Foundations of Garbled Circuits

By

Viet Tung Hoang
B.S. (National University of Singapore) 2007

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Phillip Rogaway (Chair)

Mihir Bellare

Matthew Franklin

Committee in Charge
2013

Contents

Viet Tung Hoang
June 2013
Computer Science

Foundations of Garbled Circuits

Abstract

Garbled circuits, a classical idea rooted in the work of Andrew Yao, have long been understood as a cryptographic *technique*, not a cryptographic *goal*. Here we cull out a primitive corresponding to this technique. We call it a *garbling scheme*. We provide a provable-security treatment for garbling schemes, endowing them with a versatile syntax and multiple security definitions. The most basic of these, *privacy*, suffices for two-party secure function evaluation (SFE) and private function evaluation (PFE). We next consider *obliviousness* and *authenticity*, properties needed for private and verifiable outsourcing of computation. Starting from a PRF, we give efficient schemes to achieve all security notions above, and analyze their concrete security. Our treatment of garbling schemes provides ground for more efficient garbling, more rigorous analyses, and more modularly designed higher-level protocols.

On the practical side, we provide *extremely efficient* garbling schemes based on fixed-key AES. We justify the security of these methods in the random-permutation model, where parties have access to a public random permutation, and build the JustGarble system to implement them. JustGarble evaluates moderate-sized garbled-circuits at an amortized cost of 23.2 cycles per gate (7.25 nsec), far faster than any prior reported results.

Standard constructions of garbling schemes, including ours, provide only *static* security, meaning the input x is not allowed to depend on the garbled circuit F. But some applications—notably *one-time programs* (Goldwasser, Kalai, and Rothblum 2008) and *secure outsourcing* (Gennaro, Gentry, Parno 2010)—need *adaptive* security, where x may depend on F. We identify gaps in proofs from these papers with regard to adaptive security, which signifies the absence of a good abstraction boundary. We then investigate adaptive security of garbling schemes, giving definitions encompassing privacy, authenticity, and obliviousness, with either *coarse-grained* or *fine-grained* adaptivity. We show how adap-

tively secure garbling schemes support simple solutions for one-time programs and secure outsourcing, with privacy being the goal in the first case and obliviousness and authenticity the goal in the second. We give transforms that promote static-secure garbling schemes to adaptive-secure ones. This gives another compelling evidence that conceptualizing garbling schemes as a first-class cryptographic primitive can simplify, unify, or improve treatments for higher-level protocols.

Acknowledgments

First and foremost, I would like to thank my advisor, Phillip Rogaway. Phil is also a close friend of mine, a role model, and a superb teacher. His obsession with elegance and typography, his forceful yet humorous writing style, and his unique eccentricity are all contagious. During last five years, under Phil's inspiring guidance, I have improved tremendously, both as a person and as a scientist.

Mihir Bellare serves as my second advisor, a source of wisdom, and an oracle of knowledge. I have imbibed a vast amount of his refined taste for research questions and his brilliant problem-solving skill. Mihir also instills in me the hope and confidence to become a great researcher.

I wish to thank Sriram Keelveedhi for his excellent work in implementing the JustGarble system, and Matthew Franklin for serving in my thesis committee. Matt introduced me to the concept of garbled circuits and strongly influenced my decision to pursue the topic.

Finally, I thank my family for their unconditional love and for filling my life with happiness and joy.

Chapter 1

Introduction

Consider the following motivating examples:

- *Private outsourcing:* A client wants to outsource a computation on her confidential data x to a cloud, yet doesn't want to reveal x to the cloud.
- *Electronic voting:* A group of people want to vote over the Internet. The election protocol should ensure that (i) each person votes at most once, (ii) each ballot votes for at most one candidate, and (iii) everybody, except the voter, knows nothing about the content of the ballot.
- *Privacy-preserving auction:* A group of users want to bid for an item, and who bids the highest price wins the auction. There is no trusted arbitrator, and the winner pays the *second* highest price. The protocol should ensure that each user knows if she wins the auction, and, for the winner, the amount of money she has to pay—but nothing else.

The examples above are special cases of the Multiparty Computation problem (MPC). There are m players $P_1, \ldots, P_m$, and each player P_i has private input x_i and public function f_i, for every $i \in \{1, \ldots, m\}$. The goal is to let P_i compute $f_i(x_1, \ldots, x_m)$ without learning anything else about the other players' input. Being an important problem in its own right, the case $m = 2$ is often referred to as Secure Function Evaluation (SFE). Andrew Yao [96, 97] is the first to describe and study MPC. In talks [38, p. 27], he even suggested a technique,

commonly known as Yao's *garbled circuits* (GC), to achieve SFE. GCs eventually become a central tool in cryptography, enjoying diverse applications [9, 34, 35, 50, 78, 88]. Beyond this, some GC-based protocols have turned practical. Beginning with Fairplay [72], a bit of a cottage industry has emerged to improve the efficiency and practicality of GC-based MPC [49, 51, 64, 66, 84].

In this thesis, we deviate from the ingrained mindset that views GCs as a cryptographic technique. We instead advocate a different point of view, one that sees *garbling schemes* as a stand-alone cryptographic object. This abstraction enables more modular use of garbled circuits in higher-level protocols, resulting in the discovery of critical bugs subtly hidden in several well-known papers [6, 37, 45, 64, 84]. Our foundational, practice-oriented treatment also leads to new and highly efficient schemes, ones which are more than two orders of magnitude faster than prior work [51, 66].

1.1 A brief history

While Yao's 1982/1986 papers [96, 97] are often cited for reference to GCs, strangely, there is *no* description of GCs in these two papers. From what the text in these papers literally suggests, the 1982 paper only raises the question of MPC and claims that there *exists* a method to achieve SFE based on trapdoor one-way functions, without giving any further details [1]; while the 1986 paper merely hints that the technique is also based on probabilistic encryptions [43].

The first written account of the method is given by Goldreich, Micali, and Wigderson (GMW) [41]. The protocol they describe, crediting Yao [96], involves associating two *tokens* to each wire of a boolean circuit, these having hidden semantics of 0 and 1. Means are then provided to propagate tokens across a gate, preserving the hidden semantics. More

[1] According to Y. Ishai (personal communication), Yao already conceived the idea of GCs in his 1982 paper, but lacked the language to write everything down clearly. Only after the concept of computational indistinguishability appeared [43] was Yao able to clarify his thoughts and give oral presentations on the technique.

specifically, there's a four-row table for each gate of the circuit, each row employing public-key encryption[2] to encrypt a pair of random strings whose xor is the token for the outgoing wire.

The term *garbled circuit*[3] is from Beaver, Micali, and Rogaway [11], where the method was first based on a symmetric primitive. Garbled circuits took on a modern, PRF-based instantiation in work by Naor, Pinkas, and Sumner on privacy-preserving auctions [78].

Yao's garbled-circuits technique has been extremely influential, engendering an enormous number of applications, implementations, and refinements. Still, there has been little definitional attention paid to garbled circuits themselves. A 2004/2009 paper by Lindell and Pinkas [68, 70] provides the first explicit proof of Yao's protocol—to the extent one can say that a particular scheme is Yao's—but, even there, the authors do not formalize garbled circuits or what it means to securely create one. Instead, they prove that a particular garbled-circuit-using protocol, one based on double encryption,[4] is a secure two-party SFE. Implemented SFE methods [84] do not coincide with what's in Lindell and Pinkas [70], but still refer to the proof in [70] to (incorrectly) justify the security of their protocols.

1.2 Motivation for our study

Despite the enormous impact of GCs, there is a crisis of rigor in GC-based applications and implementations. Given the complexity of GCs, some authors choose to neglect a security proof, and instead rely on the intuition that their protocols are correct [35, 78]. Some other authors instead try to single out, definitionally, precisely what they need from GCs for some intending application, but none of the papers pick up definitions from any other, nor does any prove that any particular construction satisfies the notion given [1, 9, 23, 59, 63].

[2]It seems to have been almost forgotten that garbled circuits were originally conceived as a technique based on public-key techniques. Abadi and Feigenbaum (1990), for example, explain that an advantage of their approach is that only one composite $N = pq$ is needed for the entire circuit, not a different one for each gate [1]. Garbled circuits have long since lost their association to public-key encryption, let alone a specific public-key technique.

[3]Synonyms in the literature include *encrypted circuit* and *scrambled circuit*, while GMW refer to Yao's protocol (GCs + oblivious transfers) as *combined oblivious transfer*. The term *garbled circuit* however caught on and now becomes the standard name.

[4]This approach for making the rows of the garbled gate is first mentioned by Goldreich [38].

Other authors choose a concrete instantiation of GC to work with, but the absence of a good abstraction boundary makes daunting the task of providing a full proof [37, 45, 69, 75]. Sometimes, the needed property is beyond what the standard constructions can deliver, and this approach may lead to serious and subtle bugs, as in [37, 45].

A line of definitions begins with Ishai and Kushilevitz [54] and continues with [2, 4, 6, 7, 55, 56, 88]. These works define various flavors of *randomized encodings*. Their authors do see randomized encodings as a general-purpose primitive, and the definitions elegantly support a variety of theory-centered work. However, they lack the fine-grained syntax that we shall need for properties needed for one-time programs [45] and secure outsourcing [37], and precise measures of efficiency. Indeed, this causes a bug in the secure outsourcing constructed in [6].

On the other hand, it is common for implementations of GC to start from a basic, proven scheme, and then implement an instantiation, enhancement, or variant that is not itself proven. For example, there is still no proof for schemes that use both the free xor [84] and garbled-row reduction [84]. Absence of proof can belie presence of error. Indeed, during the course of our study, we discover a bug that breaks a few instantiations [64, 84].

Once viewed as a "theoretical" approach for multiparty computation, a long line of work, beginning with Fairplay [72], has made clear that circuit garbling is now a practical technique. State-of-the-art implementations by Huang *et al.* and Kreuter *et al.* can handle complex functionalities and hundreds of millions of gates [51, 52, 66]. Still, there are hidden opportunities to improve the speed that are realized only when GCs are properly formalized. As an analog, authenticated encryption took off after it was reconceptualized as a primitive, not a method formed of encryption schemes and MACs.

1.3 Our contributions

In this thesis we aim to instill fresh, practice-oriented foundation in an area where, historically, omitted definitions and proofs have been the norm. Below are our three main

contributions.

FORMALIZATION. We formalize what we call a *garbling scheme*. The notion is designed to support a burgeoning and practical area: the myriad applications of garbled circuits. Our definitions and results enable easy and widespread applications with modular, simplified, and yet more rigorous proofs of security. On the other hand, with a protocol's garbling scheme delineated, implementations can more reasonably offer proofs for the actual scheme employed, the "messy" optimizations stripped of surrounding interaction and protocol aims. In general, an approach where the garbling scheme is conceptually separated from its use seems essential for managing complexity in this domain.

We define several security notions. The most important one is *privacy*, implicitly levied by MPC protocols. Beyond privacy, we consider *obliviousness* and *authenticity*; these are suitable for private, verifiable outsourcing of computation. We provide efficient schemes that meet the three notions. The schemes are conveniently described in terms of a *dual-key cipher* (DKC), a notion we put forward. We give several efficient instantiations of a DKC from either a pseudorandom function or a blockcipher.

EFFICIENT GARBLING. We show how to construct and evaluate garbling schemes at unprecedented speeds. Our gains come from two main sources. On the cryptographic side, we describe garbling schemes that need only *one* AES128 call per gate and all blockcipher invocations use the *same* key. More precisely, we design DKCs in the random-permutation model that are compatible with existing optimization techniques such as free xor or garbled-row reduction [51, 64, 84]. Each such DKC makes a single call to the ideal permutation that is instantiated by fixed-key AES. On the systems side, we exploit more efficient representations of circuits. The combination of faster DKCs and a simple representation of circuits results in impressive performance gains over previous implementations.

We also point out a critical error in prior work [64, 84]. There, the DKC of Fairplay [72] is claimed to work [64] with free xor. Other authors have gone so far as to implement MPC using this DKC [84]; the construction has only been considered undesirable because it is less

efficient than alternatives, not because its security was in doubt. We show that this not to be the case, by completely breaking the resulting schemes.

ADAPTIVE SECURITY. Standard constructions of garbling schemes, including ours, provide only *static* security, meaning the input x is not allowed to depend on the garbled circuit F. But some applications—notably *one-time programs* [45] and *secure outsourcing* [37]—need *adaptive* security, where x may depend on F. We point out gaps in proofs from these papers with regard to adaptive security, which suggests a missing abstraction boundary. The applications we point to motivate the study of adaptive security for garbling schemes, while the gaps indicate that the issues may be more subtle than recognized.

We show how adaptively secure garbling schemes support simple solutions for one-time programs and secure outsourcing, with privacy being the goal in the first case and obliviousness and authenticity the goal in the second. We give transforms that promote static-secure garbling schemes to adaptive-secure ones. The simplicity of these transformations underscores our tenet that abstracting garbling schemes and treating adaptive security for them enables modular and rigorous applications of the garbled-circuit technique. Basing the applications on garbling schemes also allows instantiations to inherit efficiency features of future schemes.

1.4 Organization and history of thesis results

ORGANIZATION. In Chapter 2 we provide general notation, and briefly review the game-based proof method commonly used in cryptography. We also give a formalization of circuits, as it is not possible to properly specify a circuit-garbling algorithm or a circuit-evaluation function, nor to carry out code-based game-playing proofs, without circuits being formalized. Paralleling the three contributions above, the framework of garbling schemes and the static security notions are given in Chapter 3; the implementation of extremely efficient garbling schemes in Chapter 4, and the formalization of adaptive security in Chapter 5.

PUBLICATION HISTORY OF THESIS RESULTS. The formalization of circuits in Chapter 2 and the framework of garbling schemes in Chapter 3 were developed by Bellare, Hoang, and Rogaway [16]. The results in Chapter 4 are the fruits of the collaboration between Bellare, Hoang, Keelveedhi, and Rogaway [13]. Keelveedhi implemented the JustGarble system and designed a simple circuit representation based on the definition in Chapter 2. The contents of Chapter 5 were developed by Bellare, Hoang, and Rogaway [18].

What's written here is the combination of the full versions [14, 17, 19] of the three papers above. This thesis also presents new material. In [16] we gave a DKC construction based on fixed-key AES without justifying its security. To fill this gap, in Section 3.5.8, we show that this instantiation indeed satisfies the DKC security.

Chapter 2

Preliminaries

2.1 Notation

We let $\mathbb{N}$ be the set of positive integers. A *string* is a finite sequence of bits and $\perp$ is a formal symbol that is not a string. If A is a finite set then $y \twoheadleftarrow A$ denotes selecting an element of A uniformly at random and assigning it to y. If A is an algorithm then $A(x_1, \ldots; r)$ denotes the output of A on inputs $x_1, \ldots$ and coins r, while $y \leftarrow A(x_1, \ldots)$ means we pick r uniformly at random and let $y \leftarrow A(x_1, \ldots; r)$. We let $[A(x_1, \ldots)]$ denote the set of y that have positive probability of being output by $A(x_1, \ldots)$. We write $\mathrm{Func}(a, b)$ for $\{f\colon \{0,1\}^a \to \{0,1\}^b\}$. Polynomial time (PT) is always measured in the length of *all* inputs, not just the first. (But random coins, when singled out as an argument to an algorithm, are never regarded as an input.) As usual, a function $\varepsilon\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for every $c > 0$ there is a K such that $\varepsilon(k) < k^{-c}$ for all $k > K$.

2.2 Code-based games

Our definitions and proofs are expressed via code-based games [21] so we recall here the language and specify the particular conventions we use. A code-based game—see Fig. 3.3.1 for an example—consists of an INITIALIZE procedure, procedures that respond to adversary

oracle queries, and a FINALIZE procedure. All procedures are optional. In an execution of game Gm with an adversary $\mathcal{A}$, the latter is given input 1^k where k is the security parameter, and the security parameter k used in the game is presumed to be the same. Procedure INITIALIZE, if present, executes first, and its output is input to the adversary, who may now invoke other procedures. Each time it makes a query, the corresponding game procedure executes, and what it returns, if anything, is the response to $\mathcal{A}$'s query. The adversary's output is the input to FINALIZE, and the output of the latter, denoted $\mathrm{Gm}^{\mathcal{A}}(k)$, is called the output of the game. FINALIZE may be absent in which case it is understood to be the identity function, so that the output of the game is the output of the adversary. We let "$\mathrm{Gm}^{\mathcal{A}}(k) \Rightarrow c$" denote the event that this game output takes value c and let "$\mathrm{Gm}^{\mathcal{A}}(k)$" be shorthand for "$\mathrm{Gm}^{\mathcal{A}}(k) \Rightarrow \mathsf{true}$." Boolean flags are assumed initialized to false and $\mathsf{BAD}(\mathrm{Gm}^{\mathcal{A}}(k))$ is the event that the execution of game Gm with $\mathcal{A}$ sets flag *bad* to true.

2.3 Circuits

While our definitions for garbling schemes are representation-independent, the garbling schemes we specify assume a circuit-based representation. Here we specify the conventions and definitions that make this formal.

There are several reasons why it is important to cleanly define circuits (which, for many reasons, are not just DAGs). First, there are many "boundary cases" where only conventions can decide if something is or is not a valid circuit.[1] The boundary cases matter; we have repeatedly found that degenerate or under-analyzed circuit types materially impact if a garbling scheme is correct.[2] Beyond this, a lack of agreement on what a circuit *is* makes even informal discourse problematic.[3] Finally, we have found that it is simply not possible

[1]For example: Can an input wire be an output wire? Can an output wire be an incoming wire to another gate? Can an output wire be used twice in forming the output? Can a wire twice feed a gate? Can constants feed a gate? Can gates compute asymmetric functions like $G(x, y) = \overline{x} \vee y$?

[2]For example, the scheme of Naor, Pinkas, and Sumner [78] cannot handle a wire being used twice as an input to another gate (as when making a NOT gate from a NAND), a restriction that is nowhere explicitly said. The scheme of Beaver, Micali, and Rogaway [11] was buggy [91] because of a dependency in gate-labels associated to fan-out ≥ 2 gates.

[3]For example, is there a single wire emanating from each gate, that one wire connected to all gates it

to properly specify a circuit-garbling algorithm or a circuit-evaluation function, nor to carry out code-based game-playing proofs, without circuits being formalized. As an added payoff, if one establishes good conventions for circuits, then these same conventions can be used when defining a garbled circuit and its evaluation function.

Besides serving the theoretical purpose, our definition of circuits leads to a practical benefit. The implementation of a garbling scheme by Kreuter, Shelat, and Shen [66] spends most of its time in running *non-cryptographic* operations. The root of this problem seems to be the developers' choice of a complex data structure to represent circuits. In Chapter 4 we describe the implementation of a simple representation of circuits to programmatically realize our mathematical definition. This significantly reduces the non-cryptographic overhead.

SYNTAX. A (conventional) *circuit* is a 6-tuple $f = (n, m, q, A, B, G)$. Here $n \geq 2$ is the number of *inputs*, $m \geq 1$ is the number of *outputs*, and $q \geq 1$ is the number of *gates*. We let $r = n + q$ be the number of *wires*. We let $\text{Inputs} = \{1, \ldots, n\}$, $\text{Wires} = \{1, \ldots, n + q\}$, $\text{OutputWires} = \{n + q - m + 1, \ldots, n + q\}$, and $\text{Gates} = \{n + 1, \ldots, n + q\}$. Then $A : \text{Gates} \to \text{Wires}\backslash\text{OutputWires}$ is a function to identify each gate's *first* incoming wire and $B \colon \text{Gates} \to \text{Wires}\backslash\text{OutputWires}$ is a function to identify each gate's *second* incoming wire. Finally $G \colon \text{Gates} \times \{0,1\}^2 \to \{0,1\}$ is a function that determines the *functionality* of each gate. We require $A(g) < B(g) < g$ for all $g \in \text{Gates}$. See the left side of Fig. 2.3.1 for an illustration of a circuit.

The conventions above embody all of the following. Gates have two inputs, arbitrary functionality, and arbitrary fan-out. The wires are numbered 1 to $n + q$. Every non-input wire is the outgoing wire of some gate. The ith bit of input is presented along wire i. The ith bit of output is collected off wire $n + q - m + i$. The outgoing wire of each gate serves as the name of that gate. Output wires may not be input wires and may not be incoming wires to gates. No output wire may be twice used in the output. Requiring $A(g) < B(g) < g$ ensures that the directed graph corresponding to f is acyclic, and that no wire twice feeds

feeds, or is there a separate wire from the output of a gate to each gate it feeds? (For us, it'll be the first.) These are very different meanings of *wire*.

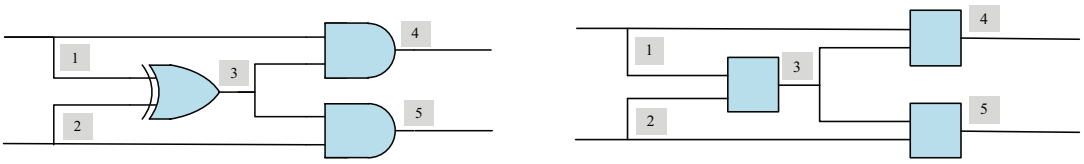

Figure 2.3.1: **Left: A conventional circuit** $f = (n, m, q, A, B, G)$. It has $n = 2$ inputs, $m = 2$ outputs, and $q = 3$ gates. Gates are numbered 3, 4, 5, according to their outgoing wires. The diagram encodes $A(3) = 1$, $B(3) = 2$, $A(4) = 1$, $B(4) = 3$, $A(5) = 3$, and $B(5) = 2$. The gate symbols indicate that $G_1(\cdot, \cdot) = \text{XOR}$ and $G_2(\cdot, \cdot) = G_3(\cdot, \cdot) = \text{AND}$. **Right: A topological circuit** f^- corresponding to the circuit on the left.

a gate; the numbering of gates comprise a topological sort.

We will routinely ignore the distinction between a circuit $f = (n, m, q, A, B, G)$ as a 6-tuple and the encoding of such a 6-tuple as a string; formally, one assumes a fixed and reasonable encoding, one where $|f|$ is $O(r \log r)$ for $r = n + q$.

Evaluating a circuit. We define a canonical evaluation function $\mathsf{ev}_{\mathsf{circ}}$. It takes a string f and a string $x = x_1 x_2 \cdots x_n$:

```
01 proc ev_circ(f, x)
02 (n, m, q, A, B, G) ← f
03 for g ← n + 1 to n + q do a ← A(g), b ← B(g), x_g ← G_g(x_a, x_b)
04 return x_{n+q−m+1} ··· x_{n+q}
```

At line 02 we adopt the convention that any string f can be parsed as a circuit. (If f does not encode a circuit, we view it as some fixed, default circuit.) This ensures that $\mathsf{ev}_{\mathsf{circ}}$ is well-defined for all string inputs f. At line 03, values x_a and x_b will always be well defined because of $A(g) < B(g) < g$. Circuit evaluation takes linear time.

Topological circuits. We say f^- is a *topological circuit* if $f^- = (n, m, q, A, B)$ for some circuit $f = (n, m, q, A, B, G)$. Thus a topological circuit is like a conventional circuit except the functionality of the gates is unspecified. See the right side of Fig. 2.3.1. Let Topo be the function that expunges the final component of its circuit-valued argument, so $f^- = \text{Topo}(f)$ is the topological circuit underlying conventional circuit f.

Discussion. The definition above evolved over a long time. For example, in an earlier

version (May 2011), we defined a circuit as a 9-tuple $(n, m, q, r, w, A, B, D, G)$, where r is the number of wires, $D : \{1, \ldots, q\} \to \text{Wires}\backslash\text{Inputs}$ is a bijective function identifying each gate's outgoing wire, and $w : \{1, \ldots, m\} \to \text{Wires}$ is a function identifying the output locations (what output are where). That is, gates are indexed $1, \ldots, q$; there is no relation between a gate index and its outgoing wire; an output bit can come from any wire; and two output bits may come from the same wire. We later simplified this to the current definition because:

- There is no need to keep r, as $r = q + n$,
- One can reindex each gate as its outgoing wire, and thus D is the identity function and can be omitted, and
- If we impose the convention that the output bits must come from the last m wires, that is $w(i) = q + n - m + i$, then w can be dropped. This may slightly increase the size of a circuit, but there are two solid reasons to do so, besides the sake of simplicity. First, when one hardwires a circuit $f = (n, m, q, w, A, B, G)$ to a universal circuit $\mathscr{U}$ and garbles the resulting circuit $\mathscr{U}(f, \cdot)$, contrary to the folklore belief, not only (n, m, q) but also w is revealed. Next, if we allow an incoming wire of a gate to be also an output wire, then the scheme Garble2 in Section 3.6 will be insecure.

Even the 9-tuple definition above represents the result of an evolution process. Only when one has a blueprint of the intended garbling schemes can one spell out a precise definition of circuits, as the perception of circuits will loosely determine how garbling schemes are constructed. For example, consider a wire connecting a gate to another. If you think of it as two separate wires, one coming out of the first gate, and one feeding the second gate, then your garbling scheme will probably look more like the LEGO scheme [80] than the scheme Garble2 in Section 3.6. On the other hand, if you define circuits in a way that there is a separate wire from the output of a gate to each gate it feeds, then you might well stumble to Applebaum, Ishai, and Kushilevitz's scheme [4].

Chapter 3

Foundational Treatment

3.1 Introduction

Overview. This chapter is about elevating garbled circuits from a cryptographic *technique* to a cryptographic *goal*. While circuit garbling has traditionally been viewed as a method for achieving SFE (secure function evaluation) or some other cryptographic goal, we view it as an end goal in its own right, defining *garbling schemes* and formalizing several notions of security for them, these encompassing *privacy*, *authenticity*, and *obliviousness*. This enables more modular use of garbled circuits in higher-level protocols and grounds follow-on work, including the development of new and highly efficient schemes.

Contributions. We formalize what we call a *garbling scheme*. Our definitions and results enable easy and widespread applications with modular, simplified, and yet more rigorous proofs of security.

