.

$\equiv$ Boolean operation that tests whether two circuits are functionally equivalent. See Definition 2.3 on page 31.

$\approx$ Denotes two probabilities that only differ by a negligible probability.

$\#$ An operation to combine two Turing machines into one. See Theorem 3.29 on page 55.

$[k]$ The set of integers from 1 to k. That is, $[k] = \{1, 2, \ldots, k\}$.

0^n, 1^n A string consisting of n zeroes (or ones). We emphasize that this is *not* the exponentiation operation. Conversely, when 2^n is used throughout the thesis, it does mean 2 to the n^{th} power. Hopefully, the overloaded operator is not confusing as it never makes sense to exponentiate 0 or 1.

$\emptyset$ Either denotes the empty set or the empty string, depending on context.

$\circ$ The string concatenation operator.

α A function bounding the min-entropy that a key distribution must satisfy in order for a symmetric encryption scheme or digital locker to remain secure. See Section 1.3.1 for an informal overview and Definition 4.1 on page 62 for technical details.

Γ, Φ, Ψ Obfuscated circuits constructed in Chapter 5. See Algorithms 5.1, 5.4, and 5.5.

γ Denotes a real number in the interval $[0, 1]$, typically used as a probability.

ε Either denotes a negligible function or a small constant. The latter use only occurs in Section 4.6.

π Depending on the context, either denotes a binary predicate that an adversary or simulator is attempting to learn, or a permutation. The first use occurs in security definitions like Definition 2.4 on page 33. The second use is in the constructions of Sections 3.2.1 and 5.4.

ν A prime sequence. See Definition 6.3 on page 128.

ξ, ρ	A polynomial, possibly multi-variate.
A, A'	An efficient (PPT) adversary.
$\boldsymbol{a}$, $\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{v}$	A vector in a d-dimensional vector space. Can be used to represent a hyperplane through the origin in the vector space. See Section 1.3.3 and Chapter 6.
C, D	A circuit, often with n bits of input and binary output, but not always.
$\mathcal{C}$, $\mathcal{D}$	A family of programs, formally represented as circuits. Often, $\mathcal{C}$ denotes a family of circuits to be obfuscated, and $\mathcal{D}$ denotes the resulting family of obfuscated circuits. See Definition 2.1 and Sections 1.2.3 and 5.2.
c	In Section 1.3.1 and Chapter 4, denotes a ciphertext. In the rest of the thesis, typically denotes a constant. Often used as the degree of a polynomial (i.e., n^c).
d	Dimension of the vector space under consideration in Chapter 6.
$\mathrm{Dec}_k(c)$	Decryption routine of a symmetric encryption scheme. Receives key k and ciphertext c, and outputs a decrypted message. See Sections 1.3.1 and 4.2.2.
E	A polynomial-time computable relation with two input strings (which are typically representations of circuits) and binary output. See Definitions 5.1 and 5.5.
$\mathrm{Enc}_k(m)$	Encryption routine of a symmetric key encryption scheme. Receives key k and message m, and outputs a ciphertext. See Sections 1.3.1 and 4.2.2.
$\mathbb{F}_p$	A finite field of order p. In this thesis, we only consider fields of prime order, in which case $\mathbb{F}_p \cong \frac{\mathbb{Z}}{p\mathbb{Z}}$. See Sections 5.3.2 and 6.2.
$\mathcal{F}_\nu^d$	The family of d-dimensional hyperplanes over finite fields whose orders are specified by ν. See Section 6.4.
f	Denotes a function.
G, G_n	A group whose order is typically a prime in the range $(2^{n-1}, 2^n]$. See Assumption 6.4 on page 128.
$\mathbb{G}$	A PPT algorithm that samples a prime p, group G for which it is hard to compute discrete logs, and generator $g \in G$. See Assumption 5.13 on page 107.
$\mathcal{G}$	A family of groups of increasing order. See Assumption 6.4 on page 128.
g	A group element. Often, we want g to be a generator of the group. We typically work over groups of prime order, so this condition simply means $g \neq 1$.
$H_{\boldsymbol{a}}(\boldsymbol{x})$	Circuit that performs hyperplane membership testing. It accepts if $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0$. See Section 1.3.3 and Chapter 6.
H_∞	The min-entropy function. Receives a distribution as input, and outputs a real number. See Definition 2.2.
$\mathcal{H}$	A family of pairwise-independent hash permutations. See Lemma 4.7 on page 66.