Roughly said, a garbling algorithm Gb is a randomized algorithm that transforms a function $f\colon \{0,1\}^n \to \{0,1\}^m$ into a triple of functions $(F, e, d) \leftarrow \mathsf{Gb}(f)$. We require that $f = d \circ F \circ e$. The *encoding function* e turns an *initial input* $x \in \{0,1\}^n$ into a *garbled input* $X = e(x)$. Evaluating the *garbled function* F on the garbled input X gives a *garbled output* $Y = F(X)$. The *decoding function* d turns the garbled output Y into the *final output*

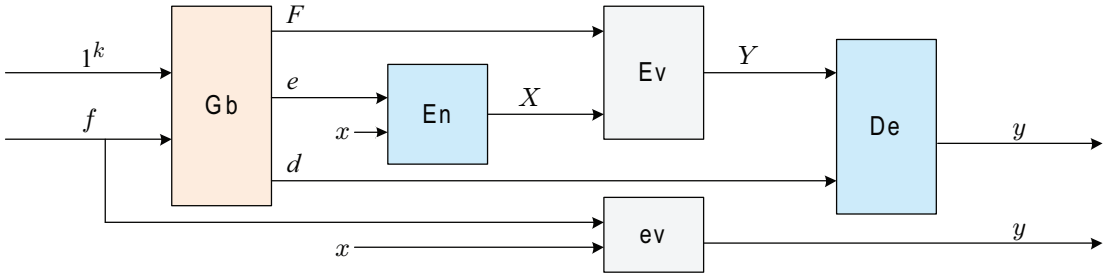

Figure 3.1.1: Components of a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Function Gb maps f and k to (F, e, d), strings encoding the garbled function, the encoding function, and the decoding function. Possession of e and x lets one compute the garbled input $X = \mathsf{En}(e, x)$; having F and X lets one calculate the garbled output $Y = \mathsf{Ev}(F, X)$; and knowing d and Y lets one recover the final output $y = \mathsf{De}(d, Y)$, which must equal $\mathsf{ev}(f, x)$.

$y = d(Y)$, which must coincide with $f(x)$. Informally, one has probabilistically factored f into $d \circ F \circ e$. Formally, it is problematic to regard Gb as operating on functions. Thus a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is regarded as a five-tuple of algorithms, with strings d, e, f, and F interpreted as functions under the auspices of functions De, En, ev, and Ev. See Fig. 3.1.1.

Our syntactic framework is representation-independent. Besides circuits, one can garble DFAs, RAMs, OBDDs, TMs, whatever. See Section 3.8, "Eclectic representations."

Of course none of this says anything about the desired security notion. We define several. The most important is *privacy*: a party acquiring (F, X, d) shouldn't learn anything impermissible beyond that which is revealed by knowing just the final output y. To formalize that which it *is* permissible to reveal, a *side-information function*, Φ, parameterizes the definition; an adversary should be able to ascertain from (F, X, d) nothing beyond $\Phi(f)$ and y. By varying Φ one can encompass the customary setting for SFE (let $\Phi(f) = f$; circuit f is not concealed) and PFE (private function evaluation) (let $\Phi(f)$ be the number of gates of f; leak just the circuit's size). We formalize privacy in multiple ways, giving an indistinguishability definition, prv.ind, and a simulation-based one, prv.sim. We show that whether or not they are equivalent depends on the side-information function Φ. For the most important ones the notions are equivalent (in general, they are not).

We provide a simple garbling scheme, Garble1, for achieving privacy. The scheme is

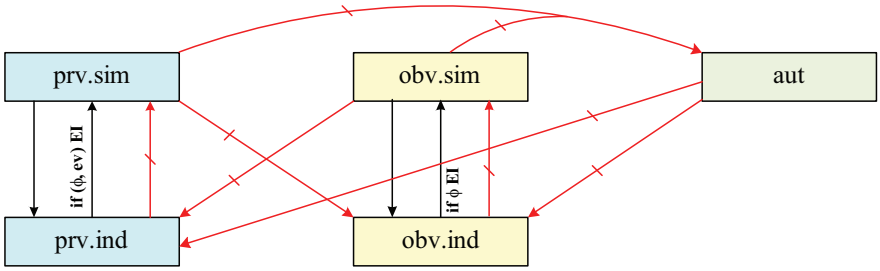

Figure 3.1.2: **Relations among security notions.** A solid arrow is an implication; an if-labeled arrow, a conditional implication; a hatched arrow, a separation. Implications and separations are in Section 3.4.

conveniently described in terms of a *dual-key cipher* (DKC), a notion we put forward. We define a DKC's security and prove privacy for Garble1 under this assumption. Garble1 is described with uncustomary precision, including a detailed and precise definition of circuits. We show how to make a DKC from a pseudorandom function (PRF), and how to realize the PRF using a conventional blockcipher, say AES128. In this way we obtain a provably secure, blockcipher-based garbling scheme where circuit evaluation takes two AES calls per gate.

We go on to suggest a still more efficient instantiation for the dual-key cipher, one where evaluating a garbled circuit needs only *one* AES128 call per gate and all blockcipher invocations use the *same* key. This is the fastest approach now known for garbling circuits. In Chapter 4 we further explore this direction to make it compatible with existing optimization techniques such as free xor or garbled-row reduction that have proven so effective [51, 64, 84].

Beyond privacy we consider *obliviousness*: a party acquiring F and X, but not d, shouldn't learn anything about f, x, or y. As with privacy, we formalize obliviousness in different but "usually" equivalent ways. Next we explore *authenticity*: a party who learns F and X should be unable to produce a garbled output Y^* different from $F(X)$ that is deemed to be valid: $d(Y^*) \neq \bot$. Our interest in obliviousness and authenticity was sparked by Gennaro, Gentry, and Parno [37]; the notions arise in the context of private, verifiable outsourcing of computation.

We prove implications and separation among all security notions we have mentioned, painting a complete picture of definitions for this space. See Fig. 3.1.2.

Protocol	**Application**	**Needs**	**Over**
Y86 [38]	2-party SFE (sh)	prv	Φ_{circ}
AF90 [1]	PFE (sh)	prv	Φ_{size}
FKN94 [34]	server-aided SFE (sh)	prv	Φ_{circ}
NPS99 [78]	private auctions	prv	Φ_{circ}
KO04 [63]	2-party SFE (ma)	prv	Φ_{circ}
FAZ05 [35]	private credit checking	prv	Φ_{size}
FM06 [75]	2-party SFE (ma)	prv	Φ_{circ}
AL07 [8]	2-party SFE (covert)	prv	Φ_{circ}
LP07 [69]	2-party SFE (ma)	prv	Φ_{circ}
GMS08 [47]	2-party SFE (co)	prv	Φ_{circ}
BFK+09 [10]	priv medical diag	obv	Φ_{circ}
PSS09 [81]	private credit checking	prv	Φ_{topo}
BHHI10 [9]	KDM encryption	prv	Φ_{size}
KM10 [61]	secure text processing	prv	Φ_{topo}
HS10 [50]	2P guaranteed SFE	prv	Φ_{circ}
SS10 [88]	worry-free encryption	prv	Φ_{size}
A11 [2]	KDM encryption	prv	Φ_{size}
KMR11 [59]	server-aided SFE (ma)	aut + obv	Φ_{circ}
LP11 [71]	2-party SFE (ma)	prv	Φ_{circ}

Figure 3.1.3: **Recasting protocols in more generic terms.** sh = semi-honest; co = covert; ma = malicious. All but [37] need the scheme to be projective.

We define a protocol, Garble2, to simultaneously achieve privacy, obliviousness, and authenticity. The assumption required is the same as before. The scheme is only a bit more complex than Garble1, the efficiency, only a little worse.

Discussion. Garble1 and Garble2 are close to numerous other protocols (especially [78]) that incarnate Yao's idea. Given this, one might assume that, once good definitions *are* written down, proving security would be easy, based on prior work [70]. From our experience, this is not the case; the proofs we provide are not implicit in prior work.

One thing novel about our schemes is that they admit efficient AES-based instantiations whose quantitative security may be inferred via the concrete security bounds associated to our theorems. In the past, SFE schemes supported by proofs would use objects less efficiently realizable in practice [70], or, for practical realizations, would abandon proven-secure schemes and use hash-based ones, sometimes with an unproven claim that security

is maintained in the random-oracle model. Given the increasing ubiquity of AES hardware support, we believe that optimized, proven, blockcipher-based schemes are a good direction.

A thesis underlying our definitions is that they *work*—that most (though not all) applications described as using garbled circuits can be built from an arbitrary garbling scheme, instead. To date we have surveyed 20 papers containing protocols that can be recast to use a generic garbling scheme. See Fig. 3.1.3. In all cases we gain in simplicity and modularity. Applications benefit from the increased efficiency of our garbling schemes. The improvement is particularly marked in the application to KDM encryption (security with respect to key-dependent messages), where use of our abstraction leads to substantial efficiency gains over the use of the abstractions in previous work [2, 9].

3.2 Garbling schemes and their security

We define garbling schemes and security notions for them. See Chapter 2 should any notation seem non-obvious.

3.3 Syntax

A *garbling scheme* is a five-tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. The first of these is probabilistic; the remaining algorithms are deterministic. A string f, the *original function*, describes the function $\mathsf{ev}(f, \cdot) : \{0,1\}^n \to \{0,1\}^m$ that we want to garble.[1] The values $n = f.n$ and $m = f.m$ depend on f and must be easily computable from it. Specifically, fix linear-time algorithms n and m to extract $f.n = \mathsf{n}(f)$ and $f.m = \mathsf{m}(f)$.[2] On input f and a security parameter $k \in \mathbb{N}$, algorithm Gb returns a triple of strings $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$. String e describes an *encoding function*, $\mathsf{En}(e, \cdot)$, that maps an *initial input* $x \in \{0,1\}^n$ to a

[1] By way of example, the string f may encode a circuit that $\mathsf{ev}(f, \cdot)$ can evaluate at input x.

[2] For concreteness, one can define $\mathsf{n}(f)$ and $\mathsf{m}(f)$ to be n and m if f is a tuple $(n, m, \ldots)$ and define $\mathsf{n}(f) = \mathsf{m}(f) = 1$ otherwise. Of course other encoding conventions are also fine.

garbled input $X = \mathsf{En}(e, x)$.[3] String F describes a *garbled function*, $\mathsf{Ev}(F, \cdot)$, that maps each garbled input X to a *garbled output* $Y = \mathsf{Ev}(F, X)$. String d describes a *decoding function*, $\mathsf{De}(d, \cdot)$, that maps a garbled output Y to a *final output* $y = \mathsf{De}(d, Y)$.

We levy some simple requirements on garbling schemes. First, $|F|$, $|e|$, and $|d|$ may depend only on k, $f.n$, $f.m$, and $|f|$. Formally, if $f.n = f'.n$, $f.m = f'.m$, $|f| = |f'|$, $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$, and $(F', e', d') \in [\mathsf{Gb}(1^k, f')]$, then $|F| = |F'|$, $|e| = |e'|$, and $|d| = |d'|$. This is the *length* condition. Second, e and d may depend only on k, $f.n$, $f.m$, $|f|$ and the random coins r of Gb. Formally, if $f.n = f'.n$, $f.m = f'.m$, $|f| = |f'|$, $(F, e, d) = \mathsf{Gb}(1^k, f; r)$, and $(F', e', d') = \mathsf{Gb}(1^k, f'; r)$, then $e = e'$ and $d = d'$. This is the *nondegeneracy* condition. Finally, if $f \in \{0,1\}^*$, $k \in \mathbb{N}$, $x \in \{0,1\}^{f.n}$, and $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$, then $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = \mathsf{ev}(f, x)$. This is the *correctness* condition.

We say that a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is a *circuit-garbling scheme* if ev interprets f as a circuit: formally, $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$ for the canonical circuit-evaluation function that we defined in Section 2.3.

3.3.1 Projective schemes

A common approach in existing garbling schemes is for e to encode a list of *tokens*, one pair for each bit in $x \in \{0,1\}^n$. Encoding function $\mathsf{En}(e, \cdot)$ then uses the bits of $x = x_1 \cdots x_n$ to select from $e = (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ the subvector $X = (X_1^{x_1}, \ldots, X_n^{x_n})$. Formally, we say that garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is *projective* if for all f, $x, x' \in \{0,1\}^{f.n}$, $k \in \mathbb{N}$, and $i \in [1..n]$, when $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$, $X = \mathsf{En}(e, x)$ and $X' = \mathsf{En}(e, x')$, then $X = (X_1, \ldots, X_n)$ and $X' = (X'_1, \ldots, X'_n)$ are n vectors, $|X_i| = |X'_i|$, and $X_i = X'_i$ if x and x' have the same ith bit.

Our definitions of security do not require schemes be projective. Nevertheless, this property is needed for some important applications. For example, SFE can be achieved by combining a projective garbling scheme and a scheme for oblivious transfer.

[3] By way of example, the encoding function e might be a sequence of $2n$ strings, called *tokens*, a pair for each bit of x. The garbled input X might then be a sequence of n strings, or *tokens*, one for each bit of x.

3.3.2 Side-information functions

Privacy is rarely absolute; semantically secure encryption, for example, is allowed to reveal the length of the plaintext. Similarly, a garbled circuit might reveal the size of the circuit that was garbled, its topology (that is, the graph of how gates are connected up), or even the original circuit itself. The information that we expect to be revealed is captured by a *side-information function*, Φ, which deterministically maps f to a string $\phi = \Phi(f)$. We will parameterize our advantage notions by Φ, and in this way simultaneously define garbling schemes that may reveal a circuit's size, topology, identity, or more. We require that $f.n$ and $f.m$ be easily determined from $\phi = \Phi(f)$; formally, there must exist linear-time algorithms n' and m' that compute $f.n = \mathsf{n}'(\phi) = \mathsf{n}(f)$ and $f.m = \mathsf{m}'(\phi) = \mathsf{m}(f)$ when $\phi = \Phi(f)$. We also require that $|f|$ be easily determined from $\Phi(f)$.

Specific side-information functions are useful for circuit garbling. Side-information function Φ_{size} reveals the number of inputs, outputs, and gates of a circuit f; formally, $\Phi_{\text{size}}(f) = (n, m, q)$ for a circuit $f = (n, m, q, A, B, G)$. Side-information function Φ_{topo} reveals the topological circuit but not the functionality of each gate: $\Phi_{\text{topo}}(f) = (n, m, q, A, B)$. Side-information function Φ_{xor} reveals the topological circuit and which gates are XOR, but obscures the functionality of the non-XOR gates. Formally, $\Phi_{\text{xor}}(f)$ is circuit (n, m, q, A, B, G') where $G'_g = \text{XOR}$ if $G_g = \text{XOR}$ and, arbitrarily, $G'_g = \text{AND}$ otherwise. Side-information function Φ_{circ} reveals the entire circuit: $\Phi_{\text{circ}}(f) = f$.

Discussion. Side-information functions Φ_{circ} and Φ_{size} are motivated by the classical applications of garbled circuits: Secure Function Evaluation (SFE) and Private Function Evaluation (PFE). The "textbook" garbling schemes [70, 78], however, deliver more security than what it is required, leaking only $\Phi_{\text{topo}}(f)$ instead of the entire circuit f. Paus, Sadeghi, and Schneider [81] exploit this property to obtain faster implementation for applications such as privacy-preserving credit checking or secure data classification. Practical instantiations [51, 66] however leak more than just the topological circuit of f, as they all employ the free-xor trick [64], and therefore need to reveal which gates are XOR. Side-information

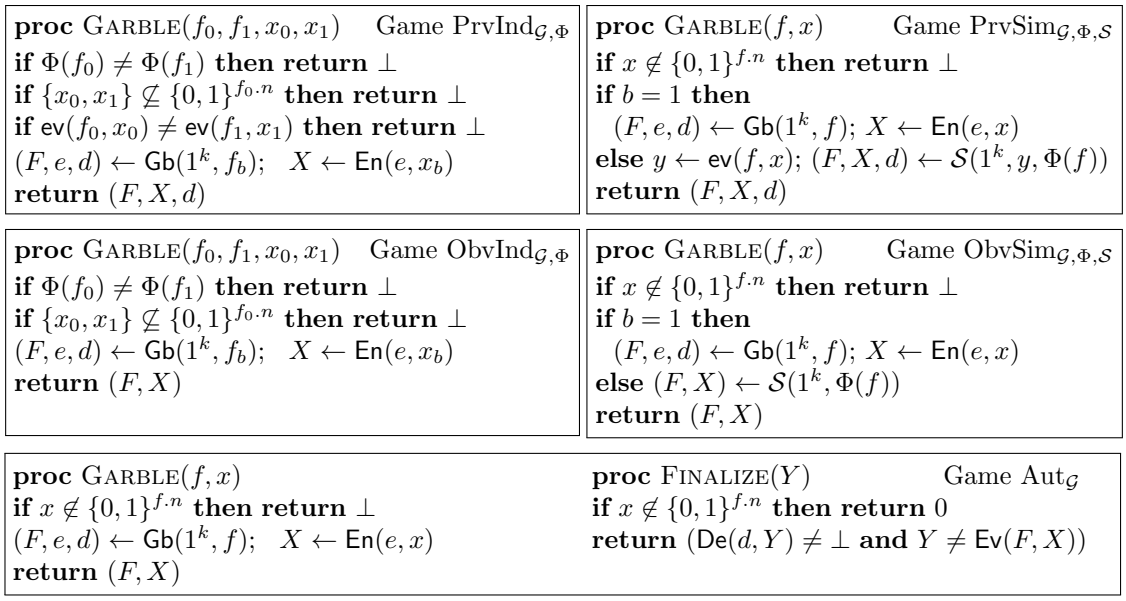

Figure 3.3.1: **Games for defining the prv.ind, prv.sim, obv.ind, obv.sim, and aut security of a garbling scheme** $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Here $\mathcal{S}$ is a simulator, Φ is an information function and k is the security parameter input to the adversary. In the first four games, procedure INITIALIZE() picks a bit $b \twoheadleftarrow \{0,1\}$, and procedure FINALIZE(b') returns $(b = b')$.

function Φ_{xor} captures what's leaked in such garbling schemes.

3.3.3 Security notions

PRIVACY. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme, $k \in \mathbb{N}$ a security parameter, and Φ a side-information function. We define an indistinguishability-based notion of privacy via game PrvInd$_{\mathcal{G},\Phi}$ (top-left of Fig. 3.3.1) and a simulation-based notion of privacy via game PrvSim$_{\mathcal{G},\Phi,\mathcal{S}}$ (top-right of Fig. 3.3.1, where $\mathcal{S}$ is a simulator). Executing either game with an adversary requires one to specify the garbling scheme, adversary, security parameter, and side-information function. Executing game PrvSim additionally requires one to specify the algorithm $\mathcal{S}$. Notation and conventions for games are specified in Chapter 2.

Refer first to game PrvInd$_{\mathcal{G},\Phi}$. Initially, procedure INITIALIZE() samples a bit b at random. Adversary $\mathcal{A}$ gets input 1^k and must make exactly one GARBLE query. That query is answered as specified in the game, the security parameter used here being the same as the one provided to the adversary. The adversary must eventually halt, outputting a bit b', and

the game's FINALIZE procedure determines if the adversary has won on this run, namely, if $b = b'$. The corresponding advantage is defined via

$$\mathbf{Adv}^{\text{prv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) = 2\Pr[\text{PrvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k)] - 1,$$

the probability, normalized to $[0, 1]$, that the adversary correctly predicts b. Scheme $\mathcal{G}$ is prv.ind secure over Φ if for every PT adversary $\mathcal{A}$ the function $\mathbf{Adv}^{\text{prv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible.

Explaining the definition, the game picks challenge bit b and the adversary chooses (f_0, x_0) and (f_1, x_1) such that $\Phi(f_0) = \Phi(f_1)$ and, also, $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. The game then garbles f_b to (F, e, d) and encodes x_b as the garbled input $X = \mathsf{En}_e(x_b)$. The adversary is given (F, X, d), which determines $y = \mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x_b))) = \mathsf{ev}(f_b, x_b)$. The adversary must guess b. In a scheme we deem secure, it should be unable to ascertain which of (f_0, x_0), (f_1, x_1) got garbled.

Next we define prv.sim security via game $\text{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$ associated to garbling scheme $\mathcal{G}$, information function Φ and an algorithm $\mathcal{S}$ called a simulator. Initially, procedure INITIALIZE() samples a bit b at random. The adversary $\mathcal{B}$ is run on input 1^k and must make exactly one GARBLE query. The query is answered as specified in Fig. 3.3.1, with k being the same as the input to the adversary. The adversary must eventually output a bit, and the game's FINALIZE procedure indicates if the adversary has won—again, if the adversary correctly predicted b. The adversary's advantage is

$$\mathbf{Adv}^{\text{prv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) = 2\Pr[\text{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)] - 1\,,$$

the probability, normalized to $[0, 1]$, that the adversary wins. Protocol $\mathcal{G}$ is prv.sim secure over Φ if for every PT adversary $\mathcal{B}$ there is a PT algorithm $\mathcal{S}$ such that $\mathbf{Adv}^{\text{prv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k)$ is negligible.

Let us again explain. For the prv.sim notion we let the adversary choose (f, x). Either we garble it to $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and $X \leftarrow \mathsf{En}(e, x)$, handing the adversary (F, X, d), or else we ask the simulator to devise a "fake" (F, X, d) based solely on k, $\phi = \Phi(f)$, and $y = \mathsf{ev}(f, x)$. From this limited information the simulator must produce an (F, X, d) indistinguishable, to the adversary, from the ones produced using the actual garbling scheme.

The indistinguishability definition for garbling schemes is simpler due to the absence of the simulator, but we consider this notion "wrong" when the side-information function is such that indistinguishability is inequivalent to the simulation-based definition. See Section 3.4.

OBLIVIOUSNESS. Informally, a garbling scheme achieves *obliviousness* if possession of a garbled function F and garbled input X lets one compute the garbled output Y, yet (F, X) leaks nothing about f or x beyond $\Phi(f)$. The adversary does not get the decoding function d and will not learn the output $\mathsf{De}(d, \mathsf{Ev}(F, X))$. Contrasting this with privacy, there the agent evaluating the garbled function *does* learn the output; here, she learns not even that, as a needed piece of information, d, is withheld. Privacy and obliviousness are both secrecy notions, and cut from the same cloth. Yet they will prove incomparable: a private scheme could divulge the output even without d; an oblivious scheme could reveal too much once d is shown.

As with privacy, we formalize two notions, obv.ind and obv.sim, via the games of Fig. 3.3.1. The formalizations consider games $\mathrm{ObvInd}_{\mathcal{G},\Phi}$ and $\mathrm{ObvSim}_{\mathcal{G},\Phi,\mathcal{S}}$, run with adversaries $\mathcal{A}$ and $\mathcal{B}$, respectively. As usual the adversary gets input 1^k and the security parameter used in the game is also k. The adversary makes a single call to the game's GARBLE procedure and outputs a bit b'. We define

$$\begin{aligned} \mathbf{Adv}^{\mathrm{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) &= 2\Pr[\mathrm{ObvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k))] - 1 \quad \text{and} \\ \mathbf{Adv}^{\mathrm{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) &= 2\Pr[\mathrm{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)] - 1 \end{aligned}$$

as the probability, normalized to $[0, 1]$, that adversary's output is a correct guess of the underlying bit b. Protocol $\mathcal{G}$ is obv.ind secure over Φ if for every PT adversary $\mathcal{A}$, we have that $\mathbf{Adv}^{\mathrm{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k)$ is negligible. It is obv.sim secure over Φ if for every PT adversary $\mathcal{B}$ there exists a PT simulator $\mathcal{S}$ such that $\mathbf{Adv}^{\mathrm{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, \cdot)$ is negligible.

Let us explain the difference between prv.ind and obv.ind. First, we no longer demand that $\mathsf{ev}(f, x_0) = \mathsf{ev}(f, x_1)$: the adversary may now name any (f_0, x_0) and (f_1, x_1) as long as

the functions have the same side information. Second, the decoding function d is no longer provided to the adversary. The adversary must guess if (F, X) stems from garbling (f_0, x_0) or (f_1, x_1).

Similarly, the difference between prv.sim and obv.sim is two-fold. First, in the obliviousness notion the simulator is denied $y = \mathsf{ev}(f, x)$; it must create a convincing (F, X) without that. Second, the simulator no longer returns to the adversary the (simulated) decoding function d; the return value is (F, X) and not (F, X, d).

AUTHENTICITY. So far we have dealt exclusively with secrecy notions. One can formalize an authenticity property as well [37], which we do via game $\mathrm{Aut}_{\mathcal{G}}$ of Fig. 3.3.1. Authenticity captures an adversary's inability to create from a garbled function F and its garbled input X a garbled output $Y \neq F(X)$ that will be deemed authentic.

Fix a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, adversary $\mathcal{A}$, and security parameter $k \in \mathbb{N}$. Run adversary $\mathcal{A}$ on input 1^k, allowing it a single call to the GARBLE procedure of the game. The adversary outputs a string Y, and, when it does, the game's FINALIZE procedure is called to decide if the adversary has won. The adversary's aut-advantage is defined as $\mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) = \Pr[\mathrm{Aut}^{\mathcal{A}}_{\mathcal{G}}(k)]$. Protocol $\mathcal{G}$ is aut-secure if $\mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible for all PT adversaries $\mathcal{A}$.

SETS OF GARBLING SCHEMES. To compactly and precisely express relations between notions we will write them as containments and non-containments between sets of garbling schemes. To this end, for $\mathrm{xxx} \in \{\mathrm{prv.ind}, \mathrm{prv.sim}, \mathrm{obv.ind}, \mathrm{obv.sim}\}$ we let $\mathsf{GS}(\mathrm{xxx}, \Phi)$ be the set of all garbling schemes that are xxx-secure over Φ. Similarly, we let $\mathsf{GS}(\mathrm{aut})$ be the set of all garbling schemes that are aut-secure.

We also let $\mathsf{GS}(\mathsf{ev})$ be the set of all garbling schemes $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ whose evaluation function is ev. This captures garbling schemes for a particular class of functions. As per our previous notation, $\mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ now denotes the set of all circuit-garbling schemes.

3.3.4 Remarks

We end this section with discussion of our definitions.