$I_{\{w_1,\ldots,w_m\}}$ An "indicator" circuit that stores the strings $w_1, \ldots, w_m$ in a readily identifiable manner. It accepts the w_i and rejects all other inputs. Note that the set can be empty: $I_\emptyset$ is a circuit that rejects all inputs. Also, when only one input is accepted, the set notation is dropped and the circuit is denoted I_w. The family of all such circuits is denoted by $\mathcal{P}^m$. See Section 2.1.

$I_{(k,m)}$ A point circuit with multi-bit output. If its input equals k, then it outputs m; otherwise, it outputs a failure message $\perp$. Note that this functionality is *not* the same as $I_{\{k,m\}}$! See Sections 2.1 and 4.2.1.

$\mathcal{I}$ The family of point circuits with multi-bit output (that is, of all possible $I_{(k,m)}$). See Sections 2.1 and 4.2.1.

k Key in an encryption scheme or digital locker. See Sections 1.3.1 and 4.2.

$\mathcal{L}$, $\mathcal{R}$ Denotes a family of distributions of hyperplanes. More formally, $\mathcal{L} = \{\mathcal{L}_n\}_{n\in\mathbb{N}}$, where each $\mathcal{L}_n$ is a distribution of vectors (which are used to identify hyperplanes through the origin) in the finite vector space $\mathbb{F}^d_{\nu(n)}$. (Note that $\mathcal{L}$ thus implicitly depends on the prime sequence ν.) See Assumption 6.4 on page 128.

$\mathcal{M}^d_{\nu,\ell}$ Family of hyperplane testing circuits with multi-bit output with dimension d, length ℓ, and prime sequence ν. See Section 6.5.

ℓ The length of a message m in a point circuit with multi-bit output in Chapter 4 and Section 6.5. See Definition 4.1 on page 62.

m Either denotes the message in an encryption scheme or digital locker, or the number of points accepted by a multi-point circuit. The first use occurs in Chapters 4 and 6, and the second use occurs in Chapter 5.

$\mathbb{N}$ The set of natural numbers (i.e., positive integers).

n Either denotes the security parameter in the cryptographic protocol under consideration, or the input length of a given circuit. Note that these two values are often the same: the input length of a circuit also equals the security parameter used when obfuscating the circuit. See Chapter 2.

$\mathrm{negl}(n)$ Shorthand for "there exists a function that is negligible in n such that" the mathematical expression holds.

$\mathcal{O}$ Generic circuit that performs obfuscation. Receives a random string and (the description of) a circuit as input, and outputs a circuit. Specific constructions typically have a subscript to denote the family that they obfuscate; for example, $\mathcal{O}_{\mathcal{P}^1}$ is an obfuscator for point circuits. See Definition 2.4 on page 33.

P, Q Typically denotes a program, formally represented as a circuit. Occasionally used in Section 1.3.3 to denote a hyperplane.

$\mathcal{P}^m$, $\mathcal{P}^{m+}$ Families of circuits that accept at most m points. The family $\mathcal{P}^m$ includes the circuit $I_\emptyset$ that rejects all inputs, whereas $\mathcal{P}^{m+}$ does not. See Sections 2.1, 5.3, and 5.4.

p A prime number that is n (the security parameter) bits long. See Assumption 3.20 on page 45

q — In Chapter 5 and Section 6.3.2, denotes a random oracle query. In the rest of the thesis, used as a second prime number in addition to p when needed, such as in Assumption 3.20 on page 45.

$\text{poly}(n)$ — Shorthand for "there exists a polynomial in n such that" the mathematical expression holds.