Universal circuits. In several applications such as PFE, the circuit f is also private and we want to leak nothing but its size. Direct constructions of garbled circuits however leak the topological circuit or even more. To achieve this end, instead of garbling f, we hardwire it to a universal circuit [93] and garble the resulting one. This idea was first written down explicitly in a 1990 paper by Abadi and Feigenbaum [1], and now becomes folklore. See Section 3.9 for details on universal circuits and the overhead they entail.

Non-degeneracy. In garbling f by Gb we intend to partition f into e, F, d where e describes how to obscure the input x and where d describes how to unobscure the answer Y. We do not want $\mathsf{En}(e,\cdot)$ or $\mathsf{De}(d,\cdot)$ to actually compute $f(x)$. But this could happen if we permitted decompositions like $e = f$, $F = d = \varepsilon$, $\mathsf{En}(e,x) = \mathsf{ev}(f,x)$, and $\mathsf{Ev}(F,X) = \mathsf{De}(d,X) = X$. The nondegeneracy condition outlaws this, formalizing a sense in which e and d are independent of f. Note that we do allow e and d to depend on m, n, and even $|f|$.

Invalid queries. Referring to Fig. 3.3.1, we note that the bit b is well-defined—it is set in the Initialize procedure—even if the adversary's query to Garble is invalid (meaning that it returns $\perp$) in games PrvInd or ObvInd. If this were not the case then the semantics of the Finalize procedure would be unclear: one would be asking if $b = b'$, but b would be undefined.

Strict correctness. Our correctness condition is *strict*: you always get $\mathsf{ev}(f,x)$ by computing $\mathsf{De}(d,\mathsf{Ev}(F,\mathsf{En}(e,x)))$. One can certainly relax this requirement, and you would have to in order to regard what goes on within Lindell and Pinkas [70], say, as a garbling scheme. Yet strict correctness is not hard to achieve. Our definition could certainly be extended to say that a scheme is correct if $\Pr[(F,e,d) \leftarrow \mathsf{Gb}(1^k,f)\colon \mathsf{De}(d,\mathsf{Ev}(F,\mathsf{En}(e,x))) \neq \mathsf{ev}(f,x)]$ is negligible as a function of k for all f.

An undesirable way to do asymptotics. Several prior papers [62, 68, 69] conflate the

security parameter k and f's input length n. These are conceptually distinct, and it makes perfect sense to think of f, and therefore n, as fixed, while the security parameter varies. In our treatment, the security parameter k is provided to the adversary and it selects the functions to use in its attack and so, as a result, the input length n is polynomially bounded if the adversary is. The security parameter limits the input length—the input length does not define the security parameter.

INDISTINGUISHABILITY WITHOUT SIDE-INFORMATION. The side-information function Φ does more than allow one to capture that which may be revealed by F; our prv.ind definition would be meaningless if we had effectively dialed-in $\Phi(f) = f$, the "traditional" understanding for 2-party SFE. Suppose here that we wish only to garble SHA-256, so $\mathsf{ev}(f, x) = \text{SHA-256}(x)$ for all f, x. Then the adversary can't find any distinct x_0 and x_1 such that $\mathsf{ev}(f, x_0) = \mathsf{ev}(f, x_1)$—which means that good prv.ind security will be achieved no matter *what* the garbling scheme does. An interpretation of this observation is that prv.ind is an unacceptable definition when $\Phi(f) = f$—one must ensure that less leaks about f before the definition starts to say something useful. When the adversary needs only to find $(f_0, x_0) \neq (f_1, x_1)$ such that $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$, and when Φ is designed to make sure this is an easy job for her, the definition is more meaningful.[4]

IDEALIZED MODELS. As in many cryptographic domains, it seems possible to obtain better efficiency working in idealized models [20]. All of our security definitions easily lift to ideal-model settings. In the *random-oracle* model (ROM) [20], we provide any adversary, and any algorithms among the first four components of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, with access to a random oracle HASH. We then distinguish between the PROM (Programmable-ROM) and the NPROM (Non-Programmable ROM) whose procedures HASH are given in Fig. 3.3.2 (left and right, respectively). In the latter model, the simulator too has oracle access to the random oracle, but in the former model, it does not have such access and will instead itself reply to the queries made by the adversary to its random oracle. In the code, `ro` is

[4] An asymptotic version of the counterexample is in Proposition 3.4.10.

proc HASH(ℓ, w)	**proc** HASH(ℓ, w)
if $\mathsf{H}[\ell, w] = \bot$ **then**	**if** $\mathsf{H}[\ell, w] = \bot$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
if $b = 1$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$	**return** $\mathsf{H}[\ell, w]$
else $\mathsf{H}[\ell, w] \twoheadleftarrow \mathcal{S}(\ell, w, \mathtt{ro})$	
return $\mathsf{H}[\ell, w]$	

Figure 3.3.2: **Extending garbling-scheme security to ROM.** Games of our security notions may be extended to include either the procedure on the left (the PROM) or the right (the NPROM). The adversary and scheme algorithms have oracle access to HASH. In the NPROM case, the simulator also has access to HASH. In the PROM case the simulator does not have access to HASH and instead, when the challenge bit b is 0, must itself answer queries to HASH as indicated above.

a formal symbol indicating to the simulator that it is being asked to answer a query to HASH. In the *ideal-cipher* model we provide, instead, an ideal cipher $E\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ and its inverse $D\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$, each key K naming an independent random permutation $E(K,\cdot)$, where ℓ may depend on the security parameter. In the *ideal-permutation* model we provide, instead, a random permutation $\pi\colon \{0,1\}^\ell \to \{0,1\}^\ell$ and its inverse $\pi^{-1}\colon \times\{0,1\}^\ell \to \{0,1\}^\ell$. Security results for any of these models would bound the adversary's advantage in terms of the number and type of its oracle queries.

3.4 Relations

We show that prv.sim always implies prv.ind, and prv.ind implies prv.sim under certain added conditions on the side-information function. We show that the same holds for obv.ind and obv.sim, under a weaker assumption on the side-information function. The conditions on the side-information function are relatively mild. We will also justify the non-implications for the security notions compactly summarized in Fig. 3.1.2. As part of this we will show that prv.ind does not always imply prv.sim and obv.ind does not always imply obv.sim.

3.4.1 Invertibility of side-information functions

Let Φ be a side-information function. An algorithm M is called a Φ-inverter if on input ϕ in the range of Φ it returns a preimage under Φ of that point, meaning a string f such that $\Phi(f) = \phi$. Such an inverter always exists, but it might not be efficient. We say that Φ is *efficiently invertible* if there is a polynomial-time Φ-inverter. Similarly, an algorithm M is called a (Φ, ev)-inverter if on input (ϕ, y), where $\phi = \Phi(f')$ and $y = \mathsf{ev}(f', x')$ for some f' and $x \in \{0,1\}^{f'.n}$, returns an (f, x) satisfying $\Phi(f) = \phi$ and $\mathsf{ev}(f, x) = y$. We say that (Φ, ev) is *efficiently invertible* if there is a polynomial-time (Φ, ev)-inverter.

The following theorem summarizes the invertibility attributes of the circuit-related size-information functions we defined earlier. It shows that $\Phi_{\text{circ}}, \Phi_{\text{xor}}, \Phi_{\text{topo}}$, and Φ_{size} are efficiently invertible, and $(\Phi_{\text{size}}, \mathsf{ev}_{\mathsf{circ}}), (\Phi_{\text{topo}}, \mathsf{ev}_{\mathsf{circ}})$, and $(\Phi_{\text{xor}}, \mathsf{ev}_{\mathsf{circ}})$ are efficiently invertible.

Proposition 3.4.1. For $\Phi \in \{\Phi_{\text{size}}, \Phi_{\text{topo}}, \Phi_{\text{xor}}, \Phi_{\text{circ}}\}$, there is a linear-time inverter. For $\Phi \in \{\Phi_{\text{size}}, \Phi_{\text{topo}}\}$ there is a linear-time $(\Phi, \mathsf{ev}_{\mathsf{circ}})$-inverter. There is a cubic-time $(\Phi_{\text{xor}}, \mathsf{ev}_{\mathsf{circ}})$-inverter.

In contrast, there is no efficient $(\Phi_{\text{circ}}, \mathsf{ev}_{\mathsf{circ}})$-inverter (under a computational assumption); consider the case where f is drawn from a family implementing a one-way function.

Proof. We first specify a linear-time $(\Phi_{\text{topo}}, \mathsf{ev}_{\mathsf{circ}})$-inverter M_{topo}. It gets input a topological circuit f^- and an m-bit binary string $y = y_1 \cdots y_m$ and proceeds as follows:

proc $M_{\text{topo}}(f^-, y)$
$(n, m, q, A, B) \leftarrow f^-,\ \ y_1 \cdots y_m \leftarrow y$
for $(g, i, j) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ **do**
  if $g \leq n + q - m$ **then** $G_g(i, j) \leftarrow 0$ **else** $G_g(i, j) \leftarrow y_{g-(n+q-m)}$
$f \leftarrow (n, m, q, A, B, G),\ x \leftarrow 0^n$
return (f, x)

We have $\text{Topo}(f) = f^-$ and $\mathsf{ev}_{\mathsf{circ}}(f, x) = y$ as desired.

```
proc M_xor(φ, y)
(n, m, q, A, B, G') ← φ,  y_1 ··· y_m ← y,  S ← ∅
for g ∈ {n+1, ..., n+q} do
   a ← A(g),  b ← B(g)
   if G'_g = XOR then
      if g ≤ n+q−m then S ← S ∪ {x_a ⊕ x_b ⊕ x_g = 0}
      else S ← S ∪ {x_a ⊕ x_b = y_{g−(n+q−m)}}
(x_1, ..., x_{n+q−m}) ← GAUSS(S)
for (g, i, j) ∈ {n+1, ..., n+q} × {0,1} × {0,1} do
   if G'_g = XOR then G_g ← XOR else G_g(i, j) ← x_g
f ← (n, m, q, A, B, G),  x ← x_1 ··· x_n
return (f, x)
```

Figure 3.4.1: **The inverter M_{xor} for the proof of Proposition 3.4.1.**

Next we specify a linear-time $(\Phi_{\text{size}}, \mathsf{ev}_{\mathsf{circ}})$-inverter M_{size}. It gets input (n, m, q) and an m-bit binary string $y = y_1 \cdots y_m$ and proceeds as follows:

proc $M_{\text{size}}((n, m, q), y)$

for $g \in \{n+1, \ldots, n+q\}$ **do** $A_g \leftarrow 1$, $B_g \leftarrow 2$

$f^- \leftarrow (n, m, q, A, B)$, $(f, x) \leftarrow M_{\text{topo}}(f^-, y)$

return (f, x)

We have $\Phi_{\text{size}}(f) = (n, m, q)$ and $\mathsf{ev}_{\mathsf{circ}}(f, x) = y$ as desired.

We specify a cubic-time $(\Phi_{\text{xor}}, \mathsf{ev}_{\mathsf{circ}})$-inverter M_{xor} as follows. Let GAUSS(S) be the algorithm that takes as input a system S of linear equations in GF(2), uses Gaussian elimination to solve it, and then lets each free variable be 0. The inverter M_{xor} gets as input $\phi = (n, m, q, A, B, G')$ and a string $y \in \{0,1\}^m$, and proceeds as the code in Fig. 3.4.1. We then have (f, x) as desired. The system S has $q + n - m$ variables, and at most q equations. Hence the running time of GAUSS(S) is at most $O\big((q+n)^3\big)$, and so is M_{xor}'s running time.

Now a linear-time Φ_{topo}-inverter, on input $f^- = (n, m, q, A, B)$, can let $y \leftarrow 0^m$ and return $M_{\text{topo}}(f^-, y)$. A linear-time Φ_{size}-inverter, on input (n, m, q), can let $y \leftarrow 0^m$ and return $M_{\text{size}}((n, m, q), y)$. A linear-time Φ_{xor}-inverter, on input $f' = (n, m, q, A, B, G')$, simply returns f'. Finally, a linear-time Φ_{circ}-inverter is trivial, returning f on input f. □

3.4.2 Equivalence of prv.ind and prv.sim

The following says that prv.sim implies prv.ind security, and conversely if (Φ, ev) is efficiently invertible.

Proposition 3.4.2. [prv.ind $\approx$ prv.sim] For any PT Φ: (1) $\mathsf{GS}(\text{prv.sim}, \Phi) \subseteq \mathsf{GS}(\text{prv.ind}, \Phi)$ and (2) If (Φ, ev) is efficiently invertible then $\mathsf{GS}(\text{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \subseteq \mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$.

The first part says that if garbling scheme $\mathcal{G}$ is prv.sim secure over Φ then $\mathcal{G}$ is prv.ind secure over Φ. The second part says that if garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is prv.ind secure over Φ and (Φ, ev) is efficiently invertible then $\mathcal{G}$ is prv.sim secure over Φ. Proposition 3.4.10 proves that efficient invertibility of (Φ, ev) is required to prove that prv.ind implies prv.sim, so the notions are not always equivalent.

The reductions underlying Proposition 3.4.2 are tight. This is evidenced by Eq. (3.4.1) and Eq. (3.4.2) in the proof and the fact that the running times of the constructed adversaries or simulators are about the same as that of the starting adversary.

Proof. For part (1), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{prv.sim}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{prv.ind}, \Phi)$. Let $\mathcal{A}$ be a PT adversary attacking the prv.ind-security of $\mathcal{G}$ over Φ. We construct a PT prv.sim-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ run $\mathcal{A}(1^k)$. Without loss of generality, suppose that $\mathcal{A}$ queries (f_0, f_1, x_0, x_1) such that $\Phi(f_0) = \Phi(f_1)$, $x_0, x_1 \in \{0,1\}^{f_0.n}$, and $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$; otherwise $\mathbf{Adv}^{\text{prv.ind},\Phi}_{\mathcal{G}}(\mathcal{A}, k) = 0$ and it will be trivial to construct adversary $\mathcal{B}$ such that $\mathbf{Adv}^{\text{prv.sim},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) = 0$ for any simulator $\mathcal{S}$. Adversary $\mathcal{B}$ picks a bit c at random and queries f_c, x_c to its own GARBLE oracle to get back (F, X, d) and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 if $b' = c$, and returns 0 otherwise. The running time of $\mathcal{B}$ is about the same as that of $\mathcal{A}$. Let $\mathcal{S}$ be *any* algorithm

playing the role of the simulator. Then

$$\begin{aligned}\Pr\left[\,\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)\mid b=1\,\right] &= \frac{1}{2}+\frac{1}{2}\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)\\ \Pr\left[\,\neg\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)\mid b=0\,\right] &= \frac{1}{2}\end{aligned}$$

where b denotes the challenge bit in game $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$. Subtracting, we see that

$$\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)\le 2\cdot\mathbf{Adv}^{\mathrm{prv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k)\;. \tag{3.4.1}$$

By assumption there is a PT $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well.

For part (2), let $\mathcal{G}=(\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})\in\mathsf{GS}(\mathrm{prv.ind},\Phi)$ and let M be a (Φ,ev)-inverter. We want to show that $\mathcal{G}\in\mathsf{GS}(\mathrm{prv.sim},\Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the prv.sim-security of $\mathcal{G}$ over Φ. Without loss of generality, suppose that $\mathcal{B}$ queries (f,x) such that $x\in\{0,1\}^{f.n}$. We define a simulator $\mathcal{S}$ that on input $1^k,y,\phi$, lets $(f,x)\leftarrow M(\phi,y)$ then $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f)$. It outputs $(F,\mathsf{En}(e,x),d)$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1,x_1 to GARBLE, adversary $\mathcal{A}$ lets $(f_0,x_0)\leftarrow M(\Phi(f_1),\mathsf{ev}(f_1,x_1))$ and then queries f_0,f_1,x_0,x_1 to its own GARBLE oracle to get back (F,X,d), which it returns to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. The running time of $\mathcal{A}$ is about that of $\mathcal{B}$ plus the time to run M on $\mathcal{B}$'s query. In addition,

$$\begin{aligned}\Pr\left[\,\mathrm{PrvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k)\mid b=1\,\right] &= \Pr\left[\,\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)\mid c=1\,\right]\\ \Pr\left[\,\neg\mathrm{PrvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k)\mid b=0\,\right] &= \Pr\left[\,\neg\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k)\mid c=0\,\right]\end{aligned}$$

where b and c denote the challenge bits in games $\mathrm{PrvInd}_{\mathcal{G},\Phi}$ and $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$, respectively. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{prv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k)\le\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)\;. \tag{3.4.2}$$

But the RHS is negligible by assumption, hence the LHS is as well. □

A corollary of Propositions 3.4.1 and 3.4.2 is that prv.sim and prv.ind are equivalent for circuit-garbling schemes over side-information functions $\Phi_{\text{xor}}, \Phi_{\text{topo}}$, and Φ_{size}, which we summarize as:

Corollary 3.4.3. For any $\Phi \in \{\Phi_{\text{xor}}, \Phi_{\text{topo}}, \Phi_{\text{size}}\}$, we have $\mathsf{GS}(\text{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) = \mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$.

Equivalence in idealized models. In idealized models, define prv.nprom as prv.sim security in which the simulator has oracle access to the ideal primitives, and prv.prom as prv.sim security in which the simulator doesn't have access to the ideal primitives and will instead itself reply to the oracle queries made by the adversary. Proposition 3.4.2 implies that if (Φ, ev) is efficiently invertible then prv.prom and prv.nprom are equivalent. It suffices to show that prv.prom security implies prv.nprom, since the latter obviously implies the former. By part (1) of Proposition 3.4.2, prv.prom security implies prv.ind security. The proof still holds, even if the simulator $\mathcal{S}$ uses the programmability power to collude with the prv.prom adversary $\mathcal{B}$ to fool the prv.ind adversary $\mathcal{A}$, because what $(\mathcal{S}, \mathcal{B})$ receives is independent of $\mathcal{A}$'s challenge bit. Because (Φ, ev) is efficiently invertible, by part (2) of Proposition 3.4.2, prv.ind security then implies prv.nprom security.

3.4.3 Equivalence of obv.ind and obv.sim

The following says that obv.sim implies obv.ind security, and conversely if Φ is efficiently invertible. The invertibility condition is thus weaker than in the privacy case. Proposition 3.4.4 also implies that if Φ is efficiently invertible then obv.prom and obv.nprom are equivalent, where the latter is defined as obv.sim security in which the simulator has oracle access to the ideal primitives, and the former as obv.sim security in which the simulator doesn't have access to the ideal primitives and will instead itself reply to the oracle queries made by the adversary.

Proposition 3.4.4. [obv.ind $\approx$ obv.sim] For any PT Φ, we have (1) $\mathsf{GS}(\text{obv.sim}, \Phi) \subseteq$

$\mathsf{GS}(\text{obv.ind}, \Phi)$ and (2) If Φ is efficiently invertible, $\mathsf{GS}(\text{obv.ind}, \Phi) \subseteq \mathsf{GS}(\text{obv.sim}, \Phi)$.

Proposition 3.4.11 shows that Φ being efficiently invertible is required to prove that obv.ind implies obv.sim. But the side-information function Φ we use is artificial; for any "reasonable" one we know, obv.ind and obv.sim will be equivalent. The reductions underlying Proposition 3.4.4 are also tight. This is evidenced by Eq. (3.4.3) and Eq. (3.4.4) in the proof and the fact that the running times of the constructed adversaries or simulators are about the same as that of the starting adversary.

Proof. The proof is analogous to that of Proposition 3.4.2; for completeness we provide details. For part (1), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{obv.sim}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{obv.ind}, \Phi)$. Let $\mathcal{A}$ be a PT adversary attacking the obv.ind-security of $\mathcal{G}$ over Φ. We construct a PT obv.sim-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ run $\mathcal{A}(1^k)$. Without loss of generality, suppose that $\mathcal{A}$ queries (f_0, f_1, x_0, x_1) such that $\Phi(f_0) = \Phi(f_1)$ and $x_0, x_1 \in \{0,1\}^{f_0}$. Adversary $\mathcal{B}$ picks a bit c at random and queries f_c, x_c to its own GARBLE oracle to get back (F, X, d) and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 if $b' = c$, and returns 0 otherwise. The running time of $\mathcal{B}$ is about the same as that of $\mathcal{A}$. Let $\mathcal{S}$ be *any* algorithm playing the role of the simulator. Then

$$\begin{aligned} \Pr\left[\,\text{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid b = 1\,\right] &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\text{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}) \\ \Pr\left[\,\neg\text{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid b = 0\,\right] &= \frac{1}{2} \end{aligned}$$

where b denotes the challenge bit in game $\text{ObvSim}_{\mathcal{S}}$. Subtracting, we see that

$$\mathbf{Adv}^{\text{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \leq 2 \cdot \mathbf{Adv}^{\text{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k)\ . \tag{3.4.3}$$

By assumption there is a PT $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well.

For part (2), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{obv.ind}, \Phi)$ and let M be a Φ-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{obv.sim}, \Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the obv.sim-security of $\mathcal{G}$ over Φ. Without loss of generality, suppose that $\mathcal{B}$ queries (f, x) such that $x \in$

$\{0,1\}^{f.n}$. We define a simulator $\mathcal{S}$ that on input $1^k, y, \phi$, lets $f \leftarrow M(\phi, y)$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$. It outputs $(F, \mathsf{En}(e, x), d)$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1, x_1 to GARBLE, adversary $\mathcal{A}$ lets $f_0 \leftarrow M(\Phi(f_1), \mathsf{ev}(f_1, x_1))$ and $x_0 \leftarrow 0^{f_0.n}$ and then queries (f_0, f_1, x_0, x_1) to its own GARBLE oracle to get back (F, X, d), which it returns to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. The running time of $\mathcal{A}$ is about that of $\mathcal{B}$ plus the time to run M on $\mathcal{B}$'s query. In addition,

$$\begin{aligned} \Pr\left[\,\mathrm{ObvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k) \mid b = 1\,\right] &= \Pr\left[\,\mathrm{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid c = 1\,\right] \\ \Pr\left[\,\neg\mathrm{ObvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k) \mid b = 0\,\right] &= \Pr\left[\,\neg\mathrm{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid c = 0\,\right] \end{aligned}$$

where b and c denote the challenge bits in games $\mathrm{ObvInd}_{\mathcal{G},\Phi}$ and $\mathrm{ObvSim}_{\mathcal{G},\Phi,\mathcal{S}}$, respectively. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) \le \mathbf{Adv}^{\mathrm{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k)\ . \tag{3.4.4}$$

But the RHS is negligible by assumption, hence the LHS is as well. □

Again a corollary of Propositions 3.4.1 and 3.4.4 is that obv.sim and obv.ind are equivalent for circuit-garbling schemes over $\Phi_{\mathrm{circ}}, \Phi_{\mathrm{xor}}, \Phi_{\mathrm{topo}}$ and Φ_{size}:

Corollary 3.4.5. For every side-information function $\Phi \in \{\Phi_{\mathrm{xor}}, \Phi_{\mathrm{topo}}, \Phi_{\mathrm{size}}, \Phi_{\mathrm{circ}}\}$, we have $\mathsf{GS}(\mathrm{obv.ind}, \Phi) = \mathsf{GS}(\mathrm{obv.sim}, \Phi)$.

3.4.4 Separations

We justify the non-implications for the security notions compactly summarized in Fig. 3.1.2. We state these as non-containments $\mathsf{A} \not\subseteq \mathsf{B}$ between sets of garbling schemes. We always assume $\mathsf{A} \neq \emptyset$, since otherwise the claim trivially fails.

The following says that privacy does not imply obliviousness, even when we take the strong form of privacy (simulation-style) and the weak form of obliviousness (ind-style):

Proposition 3.4.6. For every Φ and for $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$, we have $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{obv.ind}, \Phi)$.

Proof. By assumption $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \neq \emptyset$ so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{obv.ind}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ pick $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $((F, d), e, d)$. Let $\mathsf{Ev}'((F, d), X) = \mathsf{Ev}(F, X)$. Including d in the description of the garbled function does not harm prv.sim-security because an adversary is always given the descriptions of the garbled function and the decoding function simultaneously, so $\mathcal{G}'$ inherits the prv.sim-security of $\mathcal{G}$. On the other hand, $\mathcal{G}'$ fails to achieve obv.ind. An adversary simply makes query $(\text{OR}, \text{OR}, x_0, x_1)$ where $x_0 = 00$ and $x_1 = 11$. On receiving reply $((F, d), X)$, it outputs 0 if $\mathsf{De}(d, \mathsf{Ev}(F, X)) = \mathsf{ev}(\text{OR}, x_0)$ and outputs 1 otherwise. This works because $0 = \mathsf{ev}(\text{OR}, x_0) \neq \mathsf{ev}(\text{OR}, x_1) = 1$ and correctness guarantees that $\mathsf{De}(d, \mathsf{Ev}(F, X)) = \mathsf{ev}(\text{OR}, x_b)$ where b is the challenge bit. □

The following says that obliviousness does not imply privacy, even when we take the strong form of obliviousness (simulation-style) and the weak form of privacy (ind-style):

Proposition 3.4.7. Let $\Phi = \Phi_{\text{topo}}$ and $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$. Then, $\mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{prv.ind}, \Phi)$.

Proof. By assumption $\mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \neq \emptyset$ so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}', \mathsf{Ev}, \mathsf{ev})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{prv.ind}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ pick $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $(F, e, (d, e))$. Let $\mathsf{De}'((d, e), Y) = \mathsf{De}(d, Y)$. Including e in the description of the decoding function does not harm obv.sim-security because an adversary is never given the description of the decoding function, so $\mathcal{G}'$ inherits the obv.sim-security of $\mathcal{G}$. On the other hand, $\mathcal{G}'$ fails to achieve prv.ind. An adversary simply makes query $(f_0, f_1, 11, 11)$ where $f_0 = \text{AND}$ and $f_1 = \text{OR}$, which is valid because $\mathsf{ev}(f_0, 11) = \mathsf{ev}(f_1, 11)$. On receiving reply $(F, X, (d, e))$, it outputs 0 if $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, 01))) = 0$ and 1

otherwise. This works because $0 = \mathsf{ev}(f_0, 01) \neq \mathsf{ev}(f_1, 01) = 1$ and correctness guarantees that $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, 01))) = \mathsf{ev}(f_b, 01)$ where b is the challenge bit. □

The following says that privacy and obliviousness, even in conjunction and in their stronger forms (simulation-style), do not imply authenticity.