R — A random oracle. In this thesis, we mostly consider random permutations, although Chapter 3 and Section 5.3.2 use random functions. See Definitions 5.7 and 5.8 on page 98.

r — A string that is chosen uniformly at random. Constructions of obfuscators in the random oracle model use this string as a "salt" when hashing an input using the random oracle. See Sections 5.3 and 5.4.

S — Typically denotes an efficient (PPT) simulator. Occasionally used to denote a set.

s — In Chapters 4 and 6, a polynomial that upper bounds the size of a circuit or set. See Definition 2.1 and Lemma 6.13. In Chapter 5, used in the proofs of security for the various constructions to denote a special "source" input to the random oracle that would yield the target t (as opposed to the variable q, which is used to denote any input to the oracle).

Σ — A common reference string. See Definition 5.9 on page 98.

σ — A digital signature. See Sections 5.4 and 6.6.

t — In Chapter 4, denotes the composability of an obfuscator; that is, the maximum number of obfuscated circuits that an adversary can receive and still have security hold. See Definition 2.5 on page 34. In Chapter 5, denotes an output of a random oracle that is a special "target." See Algorithms 5.1, 5.4, and 5.5.

U — The uniform distribution on a given set (depending on the context). For example, U_n denotes the uniform distribution on all strings of length n; that is, on the set $\{0,1\}^n$.

V — A set of vectors. See Sections 6.2 through 6.4.

w — A password, represented as a string. See Sections 1.3.2 and 2.1.

x, $\boldsymbol{x}$ — Typically denotes the input to a circuit. The latter notation is used when the input is a vector. See Sections 2.1 and 6.1.

$\mathbb{Z}$ — The ring of integers.

z — Auxiliary input to an adversary attempting to break obfuscation or encryption. Typically regarded as a function in Chapter 4 and a string in Chapter 5. See Definition 4.24 for an example of the function usage, and Definitions 2.4, 5.1, and 5.5 for the string usage.

List of Algorithms

List of Figures

Bibliography

[1] Ben Adida and Douglas Wikström. How to shuffle in public. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 2007.

[2] Divesh Aggarwal and Ueli M. Maurer. Breaking RSA generically is equivalent to factoring. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2009.

[3] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, 2009.

[4] Kristina Altmann, Tibor Jager, and Andy Rupp. On black-box ring extraction and integer factorization. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 437–448. Springer, 2008.

[5] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2009.

[6] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[7] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[8] Michael Backes, Markus Dürmuth, and Dominique Unruh. OAEP is secure under key-dependent messages. In *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 506–523. Springer, 2008.

[9] Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks - brsim/uc-soundness of symbolic encryption with key cycles. In *CSF*, pages 112–124. IEEE Computer Society, 2007.

[10] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[11] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2004.

[12] Nir Bitansky and Ran Canetti. On obfuscation with strong simulators. Cryptology ePrint Archive, 2010. `http://eprint.iacr.org/`.

[13] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.

[14] Dan Boneh. The Decision Diffie-Hellman problem. In *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.

[15] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[16] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[17] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[18] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from Decision Diffie-Hellman. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2008.

[19] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 1996.

[20] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, 35:119–152, 2002.

[21] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[22] Jan Camenisch, Nishanth Chandran, and Victor Shoup. A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2009.

[23] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1997.

[24] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science*, pages 136–145, 2001.

[25] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 489–508. Springer, 2008.

[26] Ran Canetti, Yael Kalai, Mayank Varia, and Daniel Wichs. On symmetric encryption and point obfuscation. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 2010.

[27] Ran Canetti, Yael Kalai, Mayank Varia, and Daniel Wichs. On symmetric encryption and point obfuscation. Cryptology ePrint Archive, Report 2010/049, 2010. `http://eprint.iacr.org`.

[28] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 131–140, 1998.

[29] Ran Canetti, Ron Rivest, and Eran Tromer. Comments on NIST draft requirements and criteria for hash algorithm. `http://people.csail.mit.edu/tromer/papers/hash-draft-comments.pdf`.

[30] Ran Canetti, Guy Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2010.

[31] Ran Canetti and Mayank Varia. Non-malleable obfuscation. Cryptology ePrint Archive, Report 2008/495, 2008. `http://eprint.iacr.org`.