Proposition 3.4.8. For all Φ and for $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$: $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{aut})$.

Proof. By assumption $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \neq \emptyset$ so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}', \mathsf{Ev}', \mathsf{ev})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\text{obv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut})$. The construction is as follows. Let $\mathsf{Ev}'(F, X) = \mathsf{Ev}(F, X) \| 0$ and $\mathsf{De}'(d, Y \| b) = \mathsf{De}(d, Y)$ if $b = 0$ and 1 otherwise, where $b \in \{0, 1\}$. Appending a constant bit to the garbled output does not harm prv.sim security or obv.sim-security. On the other hand, $\mathcal{G}'$ fails to achieve aut. An adversary simply makes query $(\text{OR}, 00)$ and then outputs $1 \| 1$. □

The following says that authenticity implies neither privacy nor obliviousness, even when the latter are in their weaker (ind style) form.

Proposition 3.4.9. Let $\Phi = \Phi_{\text{topo}}$ and $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$. Then $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{prv.sim}, \Phi) \cup \mathsf{GS}(\text{obv.sim}, \Phi)$.

Proof. By assumption $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}) \neq \emptyset$ so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{prv.sim}, \Phi) \cup \mathsf{GS}(\text{obv.sim}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ pick $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $((F, f), e, d)$. Let $\mathsf{Ev}'((F, f), X) = \mathsf{Ev}(F, X)$. Appending f to F does not harm authenticity as the adversary has chosen f, and thus already knows it, in its attack. On the other hand, the garbled function leaks f so privacy and obliviousness both fail over Φ_{topo}. □

We saw in Proposition 3.4.2 that prv.ind implies prv.sim if (Φ, ev) is efficiently invertible. Now we show that this assumption is necessary by showing that in general prv.ind does not imply prv.sim. We say that $P\colon \{0,1\}^* \to \{0,1\}^*$ is a permutation if: (1) for every $x \in \{0,1\}^*$ we have $|P(x)| = |x|$; (2) for every distinct $x_0, x_1 \in \{0,1\}^*$ we have $P(x_0) \neq P(x_1)$. We say that P is *one-way* if for every PT adversary $\mathcal{I}$ the function $\mathbf{Adv}^{\mathrm{ow}}_P(\mathcal{I}, \cdot)$ is negligible, where for each $k \in \mathbb{N}$ we have let

$$\mathbf{Adv}^{\mathrm{ow}}_P(\mathcal{I}, k) \;=\; \Pr[\mathcal{I}(P(x)) = x]\ ,$$

with the probability over $x \twoheadleftarrow \{0,1\}^k$. We associate to P the evaluation function $\mathsf{ev}^P(f, x) = P(x)$ for all $f, x \in \{0,1\}^*$.

Proposition 3.4.10. Let Φ be the identity function. Let P be a one-way permutation and let $\mathsf{ev} = \mathsf{ev}^P$. Then $\mathsf{GS}(\text{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{prv.sim}, \Phi)$.

We note that the (Φ, ev) in Proposition 3.4.10 is not efficiently invertible due to the one-wayness of P, so this separation is consistent with Proposition 3.4.2.

Proof. We build $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ so that $\mathcal{G} \in \mathsf{GS}(\text{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G} \notin \mathsf{GS}(\text{prv.sim}, \Phi)$. Let $\mathsf{Gb}(1^k, f) = (f, \varepsilon, \varepsilon)$ for any f. Let $\mathsf{En}(\varepsilon, x) = x$ and $\mathsf{De}(\varepsilon, Y) = Y$ for all $x, Y \in \{0,1\}^n$. We claim that $\mathbf{Adv}^{\text{prv.ind}, \Phi}_{\mathcal{G}}(\mathcal{A}) = 0$ for any (even computationally-unbounded) adversary $\mathcal{A}$. Consider an adversary $\mathcal{A}$ that makes GARBLE query (f_0, f_1, x_0, x_1). For the response to not be $\perp$ it must be that $f_0 = f_1$ and $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$, meaning $P(x_0) = P(x_1)$. Since P is a permutation, it follows that $x_0 = x_1$, and thus the advantage of the adversary must be 0. However, one can trivially break the prv.sim security of $\mathcal{G}$, with respect to any PT simulator $\mathcal{S}$ as follows. Adversary $\mathcal{A}(1^k)$ lets $f \leftarrow \varepsilon$ and $x \twoheadleftarrow \{0,1\}^k$. It then queries (f, x) to the oracle GARBLE. On receiving (F, X, d), it outputs 1 if $X = x$, and 0 otherwise. The simulator $\mathcal{S}$ gets input f and $y = \mathsf{ev}(f, x) = P(x)$ and produces (F, X, d). The probability that $X = x$ is negligible by the one-wayness of P, so the adversary's output is 1 with negligible probability when the challenge bit is 0. □

We saw in Proposition 3.4.4 that obv.ind implies obv.sim if Φ is efficiently invertible.

Now we show that this assumption is necessary by showing that in general obv.ind does not imply obv.sim. Let π be a bijection from $\mathrm{Func}(2,1)$ to $\{0,1\}^4$. Such a bijection exists, as $|\mathrm{Func}(2,1)| = 16$. Let P be a one-way permutation. We associate to P and π the following side-information function $\Phi_{P,\pi}$. For each circuit $f = (n, m, q, A, B, G)$, let $\Phi_{P,\pi}(f) = (\mathrm{Topo}(f), P(L))$, where $L = L_1 \cdots L_q$ and $L_i = \pi(G_{n+i})$ for each $1 \leq i \neq q$.

Proposition 3.4.11. Let P be a one-way permutation and π a bijection from $\mathrm{Func}(2,1)$ to $\{0,1\}^4$. Let $\Phi = \Phi_{P,\pi}$ and $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$. Then $\mathsf{GS}(\mathrm{obv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\mathrm{obv.sim}, \Phi)$.

We note that the one-wayness of P means Φ is not efficiently invertible, so this separation is consistent with Proposition 3.4.4. We also note that although Φ might look strange it is functionally equivalent to Φ_{circ} in the sense that $\Phi(f_0) = \Phi(f_1)$ iff $\Phi_{\mathrm{circ}}(f_0) = \Phi_{\mathrm{circ}}(f_1)$. This is true because Φ reveals the topology by definition, and since π, P are bijections, $P(\pi(G))$ uniquely determines G. This implies $\mathsf{GS}(\mathrm{xxx}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \subseteq \mathsf{GS}(\mathrm{xxx}, \Phi_{\mathrm{circ}}) \cap \mathsf{GS}(\mathsf{ev})$ for both $\mathrm{xxx} \in \{\mathrm{obv.ind}, \mathrm{obv.sim}\}$. (It does not imply the sets are equal because P is one-way.) On the other hand $\mathsf{GS}(\mathrm{xxx}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}) \subseteq \mathsf{GS}(\mathrm{xxx}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ so the sets in the Proposition contain interesting and natural schemes even though they might look strange at first glance.

Proof. By assumption $\mathsf{GS}(\mathrm{obv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \neq \emptyset$ so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{obv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{obv.sim}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ pick $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $(f \| F, e, d)$. Let $\mathsf{Ev}'(f \| F, X)$ return $\mathsf{Ev}(F, X)$. We claim that $\mathcal{G}'$ is obv.ind secure over Φ but not obv.sim secure over Φ.

To justify the first claim, consider an adversary $\mathcal{A}$ that queries (f_0, f_1, x_0, x_1). For the response to not be $\perp$ it must be that $\Phi(f_0) = \Phi(f_1)$ and hence, by the functional equivalence noted above, that $\Phi_{\mathrm{circ}}(f_0) = \Phi_{\mathrm{circ}}(f_1)$. Thus $f_0 = f_1$. Prepending f to F therefore does no harm to the obv.ind security.

We justify the second claim by presenting an adversary $\mathcal{B}$ that trivially breaks the obv.sim security of $\mathcal{G}'$, with respect to any PT simulator. Adversary $\mathcal{B}(1^k)$ picks an arbitrary topolog-

ical circuit f^- of k gates and chooses $L \twoheadleftarrow \{0,1\}^{4k}$. Let $L = L_1 \cdots L_k$, where each $L_i \in \{0,1\}^4$. Let $G_{n+i} = \pi^{-1}(L_i)$ for every $1 \le i \le k$, and let $f = (f^-, G)$. The adversary then queries $(f, 0^{f.n})$ to GARBLE. When receiving the reply (F', X), it returns 1 if the first $|f|$ bits of F' equal f, and returns 0 otherwise, so that it always returns 1 when the challenge bit in the game is 1. A simulator $\mathcal{S}$ gets input 1^k and $\phi = (f^-, P(L))$ and produces an output (F', X). Let $f^- = (n, m, k, A, B)$. Note that if the simulator can produce G, it also can produce $L = L_1 \cdots L_q$ with each $L_i = \pi(G_{n+i})$. The probability that the first $|f|$ bits of F' equal $f = (f^-, G)$ is therefore negligible by the one-wayness of P, because L's sampling is independent of f^-. So the adversary's output is 1 with negligible probability when the challenge bit is 0. □

3.5 Achieving privacy: Garble1

We provide a simple, privacy-achieving circuit-garbling scheme, Garble1. It is described in terms of a new primitive, a *dual-key cipher* (DKC). We will prove security of Garble1 assuming the security of its DKC. We will then show how to instantiate a DKC using a PRF. Instantiating this PRF via AES leads to an efficient garbling scheme. Differently instantiating the DKC directly with AES can give even better efficiency.

DUAL KEY CIPHERS. Before describing Garble1 we will need to specify the syntax of a DKC. These objects formalize a two-key lockbox—one where you need *both* keys to open the box. This has long been used as a metaphor to explain how garbling schemes work (e.g., [70, pp. 163–164]), but Lindell and Pinkas also give a notion of *double-encryption security* for two-key probabilistic encryption schemes [70, pp. 170]. Dual-key ciphers provide a very different way to formalize an object sufficient to construct garbling schemes.

Formally, a *dual-key cipher* is a function $\mathbb{E}$ that associates to any $k \in \mathbb{N}$, any keys $A, B \in \{0,1\}^k$ and any tweak $T \in \{0,1\}^{\tau(k)}$ a permutation $\mathbb{E}^T_{A,B} \colon \{0,1\}^k \to \{0,1\}^k$. Let $\mathbb{D}^T_{A,B} \colon \{0,1\}^k \to \{0,1\}^k$ denote the inverse of this permutation. It is required that the maps

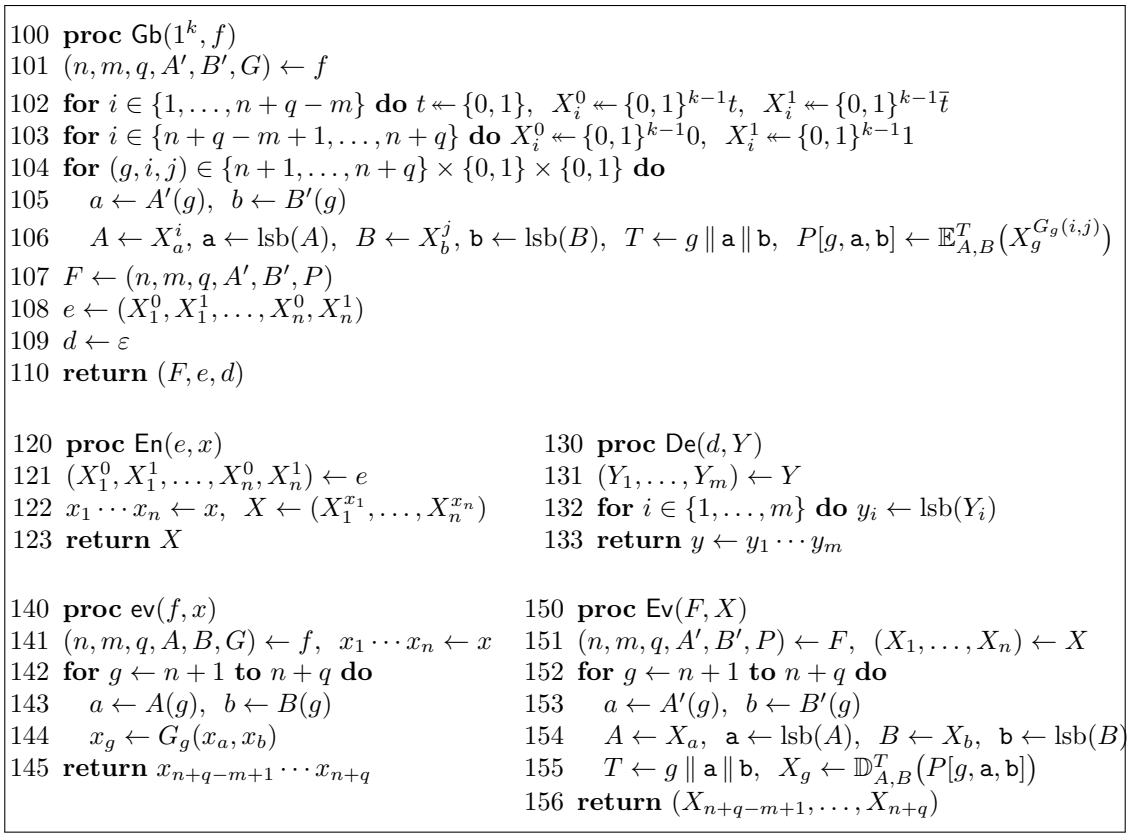

Figure 3.5.1: **Garbling scheme Garble1.** Its components are $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ where ev, shown for completeness, is the canonical circuit evaluation. We assume a DKC $\mathbb{E}$ with tweak length τ and let $\mathbb{D}$ denote its inverse. At line 102, we use $\{0,1\}^{k-1}t$ and $\{0,1\}^{k-1}\bar{t}$ to refer to the sets of k-bit binary strings whose last bit is t and $\bar{t}$ respectively.

$(A, B, T, X) \mapsto \mathbb{E}^T_{A,B}(X)$ and $(A, B, T, Y) \mapsto \mathbb{D}^T_{A,B}(Y)$ be polynomial-time computable. We refer to τ as the tweak length of $\mathbb{E}$.

The definition above describes syntax alone. We postpone giving a security definition until we've defined Garble1.

3.5.1 Definition of Garble1

Let $\mathbb{E}$ be a dual-key cipher with tweak length τ. We associate to $\mathbb{E}$ the garbling scheme Garble1[$\mathbb{E}$] as shown in Fig. 3.5.1 and illustrated in Fig. 3.5.2. Wires carry k-bit *tokens*. A token X will encode a one-bit *type*. Rather arbitrarily, the type is the final bit of the token, namely its LSB. When we write $T \leftarrow g \,\|\, \mathsf{a} \,\|\, \mathsf{b}$ (line 106 and 155) where $g \in \mathbb{N}$ and

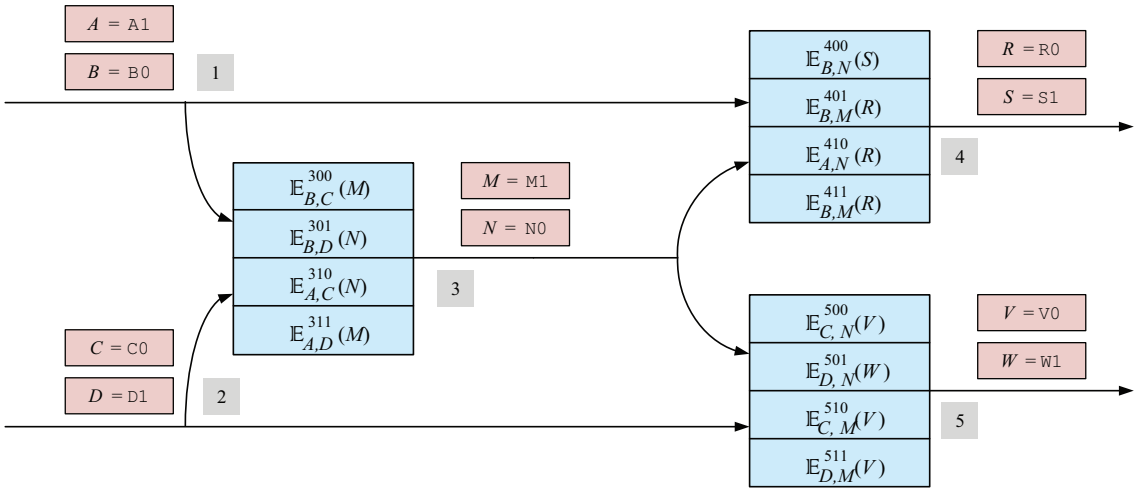

Figure 3.5.2: **Garbled circuit corresponding to the conventional circuit of Fig. 2.3.1.** For each wire i, the token with semantics 0 (that is, X_i^0) is written on top; the token with semantics 1 (that is, X_i^1) is written on bottom. Possession of token A and C, for example, lets one decrypt the third row of the leftmost garbled gate (since A ends in 1 and C ends in 0) to recover token N. The final output is the concatenation of the LSBs of the output wires.

$\mathtt{a}, \mathtt{b} \in \{0,1\}$, we mean that $g \bmod 2^{\tau(k)-2}$ is encoded as a $(\tau(k)-2)$-bit string and $\mathtt{a} \,\|\, \mathtt{b}$ is concatenated, yielding a $\tau(k)$-bit tweak. The ev function (lines 140–145) is precisely $\mathsf{ev}_{\mathsf{circ}}$; the code is repeated for completeness and to make visible the commonality with Ev (lines 150–156).

To garble a circuit, we begin selecting two tokens for each wire, one of each type. One of these will represent 0—the token is said to have *semantics* of 0—while the other will represent 1. The variable X_i^b names the token of wire i with semantics (not type!) of b. Thus the encoding function e (see lines 120–123) will map $x = x_1 \cdots x_n \in \{0,1\}^n$ to $X = (X_1^{x_1}, \ldots, X_n^{x_n})$. For each wire i that is not an output wire, we select, at line 102, random tokens of opposite type, making the association between a token's type and its semantics random. For each wire i that is an output wire, we again select random tokens of opposite types, but this time the token's type *is* the token's semantics.

Lines 104–106 compute q garbled truth tables, one for each gate g. Table $P[g,\cdot,\cdot]$ has four rows, entry $\mathtt{a}, \mathtt{b}$ the row to use when the left incoming token is of type $\mathtt{a}$ and the right incoming token is of type $\mathtt{b}$. The token that gets encrypted for this row (line 106) is the token for the outgoing-wire with the correct semantics. At lines 154–155, given two tokens X_a

and X_b we use their types to determine which row of the garbled table we need to decrypt. The description of the decoding function d (line 109) is empty because no information is needed to map an output token to its semantics, the type being the semantics.

3.5.2 Security notion for dual-key ciphers

We already defined the syntax of a DKC, a permutation $\mathbb{E}^T_{A,B}\colon \{0,1\}^k \to \{0,1\}^k$ for each A, B, T. Our definition of security will allow the adversary to select whichever of the two keys it wants to learn. We will hand it not only that key but, also, the last of the undisclosed key. (This corresponds to the type bit in runs of Garble1). We consider only nonadaptive, known-plaintext attacks. These plaintexts will be either the disclosed keys or truly random strings. We prohibit encryption cycles. During the adversary's attack, the tweaks used must be nonces—values used at most once.

More formally, the security of a DKC $\mathbb{E}\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\tau(k)} \times \{0,1\}^k \to \{0,1\}^k$ is specified using the game of Fig. 3.5.3. The game starts by choosing a bit $b \twoheadleftarrow \{0,1\}$ and a key $K \twoheadleftarrow \{0,1\}^k$. It chooses infinitely many random strings $K_1, K_2, \ldots$ such that the last bit of K_i is $i \bmod 2$. It chooses infinitely many random strings $R_1, R_2, \ldots$. Except for the last bit of K, the key K shall be kept secret. The strings $K_1, K_2, \ldots$ are initially secret, but the adversary $\mathcal{A}$ will eventually learn them through its queries. The random strings $R_1, R_2, \ldots$, used only in the "reference game" when $b = 0$, are secret. We require that the adversary $\mathcal{A}$ be nonadaptive, that is, it prepares all queries before interrogating the DKC oracle. In each query, adversary $\mathcal{A}$ has to specify an integer i indicating that it wants to use $\{K, K_i\}$ as keys of the dual-key cipher for this query, and an integer j, indicating that it wants to encrypt the string K_j. We require that $i < j$ to avoid encryption cycles. It also specifies a boolean *pos* to indicate the position, left or right, of the secret key K. Finally, it provides a tweak T, which must be a nonce. If $b = 1$ then the oracle returns the encryption of K_j to the adversary. If $b = 0$ then the oracle returns the encryption of R_j. When adversary $\mathcal{A}$ outputs a bit b' its advantage is $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}}(\mathcal{A}, k) = 2\Pr[\mathrm{DKC}^{\mathcal{A}}(k)] - 1$. We say that $\mathbb{E}$ is a secure dual-key cipher

proc INITIALIZE()	**proc** ENCRYPT(i, j, pos, T) Game DKC
$b \twoheadleftarrow \{0,1\}$, $K \twoheadleftarrow \{0,1\}^k$	**if** *used*$[T]$ **or** $i \geq j$ **then return** $\perp$
$R_1, R_2, \ldots \twoheadleftarrow \{0,1\}^k$	*used*$[T] \leftarrow$ true
for $i \in \{1, 2 \ldots\}$ **do**	**if** $pos = 1$ **then** $(A, B) \leftarrow (K, K_i)$
$K_{2i} \twoheadleftarrow \{0,1\}^{k-1}\, 0$	**else** $(A, B) \leftarrow (K_i, K)$
$K_{2i-1} \twoheadleftarrow \{0,1\}^{k-1}\, 1$	**if** $b = 1$ **then** $X \leftarrow K_j$ **else** $X \leftarrow R_j$
return lsb(K)	**return** $(K_i, K_j, \mathbb{E}^T_{A,B}(X))$

Figure 3.5.3: **Security of a dual-key cipher.** Cipher $\mathbb{E}^T_{A,B}$ has a tweak and two keys, only one of which, K, its position chosen by the adversary, is secret. The final bit of K is disclosed. Procedure FINALIZE(b') returns $(b = b')$.

if $\varepsilon(k) = \mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}}(\mathcal{A}, k)$ is negligible for every nonadaptive PPT adversary $\mathcal{A}$ whose input is 1^k and the bit returned by INITIALIZE.

DISCUSSION. By way of further explanation, ciphertexts $\mathbb{E}^{T_1}_{K,K_1}(X_1), \mathbb{E}^{T_2}_{K_2,K}(X_2), \ldots$ should be indistinguishable from random strings as long as K is secret and the tweaks $T_1, T_2, \ldots$ are nonces—even if random values K_i and X_j are all disclosed. We demand that this hold even if the last bit of K is released to the adversary and the adversary can actively choose the last bit of each K_i.

A subtle issue arises when the adversary happens to possess, say $\mathbb{E}^{T_1}_{K_1,K}(X)$ and $\mathbb{E}^{T_2}_{K_2,K}(X)$. One may be tempted to require that the two ciphertexts be indistinguishable from two independent uniformly random strings. This, however, would not allow instantiations like $\mathbb{E}^T_{A,B}(X) = E_A(E_B(X))$ for an ideal cipher E. Instead, we choose a secret $Y \twoheadleftarrow \mathcal{M}$, where $\mathcal{M}$ is the message space, and demand that the strings $\mathbb{E}^{T_1}_{K_1,K}(X)$ and $\mathbb{E}^{T_2}_{K_2,K}(X)$ be indistinguishable from $\mathbb{E}^{T_1}_{K_1,K}(Y)$ and $\mathbb{E}^{T_2}_{K_2,K}(Y)$.

The definitional intricacies for dual-key ciphers arise from wanting to require of a DKC little more than what is actually needed to prove Garble1. Too strong a definition for DKC security and interesting instantiations will be lost.

3.5.3 Security of Garble1

Our definition of DKC security suffices to prove security for Garble1. The result is stated below and proven in Section 3.5.4.

Theorem 3.5.1. If $\mathbb{E}$ is a secure dual-key cipher then $\mathcal{G} = \text{Garble1}[\mathbb{E}] \in \mathsf{GS}(\text{prv.ind}, \Phi_{\text{topo}})$.

The theorem is underlain by an explicit, blackbox, uniform reduction U such that if $\mathcal{A}(1^k)$ outputs circuits of at most r wires and fan-out at most ν, then $\mathcal{D} = U^{\mathcal{A}}$ achieves advantage $\mathbf{Adv}^{\text{dkc}}_{\mathbb{E}}(\mathcal{D}, k) \geq \frac{1}{2r}\mathbf{Adv}^{\text{prv.ind},\,\Phi_{\text{topo}}}_{\mathcal{G}}(\mathcal{A}, k)$ and makes $Q \leq 2\nu$ oracle queries, with $\mathbf{E}[Q] < 4$. It runs in time about that of $\mathcal{A}$ plus the time for $4r$ computations of $\mathbb{E}$ on k-bit keys. The small overhead implicit in the word "about" is manifest in the proof. The above assumes that $r \leq 2^{\tau(k)-2}$. In asymptotic statements, r and ν are understood as polynomials $r(k)$ and $\nu(k)$.

We comment that Garble1 does not satisfy obliviousness or authenticity. To defeat obliviousness, an adversary can just make the query (AND, OR, 00, 11) to receive (F, X), and then evaluate $Y = \mathsf{Ev}(F, X)$, returning 1 if $\mathsf{De}(\varepsilon, Y) = 1$ and 0 otherwise. This adversary has advantage 1. To defeat authenticity, an adversary can query (OR, 11), and then output $(0^k, 0^k)$. Again it has advantage 1. We will soon describe Garble2 that satisfies obliviousness and authenticity in addition to privacy.

The primitive used by Lindell and Pinkas [70] as a basis for encryption of gate rows is a randomized, IND-CPA secure symmetric encryption scheme with an *elusive* and *efficiently verifiable* range. Dual-key ciphers, in contrast, are deterministic. Our PRF-based instantiation avoids probabilistic encryption. Besides speed it results in shorter ciphertexts for each row of each gate. The additional properties of encryption assumed by LP [70] are to allow the evaluator to know which gate entry is the "correct" one. Our solution via type bits (the "point-and-permute" technique, which dates to Rogaway [85]) is well known.

3.5.4 Proof of security of Garble1

We adopt the following convention for the code-based games. Any procedure with the keyword "private" is the local code of the caller, and cannot be invoked by adversary $\mathcal{A}$. It can be viewed as a function-like macro in C/C++ programming language. That is, it still has read/write access to the variables of the caller, even if these variables are not its

parameters. In addition, any variable created by the callee still persists and is available to the caller after the callee is terminated. In this proof, the word "correct" means "as specified in game $\mathrm{PrvInd}_{\mathrm{Garble1}[\mathbb{E}],\Phi_{\mathrm{topo}}}$".

OVERVIEW. Without loss of generality, assume that $\mathcal{A}$ outputs (f_0, f_1, x_0, x_1) that satisfies $\Phi_{\mathrm{topo}}(f_0) = \Phi_{\mathrm{topo}}(f_1) = (n, m, q, A', B')$, $x_0, x_1 \in \{0,1\}^n$, and $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. We reformulate the game $\mathrm{PrvInd}_{\mathrm{Garble1}[\mathbb{E}],\Phi_{\mathrm{topo}}}$ as game Real, and specify another game Fake whose output is independent of its challenge bit; therefore $\Pr[\mathrm{Fake}^{\mathcal{A}}(k)] = 1/2$. We also describe hybrid games $\mathrm{Hy}_0, \ldots, \mathrm{Hy}_{n+q-m}$ such that the first and last hybrid games are Real and Fake respectively. We then design a DKC adversary $\mathcal{D}$ that runs $\mathcal{A}$. Informally, $\mathcal{D}$ chooses a bit $c \twoheadleftarrow \{0,1\}$ and an index $\ell \twoheadleftarrow \{1, \ldots, q+n\}$, and uses the oracle ENCRYPT to garble (f_c, x_c). When $\mathcal{A}$ halts with output c', adversary $\mathcal{D}$ returns 1 if $c' = c$. If $\ell > q+n-m$ then $\mathcal{D}$ never queries ENCRYPT; consequently, whatever $\mathcal{A}$ receives is independent of the challenge bit of game DKC, and thus $\mathcal{D}$'s advantage is 0. Suppose that $\ell \leq q+n-m$. Then, $\mathcal{D}$ aims to simulate game $\mathrm{Hy}_{\ell-1}$ if the challenge bit b of game DKC is 1, and simulate game Hy_ℓ if $b = 0$. Hence, for each fixed topological circuit (n, m, q, A', B'),

$$\begin{aligned}
\Pr\left[\,\mathrm{DKC}^{\mathcal{D}}(k) \mid b = 1\,\right] &= \frac{1}{n+q}\sum_{\ell=1}^{n+q-m}\Pr\left[\,\mathrm{Hy}^{\mathcal{A}}_{\ell-1}(k)\,\right] \\
\Pr\left[\,\neg\mathrm{DKC}^{\mathcal{D}}(k) \mid b = 0\,\right] &= \frac{1}{n+q}\sum_{\ell=1}^{n+q-m}\Pr\left[\,\mathrm{Hy}^{\mathcal{A}}_{\ell}(k)\,\right]
\end{aligned}$$

Subtracting, we bound

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}}(\mathcal{D}, k) &= \Pr\left[\,\mathrm{DKC}^{\mathcal{D}}(k) \mid b = 1\,\right] - \Pr\left[\,\neg\mathrm{DKC}^{\mathcal{D}}(k) \mid b = 0\,\right] \\
&\leq \frac{1}{n+q}\left(\Pr\left[\,\mathrm{Hy}^{\mathcal{A}}_0(k)\,\right] - \Pr\left[\,\mathrm{Hy}^{\mathcal{A}}_{n+q-m}(k)\,\right]\right) \\
&= \frac{\Pr\left[\,\mathrm{Real}^{\mathcal{A}}(k)\,\right] - 1/2}{n+q} \\
&= \frac{\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}, k)}{2(n+q)}\,.
\end{aligned}$$

```
00  proc GARBLE(f_0, f_1, x_0, x_1)                                  Game Real / Game Fake
01  (n, m, q, A', B', G) ← f_c
02  for i ∈ {1, ..., n + q} do
03      v_i ← ev(f_c, x_c, i)
04      if i ≤ n + q − m then t_i ↞ {0,1} else t_i ← v_i
05      X_i^{v_i} ↞ {0,1}^{k−1} t_i,  X_i^{v̄_i} ↞ {0,1}^{k−1} t̄_i
06  for g ∈ {n + 1, ..., n + q} do
07      a ← A'(g),  b ← B'(g),  GARB(X_g^{v_g}, 0, 0)
08      GARB$(false, 1, 0),  GARB$(false, 0, 1),  GARB$(false, 1, 1)      ⟵ Use in game Real
09      GARB$(true, 1, 0),  GARB$(true, 0, 1),  GARB$(true, 1, 1)         ⟵ Use in game Fake
10  F ← (n, m, q, A', B', P)
11  return (F, (X_1^{v_1}, ..., X_n^{v_n}), ε)
```

```
00  proc GARBLE(f_0, f_1, x_0, x_1)                                  Game Hy_ℓ
01  (n, m, q, A', B', G) ← f_c
10  for i ∈ {1, ..., n + q} do
11      v_i ← ev(f_c, x_c, i)
12      if i ≤ n + q − m then t_i ↞ {0,1} else t_i ← v_i
13      X_i^{v_i} ↞ {0,1}^{k−1} t_i,  X_i^{v̄_i} ↞ {0,1}^{k−1} t̄_i
20  for g ∈ {n + 1, ..., n + q} do
21      a ← A'(g),  b ← B'(g),  GARB(X_g^{v_g}, 0, 0)
22      GARB$(a ≤ ℓ, 1, 0),  GARB$(b ≤ ℓ, 0, 1),  Y ← GARB$(a ≤ ℓ, 1, 1)
23      if a ≤ ℓ < b and G_g(v̄_a, 0) = G_g(v̄_a, 1) then GARB(Y, 1, 0)
24  F ← (n, m, q, A', B', P)
25  return (F, (X_1^{v_1}, ..., X_n^{v_n}), ε)
```

```
30  private proc GARB(Y, α, β)
31  T ← g ‖ (t_a ⊕ α) ‖ (t_b ⊕ β)
32  A ← X_a^{v_a ⊕ α},  B ← X_b^{v_b ⊕ β}
33  P[g, t_a ⊕ α, t_b ⊕ β] ← 𝔼^T_{A,B}(Y)

40  private proc GARB$(rnd, α, β)
41  if rnd then Y ↞ {0,1}^k else Y ← X_g^{G_g(v_a ⊕ α, v_b ⊕ β)}
42  GARB(Y, α, β)
43  return Y
```

Figure 3.5.4: **Games** Real**,** Fake**, and** Hy_ℓ **(for** $0 \le \ell \le n + q - m$**) used in the proof of Theorem 3.5.1.** Each game has a procedure INITIALIZE() that samples a challenge bit $c \xleftarrow{\$} \{0,1\}$. All variables are global. Each game has local procedures GARB and GARB$ to which the adversary $\mathcal{A}$ has no access. The procedure FINALIZE(c') of each game returns $(c = c')$. At line 03 we let $\mathsf{ev}(f, x, i)$ return the bit value of wire i in the evaluation of f on input x.

GAME REAL. Consider game Real in Fig. 3.5.4. We claim that it coincides with game $\mathrm{PrvInd}_{\mathrm{Garble1}[\mathbb{E}],\Phi_{\mathrm{topo}}}$. To justify this, recall that in the Garble1 scheme, each wire i carries tokens X_i^0 and X_i^1 with semantics 0 and 1 respectively. If wire i ends up having value (semantics) v_i in the computation $y \leftarrow \mathsf{ev}(f_c, x_c)$, where c is the challenge bit of game $\mathrm{PrvInd}_{\mathrm{Garble1},\Phi_{\mathrm{topo}}}$, then token $X_i^{v_i}$ becomes visible to the adversary while $X_i^{\overline{v}_i}$ stays invisible. Game Real makes this explicit. It picks for each wire i a "visible" token and an "invisible" one. It then ensures that the tokens the adversary gets are the visible ones. Procedure $\mathsf{ev}(f, x, i)$ at line 03 returns the bit value of wire i in the evaluation of circuit f on input x.

Formally,

proc $\mathsf{ev}(f, x, i)$

$(n, m, q, A, B, G) \leftarrow f$

for $g \leftarrow n+1$ **to** $n+q$ **do** $a \leftarrow A(g)$, $b \leftarrow B(g)$, $x_g \leftarrow G_g(x_a, x_b)$

return x_i

Let us give the high-level description of procedures Garb and Garb\$. Let t_i be the last bit of the visible token at wire i. If one has all visible tokens then one can open $P[g, t_a, t_b]$ for every gate g, where a and b are the first and second incoming wires of g respectively. Procedure $\textsc{Garb}(Y, \alpha, \beta)$ writes to row $P[g, t_a \oplus \alpha, t_b \oplus \beta]$. (As a "private" procedure, it inherits variables g, a, b, and $t_1, \ldots, t_{n+q}$ from its caller.) The written value is the encryption of Y, instead of the correct token, but with the correct keys and tweak. On the other hand, procedure $\textsc{Garb}\$(\mathit{rnd}, \alpha, \beta)$ uses the correct keys and tweak to build $P[g, t_a \oplus \alpha, t_b \oplus \beta]$. If $\mathit{rnd} = \mathsf{false}$ then the plaintext is the correct token as well. Otherwise, it is a uniformly random string. This plaintext, real or random, will be handed to the caller of Garb\$.

Game Fake. Consider game Fake in Fig. 3.5.4. The game is identical to Real at garbled rows that may be opened by the visible tokens. For other garbled rows, it sets the plaintexts in those rows to be independent random strings instead of the correct tokens. (In the code, we always enable the flag *rnd* of procedure Garb\$ whenever we call it.) We claim that game Fake's output is independent of its challenge bit c. To justify this, from the topological circuit f^- and the final output $y_1 \cdots y_m = y = \mathsf{ev}(f_c, x_c)$, which are independent of c, we can rewrite game Fake as in Fig. 3.5.5. There, we refer to the visible token of wire i as V_i, and its invisible counterpart as I_i, omitting the semantics of these tokens. Plaintexts Y are random, except for garbled rows that can be opened by visible tokens.

Hybrids. Now consider the hybrid games Hy_ℓ of Fig. 3.5.4, defined for $0 \le \ell \le n+q-m$. For better readability, we describe them in two equivalent ways. We first give the recursive approach: game Hy_0 coincides with game Real, and we will describe how to go to game Hy_ℓ

for $i \in \{1, \ldots, n+q\}$ **do**
  if $i \le n+q-m$ **then** $t_i \twoheadleftarrow \{0,1\}$ **else** $t_i \leftarrow y_{i-(n+q-m)}$
  $V_i \twoheadleftarrow \{0,1\}^{k-1} t_i, \quad I_i \twoheadleftarrow \{0,1\}^{k-1} \bar{t}_i$
for $g \in \{n+1, \ldots, n+q\}$ **do**
  $a \leftarrow A'(g), \quad b \leftarrow B'(g)$
  for $(A, B) \in \{V_a, I_a\} \times \{V_b, I_b\}$ **do**
    if $A = V_a$ **and** $B = V_b$ **then** $Y \leftarrow V_g$ **else** $Y \twoheadleftarrow \{0,1\}^k$
    $T \leftarrow g \parallel \mathrm{lsb}(A) \parallel \mathrm{lsb}(B), \quad P[g, \mathrm{lsb}(A), \mathrm{lsb}(B)] \leftarrow \mathbb{E}^T_{A,B}(Y)$
$F \leftarrow (n, m, q, A', B', P)$
return $(F, (V_1, \ldots, V_n), \varepsilon)$

Figure 3.5.5: **Rewritten game** Fake **of the proof of Theorem 3.5.1.** This game depends solely on topological circuit $f^- = (n, m, q, A', B')$ and the output $v = \mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$.

from game $\mathrm{Hy}_{\ell-1}$, for every $\ell \in \{1, \ldots, n+q-m\}$. This will help explain the strategy of our constructed DKC adversary. Alternatively, we describe each hybrid game directly, which explains how to write the code.

We first give the recursive construction. In each game, every garbled row always has the correct keys and tweak. Fix $\ell \in \{1, \ldots, n+q-m\}$. In game Hy_ℓ, first run the code of game $\mathrm{Hy}_{\ell-1}$, and then do the following update Δ_ℓ. For each token, associate it to a fresh uniformly random k-bit string. The two games $\mathrm{Hy}_{\ell-1}$ and Hy_ℓ differ only at garbled rows that use the invisible token of wire ℓ as a key. Consider such a row. If the current plaintext is a token then replace it with its associated random string above. Otherwise, sample a fresh uniformly random k-bit string, and let it be the new plaintext. See Fig. 3.5.6 for illustration. In other words, for each gate g, if we hop on the chain $\Delta_1, \ldots, \Delta_{n+q-m}$, we'll visit g exactly twice, the first time in Δ_a, and the second time in Δ_b, where a and b are the first and second incoming wires of g respectively. In the first visit, we modify rows $P[g, \bar{t}_a, 0]$ and $P[g, \bar{t}_a, 1]$, changing the plaintexts from two tokens to random strings—the latter will be identical if the former are the same, namely $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$, otherwise they will be independent. In the second visit, we modify rows $P[g, t_a, \bar{t}_b]$ and $P[g, \bar{t}_a, \bar{t}_b]$, changing their plaintexts from a token and a random string respectively to two fresh, independent random strings.

Let us move on to the direct construction. Fix $\ell \in \{0, \ldots, n+q-m\}$. We will describe game Hy_ℓ. Each garbled row will be built from the correct keys and tweak. For rows that

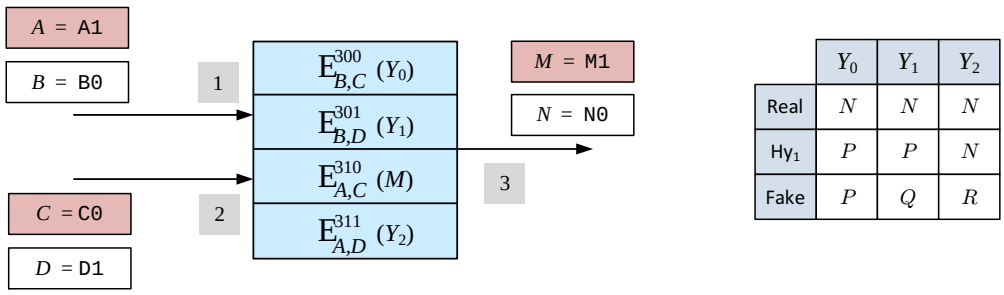

Figure 3.5.6: **Garbled circuits of hybrid games, assuming that we garble** $(f_c, x_c) = (\mathrm{OR}, 00)$**.** There are three games: $\mathrm{Real}(\mathrm{Hy}_0), \mathrm{Hy}_1$, and $\mathrm{Fake}(\mathrm{Hy}_2)$. On the left, we draw the common form of the garbled circuit for these games. The keys and tweak for each row are the same for every game, but the plaintexts Y_0, Y_1, Y_2 will be different. For each wire i, the token with semantics 0, that is, X_i^0 is written on top; the token with semantics 1, that is X_i^1 is written on bottom; visible tokens are colored. The table on the right tells what Y_0, Y_1, Y_2 are, for each game; strings P, Q, and R are uniformly random. For example, the cell at the second row and second column indicates that in game Real, plaintext Y_0 is token N of wire 3.

can be opened by visible tokens, their plaintexts are always the correct tokens. For other rows, consider a gate g with first and second incoming wires a and b respectively. Note that in $\Delta_1, \ldots, \Delta_{n+q-m}$, the first update to $P[g, \bar{t}_a, 0]$ and $P[g, \bar{t}_a, 1]$ is in Δ_a. Hence if $a \le \ell$ then the plaintexts of these two rows will be the correct tokens. Else they are random strings—identical if $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$ and $\ell < b$, and independent otherwise. Likewise, if $b \le \ell$ then the plaintext in row $P[g, t_a, \bar{t}_b]$ is the correct token, otherwise it is a random string independent of anything else.

We claim that game Hy_{n+q-m} coincides with game Fake. It is easily verified, as when $\ell = n + q - m$, at line 22, procedure GARB\$ is always invoked with $rnd = \mathsf{true}$, and line 23 is never executed.

DKC ADVERSARY. Adversary $\mathcal{D}$, given 1^k and a bit τ from procedure INITIALIZE(), runs $\mathcal{A}(1^k)$. When the latter makes a GARBLE(f_0, f_1, x_0, x_1) query, it replies via the code of Fig. 3.5.7. Recall that $\mathcal{D}$ chooses $c \twoheadleftarrow \{0,1\}$ and an index $\ell \twoheadleftarrow \{1, \ldots, q+n\}$, and uses the oracle ENCRYPT to garble (f_c, x_c). When $\mathcal{A}$ halts with output c', adversary $\mathcal{D}$ returns 1 if $c' = c$. If $\ell > q + n - m$ then $\mathcal{D}$ never queries ENCRYPT, as lines 22–23 never get executed. Consequently, whatever $\mathcal{A}$ receives is independent of the challenge bit of game DKC, and thus $\mathcal{D}$'s advantage is 0. Suppose that $\ell \le q + n - m$. Then, $\mathcal{D}$ aims to simulate game $\mathrm{Hy}_{\ell-1}$

if the challenge bit of game DKC is 1, and simulate game Hy_ℓ otherwise. Below, we will give a high-level description of the code of $\mathcal{D}$.

Initially, the adversary $\mathcal{D}$ picks the types t_i for every wire $i \neq \ell$ as in game Real. We want the key K of game DKC to play the role of the invisible token of wire ℓ, so t_ℓ is the complement of the bit τ given from procedure FINALIZE() of game DKC. Adversary $\mathcal{D}$ first walks through gates that ℓ is an incoming wire. It will query the oracle ENCRYPT (via procedure QUERY) to write to garbled rows that are supposed to use the invisible token at wire ℓ as a key; these rows are determined by τ.

We need make sure that in both games $\text{Hy}_{\ell-1}$ and Hy_ℓ, there is no garbled row that uses the invisible token of wire ℓ as its plaintext. This claim is obvious if $\ell \leq n$, namely, wire ℓ is an input wire. If $\ell > n$ then due to the topological ordering of gates, both incoming wires of gate ℓ must stay in the set $\{1, \ldots, \ell - 1\}$, and the sequence $\Delta_1, \ldots, \Delta_{\ell-1}$ therefore must change the plaintexts in all rows of gate ℓ that can't be opened by visible tokens to random strings, expelling the invisible token of wire ℓ.

We now give a high-level description of the "private" procedure $\text{QUERY}(rnd, \alpha, \beta)$. The assumption is that ℓ must be one of the incoming wires a and b of gate g. This procedure will write to the row $P[g, t_a \oplus \alpha, t_b \oplus \beta]$; the keys and tweak of this row are always correct; the flag *rnd* will indicate if the plaintext, in game $\text{Hy}_{\ell-1}$, is the correct token ($rnd = \mathsf{false}$) or a random string ($rnd = \mathsf{true}$). The written value Z is obtained from the answer (K_i, K_j, Z) of ENCRYPT; we will describe how to choose i and j for querying later. Then Z is the encryption of either K_j (if the challenge bit of game DKC is 1) or a random string R_j (if the challenge bit is 0), and the keys will be K and K_i. Let $\{w\} = \{a, b\} \backslash \{\ell\}$. Recall that $\mathcal{D}$ did not initialize the tokens except the types. Assign value K_i to the token of wire w that is a correct key of $P[g, t_a \oplus \alpha, t_b \oplus \beta]$; let t be the type of this token. As the type of K_i is $i \bmod 2$, initially, choose $i = 2w + t$. If *rnd* is false then we want to assign K_j to the token of wire g that is the correct plaintext of $P[g, t_a \oplus \alpha, t_b \oplus \beta]$; let the type of this token be t'. Then, choose $j = 2g + t'$. On the other hand, if *rnd* is true then we just want K_j to be a

```
00  proc GARBLE(f_0, f_1, x_0, x_1)                    // as defined by adversary D
01  c ←$ {0,1},  (n, m, q, A', B', G) ← f_c,  ℓ ←$ {1, ..., q + n}
10  for i ∈ {1, ..., n + q} do t_i ← v_i ← ev(f_c, x_c, i)
11  for i ← {1, ..., n + q − m} do t_i ←$ {0,1}
12  for g ∈ {n + 1, ..., n + q} do
13    a ← A'(g),  b ← B'(g),   t_ℓ ← τ̄ //τ = lsb(K)
14    if a = ℓ then QUERY(false, 1, 0),  QUERY(false, 1, 1)
15    if b = ℓ then QUERY(false, 0, 1),  Y_g ← QUERY(true, 1, 1)
20  for i ∈ {1, ..., n + q} do
21    if X_i^{v̄_i} = ⊥ and i ≠ ℓ then X_i^{v̄_i} ←$ {0,1}^{k−1} t̄_i
22    if X_i^{v_i} = ⊥ then X_i^{v_i} ←$ {0,1}^{k−1} t_i
30  for g ∈ {n + 1, ..., n + q} do
31    a ← A'(g),  b ← B'(g),  GARB(X_g^{v_g}, 0, 0)
32    if a ≠ ℓ and b ≠ ℓ then
33      GARB$(a ≤ ℓ, 1, 0),  GARB$(b ≤ ℓ, 0, 1),  Y ← GARB$(a ≤ ℓ, 1, 1)
34      if a ≤ ℓ < b and G_g(v̄_a, 0) = G_g(v̄_a, 1) then GARB(Y, 1, 0)
35    elsif a = ℓ then GARB$(false, 0, 1)
36    else GARB$(true, 1, 0),  if G_g(v̄_a, 0) = G_g(v̄_a, 1) then GARB(Y_g, 1, 0)
37  F ← (n, m, q, A', B', P)
38  return (F, (X_1^{v_1}, ..., X_n^{v_n}), ε)
```

```
40  private proc QUERY(rnd, α, β)
41  T ← g ‖ (t_a ⊕ α) ‖ (t_b ⊕ β),  γ ← v_g ⊕ G_g(v_a ⊕ α, v_b ⊕ β)
42  if a = ℓ then pos ← 1,  i ← 2b + (t_b ⊕ β) else pos ← 0,  i ← 2a + (t_a ⊕ α)
43  if rnd then j ←$ {2(g + n + q), 2(g + n + q) + 1} else j ← 2g + (t_g ⊕ γ)
44  (K_i, K_j, Z) ← ENCRYPT(i, j, pos, T),  P[g, t_a ⊕ α, t_b ⊕ β] ← Z
45  if a = ℓ then X_b^{v_b ⊕ β} ← K_i else X_a^{v_a ⊕ α} ← K_i
46  if ¬rnd then X_g^{v_g ⊕ γ} ← K_j
47  return K_j
```

Figure 3.5.7: **Constructed DKC adversary $\mathcal{D}$.** Procedure GARBLE used by adversary $\mathcal{D}$ attacking $\mathbb{E}$, based on the adversary $\mathcal{A}$ attacking the prv.ind-security of Garble1. All variables are global. Adversary $\mathcal{D}$ also makes use of procedures GARB and GARB\$ in Fig. 3.5.4. Adversary $\mathcal{A}$ has no access to procedure QUERY that is a local procedure of $\mathcal{D}$. At line 13, the bit τ is the last bit of the key K of game DKC given to $\mathcal{D}$ by INITIALIZE().

fresh random string, so pick $j \xleftarrow{\$} \{2(n+q+g), 2(n+q+g)+1\}$.

How should $\mathcal{D}$ call QUERY? If $\ell = a$ then we want to write to rows $P[g, \bar{t}_a, 0]$ and $P[g, \bar{t}_a, 1]$. Let $rnd = \mathsf{false}$ for both of them. Consequently, if the challenge bit of game DKC is 1 then the plaintexts of two rows above are the correct tokens, which is what we need for game $\mathrm{Hy}_{\ell-1}$, since $\mathrm{Hy}_{\ell-1}$ doesn't modify these rows of gate g. If, on the other hand, the challenge bit is 0 then from the description of game DKC, the plaintexts of two rows above are random strings—either independent or identical, depending on whether the two tokens in game $\mathrm{Hy}_{\ell-1}$ are different or the same. This gives what we need to construct game Hy_{ℓ}. Now

consider the case $\ell = b$. We want to write to rows $P[g, t_a, \bar{t}_b]$ and $P[g, \bar{t}_a, \bar{t}_b]$. For the former, similarly, let $rnd = \mathsf{false}$. For the latter, note that in both games, the plaintext of this row is a random string. Moreover, we claim that this string is independent of the plaintext of $P[g, t_a, \bar{t}_b]$. Our claim is true for game Hy_ℓ, as Δ_b replaces the two plaintexts from a token and a random string to two fresh, independent random strings. It is also true for game $\mathrm{Hy}_{\ell-1}$, as the plaintext of row $P[g, t_a, \bar{t}_b]$ is a token. Hence let $rnd = \mathsf{true}$. We obtain from the oracle ENCRYPT a random string Y_g that will be the plaintext of row $P[g, \bar{t}_a, \bar{t}_b]$ if the challenge bit of game DKC is 1. Save it for a later use.