[32] Ran Canetti and Mayank Varia. Non-malleable obfuscation. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2009.

[33] Martin Cochran. Notes on the Wang *et al.* 2^{63} SHA-1 differential path. Cryptology ePrint Archive, Report 2007/474, 2007. `http://eprint.iacr.org/`.

[34] Obfuscated code. Wikipedia. `http://en.wikipedia.org/wiki/Obfuscated_code#Step_by_step`, accessed 17 June 2010.

[35] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2002.

[36] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 2002.

[37] Alexander W. Dent. The hardness of the DHK problem in the generic group model. Cryptology ePrint Archive, Report 2006/156, 2006. `http://eprint.iacr.org/`.

[38] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630. ACM, 2009.

[39] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *STOC*, pages 654–663. ACM, 2005.

[40] Marc Fischlin. A note on security proofs in the generic model. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2000.

[41] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[42] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32. ACM, 1989.

[43] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562. IEEE Computer Society, 2005.

[44] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[45] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[46] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2007.

[47] Satoshi Hada. Zero-knowledge and code obfuscation. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 443–457. Springer, 2000.

[48] Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 92–112. Springer, 2010.

[49] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1998.

[50] Iftach Haitner and Thomas Holenstein. On the (im)possibility of key dependent encryption. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, 2009.

[51] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[52] Shai Halevi and Hugo Krawczyk. Security under key-dependent inputs. In *ACM Conference on Computer and Communications Security*, pages 466–475. ACM, 2007.

[53] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 214–232. Springer, 2007.

[54] Dennis Hofheinz and Dominique Unruh. Towards key-dependent message security in the standard model. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 108–126. Springer, 2008.

[55] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 2007.

[56] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer, 2009.

[57] Tibor Jager and Jörg Schwenk. On the equivalence of generic group models. In *ProvSec*, volume 5324 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2008.

[58] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 703–720. Springer, 2009.

[59] Neal Koblitz and Alfred J. Menezes. Another look at generic groups. In *Advances in Mathematics of Communications*, pages 13–28, 2006.

[60] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS*. The Internet Society, 2000.

[61] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[62] Gregor Leander and Andy Rupp. On the equivalence of RSA and factoring regarding generic ring algorithms. In *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 241–251. Springer, 2006.

[63] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2004.

[64] Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer, 1994.

[65] Ueli M. Maurer. Abstract models of computation in cryptography. In *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[66] Ueli M. Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. In *EUROCRYPT*, pages 72–84, 1998.

[67] Daniele Micciancio. The RSA group is pseudo-free. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 387–403. Springer, 2005.

[68] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *Journal of Computer Security*, 12(1):99–130, 2004.

[69] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[70] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. Comput. Syst. Sci.*, 58(2):336–375, 1999.

[71] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.

[72] National Institute of Standards and Technology. Request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register Notice, Vol. 72, No. 212. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[73] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[74] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *EUROCRYPT*, pages 224–314, 1984.

[75] Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.

[76] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[77] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196. ACM, 2008.

[78] Bart Preneel. The first 30 years of cryptographic hash functions and the NIST SHA-3 competition. In *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2010.

[79] Michael Rabin. Digitalized signatures and public-key functions as intractable as factorization. MIT Laboratory for Computer Science, 1979.

[80] Christian Rechberger, Vincent Rijmen, and Nicolas Sklavos. The NIST cryptographic workshop on hash functions. *IEEE Security & Privacy*, 4(1):54–56, 2006.

[81] Ronald L. Rivest. On the notion of pseudo-free groups. In *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 505–521. Springer, 2004.

[82] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[83] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[84] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer, 2001.

[85] Claus-Peter Schnorr and Markus Jakobsson. Security of signed elgamal encryption. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2000.

[86] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2009.

[87] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[88] Nigel P. Smart. The exact security of ECIES in the generic group model. In *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2001.

[89] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[90] Daqing Wan. Factoring multivariate polynomials over large finite fields. *Mathematics of Computation*, 54:755–770, 1990.

[91] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[92] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.

[93] Hoeteck Wee. On obfuscating point functions. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 523–532, 2005.