By calling QUERY, adversary $\mathcal{D}$ creates some tokens. The next step is to sample the other tokens according to their types, except for the invisible token of wire ℓ. Now manually construct the still vacant rows as follows. For gates that ℓ is not an incoming wire, as the two games $\mathrm{Hy}_{\ell-1}$ and Hy_ℓ agree on rows of this gate, follow the code of game Hy_ℓ. Consider another gate g of first and second incoming wires a and b respectively, with $\ell \in \{a, b\}$. each row has correct keys and tweaks. If a row can be opened by visible tokens then its plaintext is also the correct token. Otherwise, if $\ell = a$ then the only still vacant row is $P[g, t_a, \bar{t}_b]$, whose plaintext is also the correct token in both games $\mathrm{Hy}_{\ell-1}$ and Hy_ℓ. If $\ell = b$ then the only vacant row is $P[g, \bar{t}_a, t_b]$, whose plaintext is random in both games $\mathrm{Hy}_{\ell-1}$ and Hy_ℓ. But, is this random string independent of anything else or must it be a prior random string? The latter happens in game $\mathrm{Hy}_{\ell-1}$ if $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$, as the plaintexts in rows $P[g, \bar{t}_a, 0]$ and $P[g, \bar{t}_a, 1]$ are identical. If so, recall that previously, we saved a random string Y_g. If the challenge bit of game DKC is 1 then Y_g is the plaintext of row $P[g, \bar{t}_a, \bar{t}_b]$. Otherwise Y_g is independent of anything else. Now, let Y_g be the plaintext of row $P[g, \bar{t}_a, t_b]$. This yields the intended construction for both games.

Having constructed $\mathcal{D}$, we now argue that it is nonadaptive because (i) the only way that $\mathcal{D}$ can query ENCRYPT is via QUERY and in the body of QUERY, we don't make use of the prior answers of ENCRYPT, (ii) the QUERY calls are deterministic for a fixed ℓ, and (iii) $\mathcal{D}$ creates the types before using QUERY.

RESOURCES ACCOUNTING. Let Q be the random variable denoting the number of queries of $\mathcal{D}$ to the oracle ENCRYPT. Fix the topological circuit (n, m, q, A, B). Let ν_i be the number of gates that wire i is an incoming wire. Hence $\nu_i \leq \nu$, and

$$\sum_{i=1}^{n+q-m} \nu_i = 2q,$$

as both sides of this equality count the total number of incoming wires of all gates. Note that the random variable Q is uniformly distributed over the $(q+n)$-element multiset $\{2\nu_1, \ldots, 2\nu_{n+q-m}, 0, \ldots, 0\}$. Hence $Q \leq 2\nu$, and

$$\mathbf{E}[Q] = \frac{1}{n+q} \sum_{i=1}^{n+q-m} 2\nu_i = \frac{4q}{q+n} < 4 \ .$$

3.5.5 Dual-key ciphers from a PRF

Our primary interest will be in instantiating a dual-key cipher via a PRF. Let F associate to key $K \in \{0,1\}^{k-1}$ a map $\mathsf{F}_K \colon \{0,1\}^{\tau(k)} \to \{0,1\}^k$. We require that the map $K, T \mapsto \mathsf{F}_K(T)$ be polynomial-time computable. We refer to τ as the input length.

The prf-advantage of an adversary $\mathcal{D}$ against F is defined as

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{D}, k) = 2\Pr[\mathrm{PRF}^{\mathcal{D}}_{\mathsf{F}}(k)] - 1,$$

where game $\mathrm{PRF}_{\mathsf{F}}$ is as follows. INITIALIZE picks a random bit b and a random $(k-1)$-bit key K. The adversary has access to procedure FN that maintains a table $\mathtt{Tbl}[\cdot]$ initially everywhere undefined. Given $T \in \{0,1\}^{\tau(k)}$, the procedure returns $\mathsf{F}(K, T)$ if $b = 1$. Otherwise, it picks and returns $\mathtt{Tbl}[T] \twoheadleftarrow \{0,1\}^k$ if $\mathtt{Tbl}[T] = \perp$, or returns $\mathtt{Tbl}[T]$ if $\mathtt{Tbl}[T] \neq \perp$. FINALIZE$(b')$ returns $(b = b')$. We say that F is PRF-secure if $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{D}, \cdot)$ is negligible for all polynomial-time adversaries $\mathcal{D}$.

Given a PRF F as above, we define the DKC $\mathbb{E}$ via

$$\mathbb{E}^T_{A,B}(P) = \mathsf{F}_{A[1:k-1]}(T) \oplus \mathsf{F}_{B[1:k-1]}(T) \oplus P.$$

This dual-key cipher has tweak length τ and is denoted $\mathbb{E}[\mathsf{F}]$. During evaluation, token

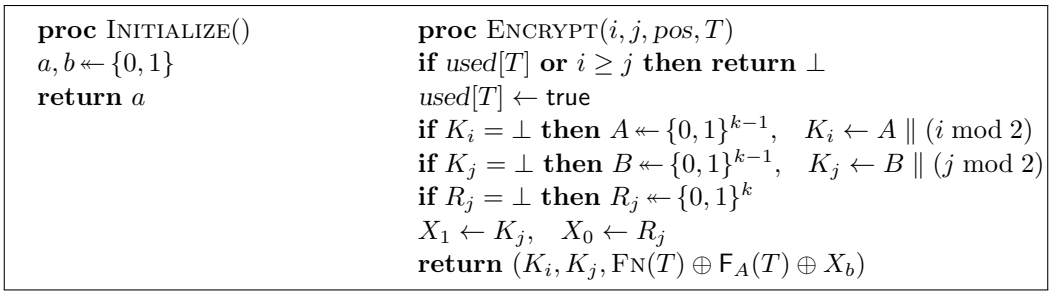

Figure 3.5.8: **Proof of Theorem 3.5.2.** The code is for the adversary $\mathcal{B}$, which has a PRF oracle Fn.

types are revealed, but the entire key of F remains secret.

The following result establishes that $\mathbb{E}[\mathsf{F}]$ is a good DKC when F is a good PRF. The reduction is tight and explicit. More specifically, the proof provides a blackbox reduction such that for any adversary $\mathcal{A}(1^k)$ attacking $\mathbb{E}[\mathsf{F}]$ there is an adversary $\mathcal{B}(1^k)$ attacking F for which $\mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(\mathcal{B}, k) = 0.5\,\mathbf{Adv}^{\text{dkc}}_{\mathbb{E}[\mathsf{F}]}(\mathcal{A}, k)$. If $\mathcal{A}$ makes Q queries to Encrypt then $\mathcal{B}$ also makes Q queries to the PRF oracle Fn. The running time of $\mathcal{B}$ is about that of $\mathcal{A}$, where the meaning of "about" is manifest in the proof that follows the theorem.

Theorem 3.5.2. Let F be a PRF. Then $\mathbb{E}[\mathsf{F}]$ is a secure dual-key cipher.

Proof. Fix an adversary $\mathcal{A}$ attacking $\mathbb{E}[\mathsf{F}]$. Consider the following adversary $\mathcal{B}$ attacking F. Adversary $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$, and follows the code of Fig. 3.5.8. In words, initially, $\mathcal{B}$ samples $b \twoheadleftarrow \{0,1\}$. For each query (i, j, pos, T), if one of K_i, K_j, or R_j is not defined, it is sampled according to the distribution specified in game DKC. Then, $\mathcal{B}$ returns $\text{Fn}(T) \oplus \mathsf{F}_A(T) \oplus X_b$ to $\mathcal{A}$, where $A = K_j[1 : k-1], X_0 = R_j$, and $X_1 = K_j$. Finally, when $\mathcal{A}$ outputs a bit b', adversary $\mathcal{B}$ will output 1 only if $b' = b$. Then

$$\Pr[\text{PRF}^{\mathcal{B}}(k) \mid c = 1] = \Pr[\text{DKC}^{\mathcal{A}}(k)] \text{ and } \Pr[\neg\text{PRF}^{\mathcal{B}}(k) \mid c = 0] = 1/2$$

where c is the challenge bit of game PRF. To justify the second claim, note that if $c = 0$ then $\text{Fn}(T) \oplus \mathsf{F}_A(T) \oplus X_b$ is a uniformly random string independent of X_b, and thus the answers to $\mathcal{A}$'s queries are independent of b. Subtracting, we obtain $\mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(\mathcal{B}, k) = \frac{1}{2}\mathbf{Adv}^{\text{dkc}}_{\mathbb{E}[\mathsf{F}]}(\mathcal{A}, k)$.

□

The instantiation of a DKC $\mathbb{E}$ by way of $\mathbb{E}[\mathsf{F}]$ is by no means the only reasonable instantiation, nor the only one that can be proven secure. We now investigate further instantiations, going all the way to a blockcipher.

3.5.6 Dual-key ciphers from double encryption

We also prove the dkc-security of the instantiation $\mathbb{E}[E]$ with $\mathbb{E}^T_{A,B}(X) = E_A(E_B(X))$, where E is an ideal cipher. In the theorem below, we will show that if an adversary $\mathcal{A}$ makes Q queries to the ENCRYPT oracle, and q_E queries to E and E^{-1} then $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}[E]}(\mathcal{A}, k) \leq (10Q^2 + 4Q + 8q_E)/2^k$. The above assumes that $Q + q_E \leq 2^{k-3}$.

Theorem 3.5.3. Let E be an ideal cipher. Then $\mathbb{E}[E]$ is a secure dual-key cipher.

Proof. Consider games G_0–G_5 in Figures 3.5.9 and 3.5.10. In each game, adversary $\mathcal{A}$ has indirect access to E and E^{-1} by calling procedures ENC and DEC. Game G_0 corresponds to game DKC, with strings K_i and R_i lazily sampled. Suppose that $\mathcal{A}(1^k)$ makes q_E queries to ENC and DEC, and Q queries to ENCRYPT, with $q_E + Q \leq 2^{k-3}$.

We explain the game chain up until the terminal game. $\triangleright G_0 \to G_1$: Instead of sampling keys K and K_i independently, we'll sample so that they are pairwise distinct. The two games are identical until either sets *bad*. Moreover,

$$\Pr[\mathsf{BAD}(G_0^{\mathcal{A}}(k))] = \sum_{\ell=1}^{2Q} \ell/2^{k-1} = 2Q(2Q+1)/2^k = (4Q^2 + 2Q)/2^k \ .$$

$\triangleright G_1 \to G_2$: Instead of calling $E_K(\cdot)$ and $E_K^{-1}(\cdot)$, we lazily implement an ideal permutation π. In addition, we keep track of prior answers by an array H so that we can answer for "redundant" queries without using π as follows. For each ENCRYPT query $(i, j, 1, T)$, we use the array H to store $H[(i, j)] \leftarrow Y$, where Y is the answer to $\mathcal{A}$. Later, if $\mathcal{A}$ makes another query $(i, j, 1, T^*)$, we immediately return Y without looking up π. Likewise, for each

```
00 proc INITIALIZE()
01 b ←$ {0,1},  K ←$ {0,1}^k
02 Keys ← {K}
03 return lsb(K)

04 proc ENC(A, B)
05 return E_A(B)

06 proc DEC(A, B)
07 return E_A^{-1}(B)
```

```
10 proc ENCRYPT(i, j, pos, T)        Game G_0 [G_1]
11 if used[T] or i ≥ j then return ⊥
12 used[T] ← true,  s ← i mod 2,  t ← j mod 2
13 if K_i = ⊥ then
14    K_i ←$ {0,1}^{k−1} s
15    if K_i ∈ Keys then
16       bad ← true,  [K_i ←$ {0,1}^{k−1} s \Keys]
17    Keys ← Keys ∪ {K_i}
18 if K_j = ⊥ then
19    K_j ←$ {0,1}^{k−1} t
20    if K_j ∈ Keys then
21       bad ← true,  [K_j ←$ {0,1}^{k−1} t \Keys]
22    Keys ← Keys ∪ {K_j}
23 if R_j = ⊥ then R_j ←$ {0,1}^k
24 X_1 ← K_j,  X_0 ← R_j
25 if pos = 1 then X ← E_{K_i}(X_b) else X ← X_b
26 Y ← E_K(X)
27 if pos = 1 then return (K_i, K_j, Y)
28 else return (K_i, K_j, E_{K_i}(Y))
```

```
30 proc INITIALIZE()
31 b ←$ {0,1}^k,  K ←$ {0,1}^k
32 Keys ← {K}
33 return lsb(K)

34 proc ENC(A, B)
35 if A ≠ K then return E_A(B)
36 bad ← true, [return ⊥]
37 if B ∉ Dom(π) then
38    Y ←$ {0,1}^k \ Ran(π),  π[B] ← Y
39 return π[B]

40 proc DEC(A, B)
41 if A ≠ K then return E_A^{-1}(B)
42 bad ← true, [return ⊥]
43 if B ∉ Ran(π) then
44    X ←$ {0,1}^k \ Dom(π),  π[X] ← B
45 return π^{-1}[B]
```

```
50 proc ENCRYPT(i, j, pos, T)        Game G_2 [G_3]
51 if used[T] or i ≥ j then return ⊥
52 used[T] ← true,  s ← i mod 2,  t ← j mod 2
53 if K_i = ⊥ then
54    K_i ←$ {0,1}^{k−1} s \Keys,  Keys ← Keys ∪ {K_i}
55 if K_j = ⊥ then
56    K_j ←$ {0,1}^{k−1} t \Keys,  Keys ← Keys ∪ {K_j}
57 if R_j = ⊥ then R_j ←$ {0,1}^k
58 X_1 ← K_j,  X_0 ← R_j
59 if pos = 1 and Y ← H[(i, j)] ≠ ⊥ then
60    return (K_i, K_j, Y)
61 if pos = 0 and Y ← H[j] ≠ ⊥ then
62    return (K_i, K_j, E_{K_i}(Y))
63 if pos = 1 then X ← ENC(K_i, X_b) else X ← X_b
64 if X ∉ Dom(π) then
65    Y ←$ {0,1}^k \ Ran(π),  π[X] ← Y
66 Y ← π[X]
67 if pos = 1 then
68    H[(i, j)] ← Y,  return (K_i, K_j, Y)
69 else H[j] ← Y,  return (K_i, K_j, E_{K_i}(Y))
```

Figure 3.5.9: **Games for the proof of Theorem 3.5.3.** Procedure FINALIZE(b') returns ($b = b'$). Games G_1 and G_3 include the corresponding boxed statements, but games G_0 and G_2 do not.

ENCRYPT query $(i, j, 0, T)$, we store $H[j] \leftarrow Y$, where $E_{K_i}(Y)$ is the answer to $\mathcal{A}$. Later, if $\mathcal{A}$ makes another query $(i^*, j, 0, T^*)$, we immediately return $E_{K_{i^*}}(Y)$ without using π. The changes are conservative.

$\triangleright G_2 \rightarrow G_3$: we take away from the adversary the power of querying ENC$(K, \cdot)$ and

```
70 proc Initialize()
71 b ←$ {0,1}^k,  K ←$ {0,1}^k
72 Keys ← {K}
73 return lsb(K)

74 proc Enc(A, B)
75 if A ≠ K then return E_A(B)
76 return ⊥

77 proc Dec(A, B)
78 if A ≠ K then return E_A^{-1}(B)
79 return ⊥

80 proc Encrypt(i, j, pos, T)        Game [G_4] G_5
81 if used[T] or i ≥ j then return ⊥
82 used[T] ← true,  s ← i mod 2,  t ← j mod 2
83 if K_i = ⊥ then
84    K_i ←$ {0,1}^{k-1} s \Keys,  Keys ← Keys ∪ {K_i}
85 if K_j = ⊥ then
86    K_j ←$ {0,1}^{k-1} t \Keys,  Keys ← Keys ∪ {K_j}
87 if R_j = ⊥ then R_j ←$ {0,1}^k
88 X_1 ← K_j,  X_0 ← R_j
89 if pos = 1 and Y ← H[(i,j)] ≠ ⊥ then
90    return (K_i, K_j, Y)
91 if pos = 0 and Y ← H[j] ≠ ⊥ then
92    return (K_i, K_j, E_{K_i}(Y))
93 if pos = 1 then X ← E_{K_i}(X_b) else X ← X_b
94 Y ←$ {0,1}^k
95 if X ∉ Dom(π) then π[X] ← Y
96 else bad ← true,  [Y ← π[X]]
97 if pos = 1 then
98    H[(i,j)] ← Y,  return (K_i, K_j, Y)
99 else H[j] ← Y,  return (K_i, K_j, E_{K_i}(Y))
```

Figure 3.5.10: **Games for the proof of Theorem 3.5.3.** Procedure Finalize(b') returns $(b = b')$. Game G_4 includes the corresponding boxed statements, but game G_5 does not.

Dec$(K, \cdot)$. The two games are identical until game G_3 sets *bad*. Consider a query to Enc or Dec. Since all strings K_i are different from K, the last bit of K is public, and each prior query to Enc or Dec removes at most a value for K, there are at least $2^{k-1} - 2Q - q_E$ equally likely values for K. Hence,

$$\Pr[\mathsf{BAD}(G_2^{\mathcal{A}}(k))] \leq q_E/(2^{k-1} - 2Q - q_E) \leq q_E/2^{k-2}$$

where the second inequality is due to the assumption that $Q + q_E \leq 2^{k-3}$.

$\triangleright G_3 \to G_4$: Instead of implementing π as an ideal permutation, we implement it as an ideal function. By PRP/PRF Switching Lemma, $\Pr[G_3^{\mathcal{A}}(k)] - \Pr[G_4^{\mathcal{A}}(k)] \leq Q(Q-1)/2^{k+1}$.

$\triangleright G_4 \to G_5$: Instead of calling $Y \leftarrow \pi[X]$, we sample Y uniformly. The two games are identical until G_4 sets *bad*. Consider the ℓth Encrypt query (i, j, pos, T), and let X be the string defined at line 93 on this query. For game G_4 to set *bad*, if $pos = 1$ then there must be no prior query $(i, j, 1, T^*)$, and if $pos = 0$ then there must be no prior query $(i^*, j, 0, T^*)$; otherwise in this query, line 96 is unreachable and *bad* won't be set. The flag *bad* is triggered

only if $X \in \mathrm{Dom}(\pi)$, where $\mathrm{Dom}(\pi)$ is the set of the points that $\pi[\cdot]$ is defined prior to this query. For each $P \in \mathrm{Dom}(\pi)$, we claim that the chance that $X = P$ is at most 2^{-k}. Hence by union bound,

$$\Pr[\mathsf{BAD}(G_4^{\mathcal{A}}(k))] \leq \sum_{\ell=1}^{Q} (\ell - 1)/2^k = Q(Q-1)/2^{k+1} \ .$$

We can justify the claim above by the following tedious case analysis. Suppose that P was added to $\mathrm{Dom}(\pi)$ by $\mathcal{A}$'s querying (i^*, j^*, pos^*, T^*).

Case 1: $X = E_{K_i}(K_j)$ and $P = E_{K_{i^*}}(K_{j^*})$. If $i = i^*$ then as mentioned above, $j \neq j^*$ so that we can reach line 96 to set *bad*, and thus $K_j \neq K_{j^*}$. Then $X \neq P$ because E_{K_i} is a permutation. On the other hand, if $i \neq i^*$ then $K_i \neq K_{i^*}$. Then $\Pr[X = P] = 2^{-k}$, since E_{K_i} and $E_{K_{i^*}}$ are independent ideal permutations, and $\mathcal{A}$ is nonadaptive.

Case 2: $X = E_{K_i}(R_j)$ and $P = E_{K_{i^*}}(R_{j^*})$. If $i = i^*$ then as mentioned above, $j \neq j^*$ so that we can reach line 96 to set *bad*. Then $X = P$ only if $R_j = R_{j^*}$, because E_{K_i} is a permutation. However, $\Pr[R_j = R_{j^*}] = 2^{-k}$, since we sample R_j and R_{j^*} independently, and $\mathcal{A}$ is nonadaptive. On the other hand, if $i \neq i^*$ then $K_i \neq K_{i^*}$. Hence $\Pr[X = P] = 2^{-k}$, since E_{K_i} and $E_{K_{i^*}}$ are independent ideal permutations, and $\mathcal{A}$ is nonadaptive.

Case 3: $X \in \{E_{K_i}(K_j), E_{K_i}(R_j)\}$ and $P \in \{K_{j^*}, R_{j^*}\}$. Then $\Pr[X = P] = 2^{-k}$, as E_{K_i} is an ideal permutation, and $\mathcal{A}$ is nonadaptive.

Case 4: $X \in \{K_j, R_j\}$ and $P \in \{E_{K_{i^*}}(K_{j^*}), E_{K_{i^*}}(R_{j^*})\}$. As in Case 3, the chance that $X = P$ is 2^{-k}.

Case 5: $X = K_j$ and $P = K_{j^*}$. As mentioned above, $j \neq j^*$ so that we can reach line 96 to set *bad*, and thus $K_j \neq K_{j^*}$.

Case 6: $X = R_j$ and $P = R_{j^*}$. As mentioned above, $j \neq j^*$ so that we can reach line 96 to set *bad*. But $\Pr[R_j = R_{j^*}] = 2^{-k}$, because we sample R_j and R_{j^*} independently, and $\mathcal{A}$ is nonadaptive.

Back to our games, note that the outputs of game G_5 are independent of the challenge bit b. (If $\mathcal{A}$ asks a redundant query then we give an answer consistent to the prior ones. Otherwise, we give a random answer.) Hence $\Pr[G_5^{\mathcal{A}}(k)] = 1/2$, and consequently, $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}[E]}(\mathcal{A}, k) = 2\Pr[G_0^{\mathcal{A}}(k)] - 1 = 2(\Pr[G_0^{\mathcal{A}}(k)] - \Pr[G_5^{\mathcal{A}}(k)]) \leq (10Q^2 + 2Q + 8q_E)/2^k$. $\square$

UNWINDING THE RESULTS. One needs to be careful in combining Theorems 3.5.1 and 3.5.3 to obtain a good bound on the security of Garble1 when instantiated with a DKC made by double encryption. Let adversary $\mathcal{A}$ attack $\mathsf{Gb1}[\mathbb{E}[E]]$ and assume $\mathcal{A}(1^k)$ outputs circuits of at most $r \leq 2^{\tau(k)-2}$ wires and fan-out at most ν. Assume further that it makes at most Q_E queries to E and E^{-1}. The corresponding DKC adversary $\mathcal{D}$ needs to use at most $4r$ calls to $\mathbb{E}$ for garbling, and thus makes at most $q_E = 8r + Q_E$ queries to E and E^{-1}. Then, from Theorems 3.5.1 and 3.5.3, there is a random variable $0 < Q \leq 2\nu$ such that $\mathbf{E}[Q] < 4$ and

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathrm{Garble1}[\mathbb{E}[E]]}(\mathcal{A}, k) &\leq \frac{r}{2^k} \cdot (20\mathbf{E}[Q^2] + 4\mathbf{E}[Q] + 16q_E) \\
&\leq \frac{r}{2^k} \cdot (20\mathbf{E}[2\nu Q] + 4\mathbf{E}[Q] + 16Q_E + 128r) \\
&< 160r\nu/2^k + 16r/2^k + 16rQ_E/2^k + 128r^2/2^k \ .
\end{aligned}$$

The bound is quite satisfactory. Above, the expectation $\mathbf{E}[Q]$ appears in the first inequality as our advantage notion satisfies the following linearity condition: if an adversary $\mathcal{A}$ behaves as adversary $\mathcal{A}_1$ with probability p, and behaves like $\mathcal{A}_2$ otherwise, then $\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}, k) = p\,\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}_1, k) + (1-p)\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}_2, k)$.

3.5.7 AES-based instantiations

We now consider concrete instantiations. This means we fix a value k of the security parameter and suggest ways to realize $\mathbb{E}$ on k-bit keys based on blockciphers, specifically AES. Security for these instantiations can be derived via the concrete security bounds that we stated above following Theorem 3.5.1. Different choices of instantiation lead to different tradeoffs

between assumptions and efficiency. We begin with ways to instantiate F on $(k-1)$-bit keys:

- Let $\mathsf{F}_K(T)$ be the first k bits of $E_K(T \,\|\, 0) \,\|\, E_K(T \,\|\, 1)$ for a blockcipher E having block length and key length of $(k-1)$; to be concrete, $E = \text{AES128}$, $k = 129$, $|K| = 128$, and $\tau = |T| = 127$. This construction is a good PRF under the standard assumption that E is a good PRP. With this instantiation, evaluating a garbled gate costs four AES operations.
- Let $\mathsf{F}_K(T)$ be $E_{K\|0}(T)$ for a blockcipher having a k-bit key and block size, say $E = \text{AES128}$ and $k = \tau = |T| = 128$ and $|K| = 127$. Assuming that E is a good PRP is not enough to prove that F is a good PRF, as zeroing out a bit of the key does not, in general, preserve PRF security [82]. Still, it seems reasonable to directly assume this F is a good PRF. Costs are halved compared to the above; now, evaluating a garbled gate requires two AES operations.

Next we suggest some further ways to make the dual-key cipher $\mathbb{E}$ directly, meaning not via a PRF. The first follows the double-encryption realization of garbled gates attributed to Yao by Goldreich [38] (which would have been understood that primitive to be probabilistic, not a blockcipher). The second method is extremely efficient—the most efficient approach now known.

- Let $\mathbb{E}^T_{A,B}(X) = E_A(E_B(X))$ (the tweak is ignored), where $E\colon \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ is a blockcipher, say AES128. For a proof we would model E as an ideal cipher. Composition of encryption schemes is understood by many researchers to be Yao's original approach, although the earliest expositions make this seem doubtful.
- Let $\mathbb{E}^T_{A,B}(X) = E_{\mathsf{const}}(K) \oplus K \oplus X$ where $K = A \oplus B \oplus T$ and $E = \text{AES128}$, say, and const is a fixed 128-bit string. Here $k = \tau = 128$. With this instantiation evaluating a gate costs only 1 AES operation. Even more important, all AES operations employ a single, fixed key. This allows one to take full advantage of AES-NI hardware support to get extremely high speeds. See Section 3.5.8 for a proof, in which we model $E_{\mathsf{const}}(\cdot)$ as a random permutation π, giving the adversary access to oracles for π and its inverse.

Other one-call, fixed-key schemes are possible, for obliviousness, authenticity, and adjustments to allow the free-xor and row-reduction optimizations [64, 84].

Basing garbled-circuit evaluation on AES and employing AES-NI in an implementation was also suggested by Kreuter, Shelat, and Shen [66]. They use AES-256, rekeying with gate evaluation.

3.5.8 Dual-key ciphers from an ideal permutation

We prove the dkc-security of the instantiation $\mathbb{E}[\pi]$ in which $\mathbb{E}^T_{A,B}(X) = \pi(K) \oplus K \oplus X$, with $K = A \oplus B \oplus T$ and π being an ideal permutation. In the following theorem, we will show that if an adversary $\mathcal{A}$ makes Q queries to the ENCRYPT oracle, and q_π queries to π and π^{-1} then $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}[\pi]}(\mathcal{A}, k) \leq 12Qq_\pi/2^k + 3Q^2/2^k$.

Theorem 3.5.4. Let π be an ideal permutation. Then $\mathbb{E}[\pi]$ is a secure dual-key cipher.

Proof. Consider games $G_0 - G_3$ in Fig. 3.5.11. In each game, the adversary $\mathcal{A}$ has indirect access to π and π^{-1} by calling procedures Π and Π^{-1}. Game G_0 corresponds to game DKC. Suppose that $\mathcal{A}(1^k)$ makes $q_\pi(k)$ queries to Π and Π^{-1}, and $Q(k)$ queries to ENCRYPT, with $q_\pi \leq 2^{k-2}$.

We explain the game chain up until the terminal game. $\triangleright G_0 \to G_1$: in these games, we sample a one-time pad S and set $\pi(P)$ to $S \oplus P$. This may cause inconsistency if $\pi(P)$ or $\pi^{-1}(S \oplus P)$ is defined before. In that case, game G_0 resets S to the value consistent with π, but game G_1 does nothing. The two games are identical until game G_1 sets *bad*. We claim that for the ℓth query to ENCRYPT, the chance that G_1 sets *bad* in this query is at most $3q_\pi/2^k + (3\ell - 3)/2^k$. Then, by union bound,

$$\Pr[\mathsf{BAD}(G_1^{\mathcal{A}}(k))] \leq \sum_{\ell=1}^{Q} 3q_\pi/2^k + (3\ell - 3)/2^k \leq 3Qq_\pi/2^k + 3Q^2/2^{k+1} \ .$$

To justify our claim, since $\mathcal{A}$ is nonadaptive, without loss of generality, suppose that during the period from the first ENCRYPT query to the last ENCRYPT query, $\mathcal{A}$ makes no query to

proc INITIALIZE()
$b \twoheadleftarrow \{0,1\}, \quad K \twoheadleftarrow \{0,1\}^k$
return $\mathrm{lsb}(K)$

proc $\Pi(X)$
if $\pi[X] = \perp$ **then**
  $\pi[X] \twoheadleftarrow \{0,1\}^k \backslash \mathrm{Ran}(\pi)$
return $\pi[X]$

proc $\Pi^{-1}(Y)$
if $\pi^{-1}[Y] = \perp$ **then**
  $\pi^{-1}[Y] \twoheadleftarrow \{0,1\}^k \backslash \mathrm{Dom}(\pi)$
return $\pi^{-1}[Y]$

proc ENCRYPT(i, j, pos, T) Game G_0 / Game G_1
if *used*$[T]$ **or** $i \geq j$ **then return** $\perp$
used$[T] \leftarrow$ true
if $K_i = \perp$ **then** $A \twoheadleftarrow \{0,1\}^{k-1}, \quad K_i \leftarrow A \,\|\, (i \bmod 2)$
if $K_j = \perp$ **then** $B \twoheadleftarrow \{0,1\}^{k-1}, \quad K_j \leftarrow B \,\|\, (j \bmod 2)$
if $R_j = \perp$ **then** $R_j \twoheadleftarrow \{0,1\}^k$
$X_1 \leftarrow K_j, \;\; X_0 \leftarrow R_j, \;\; P \leftarrow K_i \oplus K \oplus T, \;\; S \twoheadleftarrow \{0,1\}^k$
if $P \in \mathrm{Dom}(\pi)$ **or** $S \oplus P \in \mathrm{Ran}(\pi)$ **then**
  bad $\leftarrow$ true
  $S \leftarrow \Pi(P) \oplus P$ $\longleftarrow$ Use in game G_0
$\pi(P) \leftarrow S \oplus P$
return $(K_i, K_j, S \oplus X_b)$

proc INITIALIZE()
$b \twoheadleftarrow \{0,1\}, \quad K \twoheadleftarrow \{0,1\}^k$
$\mathrm{BadDom} \leftarrow \emptyset, \quad \mathrm{BadRan} \leftarrow \emptyset$
return $\mathrm{lsb}(K)$

proc $\Pi(X)$
if $X \in \mathrm{BadDom}$ **then** *bad* $\leftarrow$ true
if $\pi[X] = \perp$ **then**
  $\pi[X] \twoheadleftarrow \{0,1\}^k \backslash \mathrm{Ran}(\pi)$
return $\pi[X]$

proc $\Pi^{-1}(Y)$
if $Y \in \mathrm{BadRan}$ **then** *bad* $\leftarrow$ true
if $\pi^{-1}[Y] = \perp$ **then**
  $\pi^{-1}[Y] \twoheadleftarrow \{0,1\}^k \backslash \mathrm{Dom}(\pi)$
return $\pi^{-1}[Y]$

proc ENCRYPT(i, j, pos, T) Game G_2 / Game G_3
if *used*$[T]$ **or** $i \geq j$ **then return** $\perp$
used$[T] \leftarrow$ true
if $K_i = \perp$ **then** $A \twoheadleftarrow \{0,1\}^{k-1}, \quad K_i \leftarrow A \,\|\, (i \bmod 2)$
if $K_j = \perp$ **then** $B \twoheadleftarrow \{0,1\}^{k-1}, \quad K_j \leftarrow B \,\|\, (j \bmod 2)$
if $R_j = \perp$ **then** $R_j \twoheadleftarrow \{0,1\}^k$
$X_1 \leftarrow K_j, \quad X_0 \leftarrow R_j, \quad P \leftarrow K_i \oplus K \oplus T$
$S \twoheadleftarrow \{0,1\}^k, \quad U \leftarrow S \oplus X_b$ $\longleftarrow$ Use in game G_2
$U \twoheadleftarrow \{0,1\}^k, \quad S \leftarrow U \oplus X_b$ $\longleftarrow$ Use in game G_3
$\mathrm{BadDom} \leftarrow \mathrm{BadDom} \cup \{P\}$
$\mathrm{BadRan} \leftarrow \mathrm{BadRan} \cup \{S \oplus P\}$
return (K_i, K_j, U)

Figure 3.5.11: **Games for the proof of Theorem 3.5.4.** Procedure FINALIZE(b') returns $(b = b')$.

Π and Π^{-1}. Consider the ℓth ENCRYPT query. Since $S \twoheadleftarrow \{0,1\}^k$ and $|\mathrm{Ran}(\pi)| \leq q_\pi + \ell - 1$, the chance that $S \oplus P \in \mathrm{Ran}(\pi)$ is at most $(q_\pi + \ell - 1)/2^k$. What remains is to show that

$$\Pr[P \in \mathrm{Dom}(\pi)] \leq (q_\pi + \ell - 1)/2^{k-1} \;\; . \tag{3.5.1}$$

Recall that $P = K \oplus K_i \oplus T$. Consider $X \in \mathrm{Dom}(\pi)$. If X is added to the domain of π via a prior ENCRYPT query, then either $X = K \oplus K_i \oplus T^*$ or $X = K \oplus K_{i^*} \oplus T^*$. In the former case, since the tweaks must be unique, $T \neq T^*$, and thus $X \neq P$. In the latter case, since the adversary is nonadaptive, and the first $k-1$ bits of K_i and K_{i^*} are chosen uniformly and independently, the chance that $X = P$ is at most 2^{1-k}. On the other hand, if X is

added to the domain of π via a query to Π or Π^{-1} then this query is made before the first ENCRYPT query. As the first $k-1$ bits of P are uniformly chosen, the chance that $X = P$ is at most 2^{1-k}. Hence claim (3.5.1) follows from the union bound, as $|\mathrm{Dom}(\pi)| \le q_\pi + \ell - 1$.

$\triangleright G_1 \to G_2$: We do *not* set $\pi(P)$ to be $S \oplus P$. In addition, we maintain two sets BadDom and BadRan that are initialized to the empty sets. For each query ENCRYPT, the string P is added to BadDom and $S \oplus P$ is added to BadRan. If the adversary queries Π with a message $X \in$ BadDom or Π^{-1} with a message $Y \in$ BadRan then *bad* is set. The two games are identical until G_2 sets *bad*. Moreover, as the first $k-1$ bits of each $P \in$ BadDom are uniformly random,

$$\Pr[\mathsf{BAD}(G_2^{\mathcal{A}}(k))] \le Qq_\pi/2^{k-1} \ .$$

$\triangleright G_2 \to G_3$: we use the technique of "swapping dependent and independent variables". Namely, instead of sampling S and then computing $U \leftarrow S \oplus X_b$, we sample U and compute $S \leftarrow U \oplus X_b$. The change is conservative. Then $\Pr[G_3^{\mathcal{A}}(k)] = 1/2$, as the outputs of game G_3 are independent of b. Hence, $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}[\pi]}(\mathcal{A}, k) = 2\Pr[G_0^{\mathcal{A}}(k)] - 1 \le 10Qq_\pi/2^k + 3Q^2/2^k$. □

UNWINDING THE RESULT. Again, one needs to be careful in combining Theorems 3.5.1 and 3.5.4 to obtain a good bound on the security of Garble1 when instantiated with $\mathbb{E}[\pi]$. Consider an adversary $\mathcal{A}$ attacking $\mathsf{Gb1}[\mathbb{E}[\pi]]$ and assume that $\mathcal{A}(1^k)$ outputs circuits of at most $r \le 2^{\tau(k)-2}$ wires and fan-out at most ν. Suppose it makes at most $Q_\pi(k)$ queries to π and π^{-1}. The corresponding DKC adversary $\mathcal{D}$ needs to use at most $4r$ calls to $\mathbb{E}$ for garbling, and thus makes at most $q_\pi = 4r + Q_\pi$ queries to π and π^{-1}. Then, from Theorem 3.5.1 and Theorem 3.5.4, there is a random variable $0 < Q < 2\nu$ such that $\mathbf{E}[Q] < 4$ and

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathrm{Garble1}[\mathbb{E}[\pi]]}(\mathcal{A}, k) &\le \frac{r}{2^k} \cdot (20q_\pi \mathbf{E}[Q] + 6\mathbf{E}[Q^2]) \\
&\le \frac{r}{2^k} \cdot (20Q_\pi \mathbf{E}[Q] + 80r\mathbf{E}[Q] + 6\mathbf{E}[2\nu Q]) \\
&< 80rQ_\pi/2^k + 320r^2/2^k + 48r\nu/2^k \ .
\end{aligned}$$

The bound is satisfactory.

200 **proc** $\mathsf{Gb}(1^k, f)$
201 $(n, m, q, A', B', G) \leftarrow f$
202 **for** $i \in \{1, \ldots, n+q\}$ **do** $t \twoheadleftarrow \{0,1\}$, $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}t$, $X_i^1 \twoheadleftarrow \{0,1\}^{k-1}\bar{t}$
203 **for** $(g, i, j) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ **do**
204 $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
205 $A \leftarrow X_a^i$, $\mathtt{a} \leftarrow \mathrm{lsb}(A)$, $B \leftarrow X_b^j$, $\mathtt{b} \leftarrow \mathrm{lsb}(B)$, $T \leftarrow g \,\|\, \mathtt{a} \,\|\, \mathtt{b}$, $P[g, \mathtt{a}, \mathtt{b}] \leftarrow \mathbb{E}_{A,B}^T\big(X_g^{G_g(i,j)}\big)$
206 $F \leftarrow (n, m, q, A', B', P)$
207 $e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
208 $d \leftarrow (X_{n+q-m+1}^0, X_{n+q-m+1}^1, \ldots, X_{n+q}^0, X_{n+q}^1)$
209 **return** (F, e, d)

220 **proc** $\mathsf{En}(e, x)$
221 $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
222 $x_1 \cdots x_n \leftarrow x$, $X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$
223 **return** X

230 **proc** $\mathsf{De}(d, Y)$
231 $(Y_1, \ldots, Y_m) \leftarrow Y$, $(Y_1^0, Y_1^1, \ldots, Y_m^0, Y_m^1) \leftarrow d$
232 **for** $i \in \{1, \ldots, m\}$ **do**
233 **if** $Y_i = Y_i^0$ **then** $y_i \leftarrow 0$
234 **else if** $Y_i = Y_i^1$ **then** $y_i \leftarrow 1$ **else return** $\perp$
235 **return** $y \leftarrow y_1 \cdots y_m$

240 **proc** $\mathsf{ev}(f, x)$
241 $(n, m, q, A, B, G) \leftarrow f$, $x_1 \cdots x_n \leftarrow x$
242 **for** $g \leftarrow n+1$ **to** $n+q$ **do**
243 $a \leftarrow A(g)$, $b \leftarrow B(g)$
244 $x_g \leftarrow G_g(x_a, x_b)$
245 **return** $x_{n+q-m+1} \cdots x_{n+q}$

250 **proc** $\mathsf{Ev}(F, X)$
251 $(n, m, q, A', B', P) \leftarrow F$, $(X_1, \ldots, X_n) \leftarrow X$
252 **for** $g \leftarrow n+1$ **to** $n+q$ **do**
253 $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
254 $A \leftarrow X_a$, $\mathtt{a} \leftarrow \mathrm{lsb}(A)$, $B \leftarrow X_b$, $\mathtt{b} \leftarrow \mathrm{lsb}(B)$
255 $T \leftarrow g \,\|\, \mathtt{a} \,\|\, \mathtt{b}$, $X_g \leftarrow \mathbb{D}_{A,B}^T\big(P[g, \mathtt{a}, \mathtt{b}]\big)$
256 **return** $(X_{n+q-m+1}, \ldots, X_{n+q})$

Figure 3.6.1: **Garbling scheme Garble2.** Its components are $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ where ev, shown for completeness, is canonical circuit evaluation. We assume a dual-key cipher $\mathbb{E}$ with tweak length τ and let $\mathbb{D}$ denote its inverse.

3.6 Achieving privacy, authenticity and obliviousness: Garble2

We now describe a scheme Garble2 that satisfies not only privacy but also obliviousness and authenticity. The scheme is like Garble1 except, first, the last bit of a token is always uniform, even for output wires. This will give obliviousness. Next, the string encoding the decoding function is made to list all the tokens for all the output wires, ordered to make clear which tokens have what semantics. This engenders authenticity. See Fig. 3.6.1.

Talking through some of the pseudocode, line 202 now assigns a token with random semantics to each and every wire. Lines 203–207 compute the garbled function F and encoding function e exactly as with Garble1. Line 208 now records the vector of tokens for

each of the m output wires. (Recall that, under our conventions, the last m of the r total wires are the output wires, these providing the m output bits, in order.) At lines 230–235 decoding procedure De, when presented a $2m$-vector d and an m-vector Y, verifies that each component of the latter is in the corresponding set of two allowed values. If so, we determine the correct semantics for this output bit using our convention that Y_i^b has semantics b.

Garble2 simultaneously achieves privacy, obliviousness, and authenticity if instantiated in the same manner as we instantiated Garble1. This is captured by the following result. Again, as per Corollary 3.4.5 it does not matter whether we consider ind or sim, and for simplicity we pick the former.

Theorem 3.6.1. If $\mathbb{E}$ is a secure dual-key cipher then $\mathcal{G} = \text{Garble2}[\mathbb{E}] \in \mathsf{GS}(\text{prv.ind}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\text{obv.ind}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\text{aut})$.

As usual this asymptotic claim is underlain by concrete blackbox reductions and concrete bounds as follows. There are blackbox reductions U_{xxx} for $\text{xxx} \in \{\text{prv.ind}, \text{obv.ind}, \text{aut}\}$ s.t. if $\mathcal{A}(1^k)$ outputs circuits of at most r wires and fan-out at most ν, and then $\mathcal{D} = U^{\mathcal{A}}$ achieves xxx-advantage of at least ε, then $\mathcal{D} = U_{\text{xxx}}^{\mathcal{A}}$ achieves dkc-advantage at least $\varepsilon/2r - 2^{1-k}$, makes $Q \leq 2\nu$ oracle queries, with $\mathbf{E}[Q] < 4$. It runs in time about that of $\mathcal{A}$ plus the time for $4r$ computations of $\mathbb{E}$ on k-bit keys.

Proof. The privacy security can be proved by adapting the proof of Theorem 3.5.1 as follows. First, for each output wire i, the type t_i is chosen uniformly. Next, the games return $(F, (X_1^{x_1}, \ldots, X_n^{x_n}), d)$ instead of $(F, (X_1^{x_1}, \ldots, X_n^{x_n}), \varepsilon)$, where d is defined as in line 208 of Fig. 3.6.1. In other words, for every output wire i, in addition to the visible token of wire i, the games also return the invisible tokens of wire i, which are independent of the challenge bit c . Moreover, since no incoming wire of some gate can be an output wire, these invisible tokens won't be used as keys for the dual-key cipher $\mathbb{E}$. Hence the argument in the proof of Theorem 3.5.1 still applies here.

For obliviousness security, we again adapt the proof of Theorem 3.5.1, with the following

differences. First, the games return $(F, (X_1^{x_1}, \ldots, X_n^{x_n}))$ instead of $(F, (X_1^{x_1}, \ldots, X_n^{x_n}), \varepsilon)$. Next, for every output wire i, the type t_i is uniformly random, and thus independent of the challenge bit c of each game, although we may have $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$.

For authenticity security, we construct an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) \;\leq\; \mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{B}, k) + 2^{1-k}$$

where $\mathcal{B}$'s running time is at most that of $\mathcal{A}$ plus an overhead linear to the size of $\mathcal{A}$'s query. Moreover, $\mathcal{B}$ let $\mathcal{A}$ do all the queries to the oracle ENCRYPT; so it has as many ENCRYPT queries as $\mathcal{A}$. We then apply the privacy proof to $\mathcal{B}$. The adversary $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. Suppose that $\mathcal{A}$ queries (f, x), with $f = (n, m, q, A, B, G)$. Without loss of generality, suppose that $x \in \{0,1\}^n$, otherwise $\mathcal{A}$ will have advantage 0, and it's trivial to construct $\mathcal{B}$ of advantage 0. Let $y = y_1 \cdots y_m = f(x)$. Adversary $\mathcal{B}$ then constructs a circuit $f' = (n, m, q, A, B, G')$ as follows. For every gate g, if its outgoing wire j is an output wire then G'_g is a constant function that always outputs $y_{j-(n+q-m)}$. Otherwise, $G'_g = G_g$. Adversary $\mathcal{B}$ queries its oracle GARBLE with (f', f, x, x). Since $f'(x) = y$ and $\Phi_{\mathrm{topo}}(f') = \Phi_{\mathrm{topo}}(f)$ and the side-information function is Φ_{topo}, the query (f', f, x, x) in game $\mathrm{PrvInd}_{\mathcal{G},\Phi_{\mathrm{topo}}}$ will not result in answer $\perp$. Let (F, X, d) denote the answer. Adversary $\mathcal{B}$ gives (F, X) to $\mathcal{A}$ as response to its query (f, x). It will output answer 1 if and only if the answer Y of $\mathcal{A}$ satisfies $\mathsf{De}(d, Y) \neq \perp$ and $Y \neq F(X)$. Then

$$\Pr\left[\,\mathrm{PrvInd}^{\mathcal{B}}_{\mathcal{G},\Phi_{\mathrm{topo}}}(k) \mid b = 1\,\right] = \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k), \tag{3.6.1}$$

where b is the challenge bit of game $\mathrm{PrvInd}_{\mathcal{G},\Phi_{\mathrm{topo}}}$. We now show that

$$\Pr\left[\,\neg\mathrm{PrvInd}^{\mathcal{B}}_{\mathcal{G},\Phi_{\mathrm{topo}}}(k) \mid b = 0\,\right] \;\leq\; 2^{1-k}\;. \tag{3.6.2}$$

Subtracting Eq. (3.6.1) and Eq. (3.6.2) will give the bound claimed in the theorem. Suppose that given (F, X), adversary $\mathcal{A}$ outputs $Y = (Y_1, \ldots, Y_m)$. Let $d = (Y_1^0, Y_1^1, \ldots, Y_m^0, Y_m^1)$ and let i be the smallest integer such that $Y_i \neq Y_i^{y_i}$. This integer i is well-defined if $Y \neq F(X) = (Y_1^{y_1}, \ldots, Y_m^{y_m})$. Consequently, if $\mathsf{De}(d, Y) \neq \perp$ and $Y \neq F(X)$ then Y_i must be $Y_i^{\overline{y_i}}$. This uses the fact that $Y_i^{\overline{y_i}} \neq Y_i^{y_i}$, which is true because the construction makes their type-bits

unequal. Thus we have

$$\Pr\left[\,\neg\mathrm{PrvInd}^{\mathcal{B}}_{\mathcal{G},\Phi_{\mathrm{topo}}}(k) \mid b=0\,\right] \;\le\; \Pr\left[\,Y_i = Y_i^{\overline{y_i}} \mid Y \neq F(X)\,\right] \quad . \tag{3.6.3}$$

Let $A\|\mathtt{a} \leftarrow Y_i^{\overline{y_i}}$, and let $j = i+n+q-m$. Since the output bit at wire j is always y_i, during the garbling process of f', in line 205 of Fig. 3.6.1, we never encrypt token $Y_i^{\overline{y_i}}$. Moreover, the token $Y_i^{\overline{y_i}}$ is not used as a key of $\mathbb{E}$. The string A is therefore independent of (F, X), and thus the right-hand side of Eq. (3.6.3) is at most 2^{1-k}. □

3.7 Applications

We believe that most applications that employ garbled circuits can be recast to use an arbitrary garbling scheme possessing one or more of the security properties we've defined. While a complete reworking of all existing garbled-circuit-using applications is beyond the scope of this thesis, we sketch two examples. First we consider the classical use of garbled circuits for two-party SFE (Secure Function Evaluation) and PFE (Private Function Evaluation). Then we consider their more recent use for securely encrypting key-dependent messages.

3.7.1 Two-party SFE and PFE

The classic methods for SFE and PFE combine the garbled-circuit technique with oblivious transfer (OT). The construction and proof are monolithic and complex, incorporating proofs of the garbled circuit technique. Here we aim to show how use of our formalization can simplify this process to produce proofs that are modular and thus simpler and more rigorous. We build SFE and PFE protocols by a modular combination of an arbitrary garbling scheme and an arbitrary OT protocol, reducing the security of the constructed protocol to the security of its two constituents. Besides simplicity we gain in flexibility of instantiation, for we can plug in any garbling scheme meeting our definitions and immediately get new SFE or PFE protocols that inherit the efficiency of the garbling scheme.

Classically, in SFE, a function f is public and the interaction results in party 1 learning

$f(x_1 \| x_2)$ (but no more) while party 2 learns nothing, where x_i is the private input of party $i \in \{1, 2\}$. In PFE, party 1 has a string x, party 2 has a function f and the outcome of the protocol is that party 1 learns $f(x)$ (but no more) while party 2 learns nothing. However, through the use of universal circuits, the two versions of the problem are equivalent. Thus, we will treat only one. We pick PFE because it is more directly obtained via garbling schemes.

It is part of our thesis that this type of program can and should be carried out rigorously and fully and that our formalization of garbling schemes enables one to do this. To this end we provide self-contained definitions of security for PFE (OT as a special case). These definitions are not the only possible ones, nor necessarily the strongest, but we need to pin something down to provide a full treatment. The setting here is that of honest but curious adversaries.

TWO-PARTY PROTOCOLS. We view a two-party protocol as specified by a pair $\Pi = (\Pi_1, \Pi_2)$ of PT algorithms. Party $i \in \{1, 2\}$ will run Π_1 on its current state and the incoming message from the other party to produce an outgoing message, a local output, and a decision to halt or continue. The initial state of party i consists of the unary encoding 1^k of the security parameter $k \in \mathbb{N}$ and the (private) input I_i of this party, and the interaction continues until both parties halt. We will not further formalize this process since the details are not important to what we do. What is important is that we are able to define the PT algorithm View^i_Π that on input $(1^k, I_1, I_2)$ returns the view of party i in an execution of Π with security parameter k and inputs I_1, I_2 for the two parties, respectively. Specifically, the algorithm picks at random coins ω_1, ω_2, executes the interaction between the parties as determined by Π with the initial state and coins of party $j \in \{1, 2\}$ being $(1^k, I_j)$ and ω_j respectively, and returns $(\mathit{conv}, \omega_i)$ where the conversation conv is the sequence of messages exchanged. We let $\mathsf{Out}^i_\Pi(1^k, I_1, I_2)$ return the local output of party i at the end of the protocol. This is a deterministic function of $\mathsf{View}^i_\Pi(1^k, I_1, I_2)$.

PFE. Party 1 has a string x and party 2 has a function f. The outcome of the protocol should be that party 1 learns $f(x)$. Security requires that party 2 learns nothing about x

proc GETVIEW(x, f)	Game $\text{PfeSim}_{\mathcal{F},i,\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$	
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$	
if $b = 1$ **then return** $view \leftarrow \mathsf{View}^i_\Pi(1^k, x, f)$	
if $i = 1$ **then return** $view \leftarrow \mathcal{S}(1^k, x, \mathsf{ev}(f,x), \Phi(f))$	
if $i = 2$ **then return** $view \leftarrow \mathcal{S}(1^k, f, \lvert x\rvert)$	

Figure 3.7.1: **Game for defining the pfe.sim security of a PFE scheme $\mathcal{F} = (\Pi, \mathsf{ev})$.** Procedure FINALIZE$(b')$ returns $(b = b')$. The game depends on a security parameter $k \in \mathbb{N}$.

(beyond its length) and party 1 learns nothing about f (beyond side information we are willing to leak, such as the number of gates in the circuit f).

Formally a private function evaluation (PFE) protocol is a tuple $\mathcal{F} = (\Pi, \mathsf{ev})$ where Π is a 2-party protocol as above and ev is just like in a garbling scheme, meaning a PT deterministic map that associates to any string f a function $\mathsf{ev}(f,\cdot)\colon \{0,1\}^{f.n} \to \{0,1\}^{f.m}$. The correctness requirement is that for all f and all $x \in \{0,1\}^{f.n}$ we have

$$\Pr[\mathsf{Out}^1_\Pi(1^k, x, f) = \mathsf{ev}(f,x)] \;=\; 1 \;.$$

The security notion we consider is privacy in the honest-but-curious setting, meaning the parties follow the protocol and the intent is that their views do not allow the computation of any undesired information. An adversary $\mathcal{B}$ is allowed a single GETVIEW query in game $\text{PfeSim}_{\mathcal{F},i,\Phi,\mathcal{S}}$ of Fig. 3.7.1, and its advantage is

$$\mathbf{Adv}^{\text{pfe.sim},\Phi,\mathcal{S}}_{\mathcal{F},i}(\mathcal{B}, k) \;=\; 2\Pr[\text{PfeSim}^{\mathcal{B}}_{\mathcal{F},i,\Phi,\mathcal{S}}(k)] - 1 \;.$$

We say that $\mathcal{F}$ is pfe.sim relative to Φ if for each $i \in \{0,1\}$ and each PT adversary $\mathcal{B}$ there is a PT simulator $\mathcal{S}$ such that the function $\mathbf{Adv}^{\text{pfe.sim},\Phi,\mathcal{S}}_{\mathcal{F},i}(\mathcal{B},\cdot)$ is negligible.

OBLIVIOUS TRANSFER. The construction will utilize a protocol for 1-out-of-2 oblivious transfer where party 1 has a selection bit s, party 2 has inputs X^0, X^1, and the result is that party 1 gets X^s while party 2 gets nothing. It is convenient to assume an extension where party 1 has bits $x_1, \ldots, x_n$, party 2 has inputs $X^0_1, X^1_1, \ldots, X^0_n, X^1_n$, and the result is that party 1 gets $X^{x_1}_1, \ldots, X^{x_n}_n$ while party 2 gets nothing. Such an extended protocol may be produced by sequential repetition of the basic protocol. Formally an OT protocol is a PFE scheme $\mathcal{OT} = (\Pi^{\text{ot}}, \mathsf{ev}^{\text{ot}})$ where Π^{ot} is a 2-party protocol and

$\mathsf{ev}^{\mathrm{ot}}((X_1^0, X_1^1, \ldots, X_n^0, X_n^1), x) = (X_1^{x_1}, \ldots, X_n^{x_n})$. Here, a function is described by a vector $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$, and its evaluation on an n-bit input x is $(X_1^{x_1}, \ldots, X_n^{x_n})$. We assume a pfe.sim-secure scheme $\mathcal{OT} = (\Pi^{\mathrm{ot}}, \mathsf{ev}^{\mathrm{ot}})$ relative to the side information function $\Phi_{\mathrm{ot}}((X_1^0, X_1^1, \ldots, X_n^0, X_n^1)) = (|X_1^0|, |X_1^1|, \ldots, |X_n^0|, |X_n^1|)$.

THE PROTOCOL. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective garbling scheme that is prv.sim-secure over Φ. We define a PFE scheme $\mathcal{F} = (\Pi, \mathsf{ev})$ which allows the secure computation of exactly the class of functions $\{\mathsf{ev}(f, \cdot) : f \in \{0,1\}^*\}$ that $\mathcal{G}$ can garble. Party 2, on inputs $1^k, f$, begins by letting $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and parsing e as $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$. It sends F, d to party 1. Now the parties execute the OT protocol with party 1 having selection string x and party 2 having inputs $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$. As a result, party 1 obtains $X = (X_1^{x_1}, \ldots, X_n^{x_n})$. It now outputs $y \leftarrow \mathsf{De}(d, \mathsf{Ev}(F, X))$ and halts.

Theorem 3.7.1. Assume $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is a projective garbling scheme that is prv.sim-secure over Φ. Assume $\mathcal{OT} = (\Pi^{\mathrm{ot}}, \mathsf{ev}^{\mathrm{ot}})$ is a OT protocol that is pfe.sim-secure relative to Φ_{ot}. Let $\mathcal{F} = (\Pi, \mathsf{ev})$ be the pfe scheme constructed above. Then $\mathcal{F}$ is pfe.sim-secure relative to Φ.

Proof. Let $i \in \{1, 2\}$ and let $\mathcal{B}$ be a PT adversary attacking $\mathcal{F}$. We build a PT adversary $\mathcal{B}_{\mathcal{G}}$ attacking $\mathcal{G}$ and a PT adversary $\mathcal{B}_{\mathcal{OT}}$ attacking $\mathcal{OT}$. By assumption, these have simulators, respectively $\mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{OT}}$. We then use these simulators to build a simulator $\mathcal{S}$ for $\mathcal{B}$ such that for every $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{pfe.sim},\Phi,\mathcal{S}}_{\mathcal{F},i}(\mathcal{B}, k) \leq \mathbf{Adv}^{\mathrm{prv.sim},\,\Phi,\mathcal{S}_{\mathcal{G}}}_{\mathcal{G}}(\mathcal{B}_{\mathcal{G}}, k) + \mathbf{Adv}^{\mathrm{pfe.sim},\Phi_{\mathrm{ot}},\mathcal{S}_{\mathcal{OT}}}_{\mathcal{OT},i}(\mathcal{B}_{\mathcal{OT}}, k)\ .$$

This yields the desired conclusion. We now proceed to the constructions and analyses. We consider separately the cases $i = 1$ and $i = 2$, beginning with the former.

Adversary $\mathcal{B}_{\mathcal{G}}(1^k)$ runs $\mathcal{B}(1^k)$ to get its GETVIEW query x, f. It will compute and return a reply *view* to this query as follows. Adversary $\mathcal{B}_{\mathcal{G}}$ queries its GARBLE oracle with f, x to get back $(F, X_1, \ldots, X_n), d)$. It records (F, d) as the first message in *conv*. (This message is

from party 2 to party 1.) Now, for $i = 1, \ldots, n$ it lets $X_i^{x_i} \leftarrow X_i$ and $X_i^{1-x_i} \twoheadleftarrow \{0,1\}^{|X_i|}$. It then lets $view^{\mathrm{ot}} \leftarrow \mathsf{View}^1_{\Pi^{\mathrm{ot}}}(1^k, x, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1))$. It obtains this by direct execution of 2-party protocol Π^{ot} on inputs $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ for party 2 and x for party 1. Parsing $view^{\mathrm{ot}}$ as $(conv^{\mathrm{ot}}, \omega_1^{\mathrm{ot}})$, it appends $conv^{\mathrm{ot}}$ to $conv$ and then returns $view = (conv, \omega_1^{\mathrm{ot}})$ as the answer to $\mathcal{B}$'s query. Adversary $\mathcal{B}$ now outputs a bit b', and $\mathcal{B}$ adopts this as its own output as well.

Adversary $\mathcal{B}_{\mathcal{OT}}(1^k)$ runs $\mathcal{B}(1^k)$ to get its GetView query x, f. It will compute and return a reply $view$ to this query as follows. Adversary $\mathcal{B}_{\mathcal{OT}}$ lets $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and parses $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$. It records (F, d) as the first message in $conv$. It makes query $view^{\mathrm{ot}} \leftarrow \textsc{GetView}(x, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1))$. Parsing $view^{\mathrm{ot}}$ as $(conv^{\mathrm{ot}}, \omega_1^{\mathrm{ot}})$, it appends $conv^{\mathrm{ot}}$ to $conv$ and then returns $view = (conv, \omega_1^{\mathrm{ot}})$ as the answer to $\mathcal{B}$'s query. Adversary $\mathcal{B}$ now outputs a bit b', and $\mathcal{B}$ adopts this as its own output as well.

By assumption, the two adversaries above have simulators, respectively $\mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{OT}}$. We define simulator $\mathcal{S}$ for $\mathcal{B}$. On input $(1^k, x, y, \phi)$ it lets $(F, (X_1, \ldots, X_n), d) \leftarrow \mathcal{S}_{\mathcal{G}}(1^k, y, \phi)$ and records (F, d) as the first message in $conv$. It then lets

$$view^{\mathrm{ot}} \leftarrow \mathcal{S}_{\mathcal{OT}}(1^k, x, (X_1, \ldots, X_n), (|X_1|, |X_1|, \ldots, |X_n|, |X_n|)).$$

Parsing $view^{\mathrm{ot}}$ as $(conv^{\mathrm{ot}}, \omega_1^{\mathrm{ot}})$, it appends $conv^{\mathrm{ot}}$ to $conv$ and then returns $view = (conv, \omega_1^{\mathrm{ot}})$.

The case $i = 2$ is much easier because party 2 obtains nothing from party 1 besides what it gets from the execution of the OT protocol and thus security follows directly from the assumption that the OT protocol is secure. □

3.7.2 KDM-secure encryption

We re-establish Applebaum's result that projection-KDM security implies bounded-KDM security [2]. While our scheme is similar to the scheme of Applebaum or the scheme of of Barak, Haitner, Hofheniz, and Ishai (BHHI) [9], it actually improves the efficiency by an order of magnitude. For simplicity, we describe only the symmetric setting; the asymmetric setting is similar. In this section, ev always denotes the canonical circuit evaluation.

proc INITIALIZE()	**proc** KDM(j, f)	Game KDM
$K_1, K_2, \ldots K_\ell \leftarrow \mathcal{K}(1^k)$, $b \leftarrow \{0,1\}$	$x \leftarrow \mathsf{ev}(f, K_1 \parallel \cdots \parallel K_\ell)$	
	if $b = 1$ **then return** $\mathcal{E}_{K_j}(x)$ **else return** $\mathcal{E}_{K_j}(0^{\lvert x\rvert})$	

Figure 3.7.2: **Game for defining the KDM security.** Procedure FINALIZE(b') returns ($b = b'$).

KDM SECURITY. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme of key space $\{0,1\}^p$ and message space $\{0,1\}^s$. Let k be the security parameter. Consider the game in Fig. 3.7.2. An adversary $\mathcal{A}$, on 1^k, make queries KDM of the form (j, f) where $j \in \{1, \ldots, \ell\}$ for some ℓ that determines the number of keys, and string f encodes a function $\mathsf{ev}(f, \cdot)$ that maps a $p \cdot \ell$-bit string to an s-bit string. Finally, the adversary outputs a bit b'. Define $\mathbf{Adv}^{\mathrm{kdm}}_{\Pi,\ell}(\mathcal{A}, k) = 2\Pr[\mathrm{KDM}^{\mathcal{A}}(k)] - 1$. In the asymptotic statements, p, s, and ℓ are understood as polynomials $p(k), s(k)$, and $\ell(k)$. We say that Π is *KDM secure* if $\varepsilon(k) = \mathbf{Adv}^{\mathrm{kdm}}_{\Pi,\ell}(\mathcal{A}, k)$ is negligible for any PPT adversary $\mathcal{A}$ and for any polynomial ℓ.

Often, the choice of functions f cannot be arbitrary. Below are the types of restrictive KDM security that we discuss.

- **Projection-KDM security.** A function $h : \{0,1\}^n \to \{0,1\}^m$ is a *projection* if each of its output bit depends on at most one input bit. Scheme Π is *projection-KDM secure* if $\varepsilon(k) = \mathbf{Adv}^{\mathrm{kdm}}_{\Pi,\ell}(\mathcal{A}, k)$ is negligible for any polynomial ℓ and for any PPT adversary $\mathcal{A}$ that makes queries (j, f) such that $\mathsf{ev}(f, \cdot)$ is a projection.
- **Bounded-KDM security.** Scheme Π is *q-bounded KDM secure*, where q is a polynomial, if $\varepsilon(k) = \mathbf{Adv}^{\mathrm{kdm}}_{\Pi,\ell}(\mathcal{A}, k)$ is negligible for any polynomial ℓ and for any PPT adversary $\mathcal{A}$ that always make queries (j, f) such that f encodes a circuit of at most q gates, where q is also understood as a polynomial $q(k)$ in the asymptotic statements.

In the symmetric-key setting, the LPN scheme of Applebaum, Cash, Peikert, and Sahai (ACPS) [3] is a projection-KDM secure encryption scheme. In the asymmetric-key setting, one can use the scheme of Boneh, Halevi, Hamburg, and Ostrovsky (BHHO) [22] to instantiate a projection-KDM secure encryption scheme [5]. See the discussion of Applebaum [2,

[5] In BHHO's scheme, the public key is a list of group elements $(g_1, \ldots, g_r)$ and the private key is

```
00  proc E'(K, x)
01  n ← max{p · ℓ, s}, z ← x0^{n−|x|}
02  for i ∈ {n + 1, ..., n + q + 1} do
03     A(i) ← 1,  B(i) ← i − 1,  G_i(a, b) ← {return a}
04  for i ∈ {n + q + 2, ..., q + 2n} do
05     A(i) ← 1,  B(i) ← i − n − q,  G_i(a, b) ← {return b}
06  ID ← (n, n, q + n, A, B, G)
07  (F, e, d) ↞ Gb(1^k, ID),  X ← En(e, z)
08  return (F, d, E_K(X))

10  proc D'(K, y)
11  (F, d, Y) ← y
12  X ← D_K(Y)
13  z ← De(d, Ev(F, X))
14  return z[1 : s]
```

Figure 3.7.3: **Scheme $\Pi' = (\mathcal{K}, \mathcal{E}', \mathcal{D}') = \mathrm{S2B}[\Pi, \mathcal{G}, q, \ell]$ that is q-bounded KDM secure.** The scheme is built based on a projective garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ and a projection-KDM secure encryption $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. In lines 02–06, we construct an $(n + q)$-gate circuit ID computing the identity in $\{0,1\}^n$.

```
proc KDM(j, f)                                   Games G_0
(F, e, d) ↞ Gb(1^k, ID),  X ← En(e, C(K))
return (F, d, E_{K_j}(X))
```

```
proc KDM(j, f)                                   Games G_1
(F, e, d) ↞ Gb(1^k, C),  X ← En(e, K)
return (F, d, E_{K_j}(X))
```

```
proc KDM(j, f)                                   Games G_2
(F, e, d) ↞ Gb(1^k, C),  X ← En(e, 0^n)
return (F, d, E_{K_j}(0^{|X|}))
```

```
proc KDM(j, f)                                   Games G_3
(F, e, d) ↞ Gb(1^k, ID),  X ← En(e, C(0^n))
return (F, d, E_{K_j}(0^{|X|}))
```

Figure 3.7.4: **Code for proof of Theorem 3.7.2.**

Appendix B] for how to obtain projection-KDM security from known schemes.

THE SCHEME. Suppose that we have a symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that is projection-KDM secure, of key space $\{0,1\}^p$ and message space $\{0,1\}^s$. Fix ℓ and q. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a circuit projective garbling scheme that is prv.ind secure relative to Φ_{size}. We construct $\Pi' = (\mathcal{K}, \mathcal{E}', \mathcal{D}') = \mathrm{S2B}[\Pi, \mathcal{G}, q, \ell]$ that is q-bounded KDM secure, as shown in Fig. 3.7.3. The message space of Π' is also $\{0,1\}^s$.

Theorem 3.7.2. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective garbling scheme that is prv.ind secure relative to Φ_{size}. Fix ℓ and q. If Π is a simple-KDM secure symmetric encryption scheme then scheme $\Pi' = \mathrm{S2B}[\Pi, \mathcal{G}, q, \ell]$ is q-bounded KDM secure.

Proof. Let $\mathcal{A}$ be an adversary attacking Π'. To simplify the exposition, we first consider the case $\mathcal{A}$ makes only a single query. Then we will sketch how extend it to the general case.

$(g_1^{s_1}, \ldots, g_r^{s_r})$, with $s_1, \ldots, s_r \twoheadleftarrow \{0,1\}$. However, if we view $s = s_1 \cdots s_r$ as the private key then BHHO's scheme is projection-KDM secure.

SINGLE QUERY CASE. Suppose that $\mathcal{A}$ makes a single query (j, f). Let $n = \max\{p \cdot \ell, s\}$ and let $\mathbf{K} = K_1 \parallel \cdots \parallel K_\ell \parallel 0^{n-p\cdot\ell}$, where p is the length of each key $K_1, \ldots, K_\ell$. Let C be a circuit of n input wires, n output wires, and $q + n$ gates, such that $C(x) = \mathsf{ev}(f, x[1 : p \cdot \ell]) \parallel 0^{n-s}$. Note that

$$C(\mathbf{K}) = \mathsf{ev}(f, K_1 \parallel \cdots \parallel K_\ell) \parallel 0^{n-s} \ .$$

We construct an adversary $\mathcal{B}$ attacking Π and an adversary $\mathcal{B}'$ attacking $\mathcal{G}$ such that $\mathbf{Adv}^{\mathrm{kdm}}_{\Pi'}(\mathcal{A}, k) \le \mathbf{Adv}^{\mathrm{kdm}}_{\Pi}(\mathcal{B}, k) + 2\mathbf{Adv}^{\mathrm{prv},\Phi_{\mathrm{size}}}_{\mathcal{G}}(\mathcal{B}', k)$. Each adversary's running time is about that of $\mathcal{A}$, and $\mathcal{B}$ makes n queries.

The adversary $\mathcal{B}$ runs $\mathcal{A}$ and creates $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, C)$. Let h be a string such that $\mathsf{ev}(h, \cdot) = \mathsf{En}(e, \cdot)$. Since $\mathcal{G}$ is projective, function $\mathsf{ev}(h, \cdot)$ is a projection. Adversary $\mathcal{B}$ queries (j, h) to its oracle to receive an answer Y, and then returns (F, d, Y) to $\mathcal{A}$. It then outputs $\mathcal{A}$'s output bit. Then

$$\mathbf{Adv}^{\mathrm{kdm}}_{\Pi}(\mathcal{B}, k) = \Pr[G^{\mathcal{A}}_1(k) \Rightarrow 1] - \Pr[G^{\mathcal{A}}_2(k) \Rightarrow 1]$$

where games $G_0 - G_3$ are described in Fig. 3.7.4. (In game G_2, we make use of the fact that the length of the garbled input $X \leftarrow \mathsf{En}(e, x)$ is independent of x. So instead of writing $X \leftarrow \mathsf{En}(e, \mathbf{K})$, we let $X \leftarrow \mathsf{En}(e, 0^n)$.)

Next, we construct $\mathcal{B}'$. The adversary $\mathcal{B}'(1^k)$ first samples $K_1, K_2, \ldots, K_\ell \twoheadleftarrow \mathcal{K}(1^k)$. It chooses a bit $c \twoheadleftarrow \{0, 1\}$ and runs $\mathcal{A}(1^k)$ as a black box. If $c = 0$ it queries $\big(C, \mathrm{ID}, \mathbf{K}, C(\mathbf{K})\big)$. Otherwise, it queries $(\mathrm{ID}, C, C(0^n), 0^n)$. On receiving (F, X, d), it returns $(F, d, \mathcal{E}_{K_j}(U))$ to $\mathcal{A}$, where $U = X$ if $c = 0$, and $U = 0^{|X|}$ otherwise. It then outputs $\mathcal{A}$'s output bit. If $c = 0$, which occurs with probability $1/2$, then

$$\mathbf{Adv}^{\mathrm{prv},\Phi_{\mathrm{size}}}_{\mathcal{G}}(\mathcal{B}', k) = \Pr[G^{\mathcal{A}}_0(k) \Rightarrow 1] - \Pr[G^{\mathcal{A}}_1(k) \Rightarrow 1]$$

Otherwise, if $c = 1$ then

$$\mathbf{Adv}^{\mathrm{prv},\Phi_{\mathrm{size}}}_{\mathcal{G}}(\mathcal{B}', k) = \Pr[G^{\mathcal{A}}_2(k) \Rightarrow 1] - \Pr[G^{\mathcal{A}}_3(k) \Rightarrow 1]$$

Summing up, $\mathbf{Adv}^{\mathrm{kdm}}_{\Pi}(\mathcal{B}, k) + 2\mathbf{Adv}^{\mathrm{prv},\Phi_{\mathrm{size}}}_{\mathcal{G}}(\mathcal{B}', k) = \Pr[G^{\mathcal{A}}_0(k) \Rightarrow 1] - \Pr[G^{\mathcal{A}}_3(k) \Rightarrow 1] = \mathbf{Adv}^{\mathrm{kdm}}_{\Pi'}(\mathcal{A}, k)$.

General case. Suppose that $\mathcal{A}$ makes Q queries. We construct an adversary $\mathcal{B}$ attacking Π and an adversary $\mathcal{B}'$ attacking $\mathcal{G}$ such that $\mathbf{Adv}^{\mathrm{kdm}}_{\Pi'}(\mathcal{A}, k) \leq \mathbf{Adv}^{\mathrm{kdm}}_{\Pi}(\mathcal{B}, k) + 2Q\mathbf{Adv}^{\mathrm{prv},\Phi_{\mathrm{size}}}_{\mathcal{G}}(\mathcal{B}', k)$. The proof is similar to the single-query case, but there is a technical problem: adversary $\mathcal{B}'$ has only a single oracle query, but it receives Q queries from $\mathcal{A}$. The idea is to let $\mathcal{B}'$ choose $r \twoheadleftarrow \{0, 1, \ldots, Q-1\}$. For each of $\mathcal{A}$'s first r queries, instead of querying (f_0, f_1, x_0, x_1) to its oracle to get (F, X, d), adversary $\mathcal{B}'$ creates $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f_0)$ and lets $X = \mathsf{En}(e, x_0)$. For the next query, it queries the oracle. Finally, for each of the subsequent queries, instead of querying (f_0, f_1, x_0, x_1) to its oracle to get (F, X, d), it creates $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f_1)$ and lets $X = \mathsf{En}(e, x_1)$. Its running time is about that of $\mathcal{A}$ plus the time to garble the circuits in $\mathcal{A}$'s queries. Adversary $\mathcal{B}$, on the other hand, makes nQ queries to its oracle, and its running time is about that of $\mathcal{A}$. □

Comparison with BHHI. The scheme of BHHI and its proof are complex; see the discussion in Applebaum [2, Section 1.3.2] for criticism of BHHI's scheme. They rely on a *targeted encryption*, a public-key primitive that can be viewed as oblivious transfers with an additional KDM security property. BHHI instantiate targeted encryptions from either the scheme of BHHO [22] or the LWE-based scheme of ACPS [3]. From now on, assume that both targeted encryption and projection-KDM secure encryption are instantiated from BHHO's scheme for easy comparison. At the first glance, our scheme and BHHI's are almost the same. However, a closer look at BHHI's security proof [9, Theorem 5.2] reveals that it garbles a circuit of size $q + \max\{q \cdot \ell, s\} + \ell N$, where N is the size of a circuit that implements the decryption of BHHO's scheme. The number N is huge, making BHHI's scheme

extremely inefficient. (Recall that BHHO's decryption scheme makes $O(\log_2 |\mathbb{G}|)$ modular exponentiations, where $\mathbb{G}$ is the multiplicative group used in BHHO's scheme.)

The complexity and the inefficiency of BHHI's scheme are in part due to their security definition of garbling schemes. This notion is similar to our prv.ind relative to Φ_{size}, but the adversary must specify (f_0, x) and (f_1, x), that is, the two functions must have the *same* input. The slight change, however, leads to a tremendous difference, because if one instantiates out scheme from a garbling that satisfies BHHI's definition, then the proof no longer works. This might be the reason why BHHI did not propose our scheme, although it is a close and more natural variant of theirs.

Comparison with Applebaum. Applebaum's scheme is based on a simulation-based privacy notion. In his scheme, one runs Sim on x, where Sim is a simulator for the garbling scheme and x is the message. Both the (simulated) garbled function and the garbled input are encrypted, whereas our scheme encrypts only the latter. This makes Applebaum's scheme extremely inefficient because the size of the garbled function is large and all known encryptions that are projection-KDM secure in the standard model are slow. The inefficiency of Applebaum's scheme is due to his security definition of garbling schemes: the garbled function is lumped with the garbled input. This ignores the fact that the garbled function and garbled input have different roles and very different size. One can instead use our prv.sim notion for Applebaum's scheme and encrypt only the garbled input; this scheme is also secure. Still, its concrete performance is tied to the running time of Sim, which might be inefficient. In addition, this approach is less intuitive than ours, as the *simulated* garbled function, which might depend on the keys, is sent in the clear.

3.8 Related work

We do not attempt a comprehensive review of the literature (easily a monograph-length undertaking), but elaborate on some selected prior work.

Scheme	# of AES calls	# bits (BHHO)	Ciphertext size
BHHI [9]	$O((r+\ell N)\log(r+\ell N))$	$k \cdot \max\{p \cdot \ell, s\}$	$O(k(r+\ell N)\log(r+\ell N))$
Applebaum [2]	$O(r \log r)$	$O(kr \log r)$	$O(kr \log r)$
Our scheme	$O(r \log r)$	$k \cdot \max\{p \cdot \ell, s\}$	$O(kr \log r)$

Figure 3.7.5: **Comparison among schemes.** We base the three schemes on BHHO's scheme. Each scheme has message space $\{0,1\}^s$ and key space $\{0,1\}^p$. Here N is the size of a circuit implementing the decryption of BHHO, $r = q + \max\{p \cdot \ell, s\}$, and ℓ is the number of keys. We assume that the garbling scheme is implemented by Garble1 and Valiant's universal circuits [93]. The second column shows the number of AES calls to build the garbled function, the third columns the length of the message encrypted by BHHO's scheme, and the last column the ciphertext size.

RANDOMIZED ENCODINGS. Loosely related to garbling schemes, *randomized encodings* (initially *randomized polynomials*) begin with Ishai and Kushilevitz [54] and continue, with many definitional variants, in work by Applebaum, Ishai, Kushilevitz, and others [2, 4–7, 55, 56, 88]. The authors employ language like the following [4]: function $F(\cdot,\cdot)$ is a *randomized encoding* of $f(\cdot)$ if: (correctness) there's a PT algorithm De such that $\mathsf{De}(F(x,r)) = f(x)$ for almost all r; and (privacy) there's a PT algorithm Sim such that ensembles $F(x,\cdot)$ and $\mathsf{Sim}(f(x))$ are computationally indistinguishable. To be useful, encodings must have some extra properties,[6] for example, that every bit of $F(x,r)$ depends on at most one bit of x, a property that has been called *decomposability* [56]. Proven realizations meeting these requirements [4, 5] do not closely resemble conventional realizations of garbled circuits [70, 78].

There is a large gap, even syntactically, between the notion just given and a garbling scheme. Above, no language is provided to speak of the algorithm that transforms f to F; in contrast, the thing doing this transformation is at the center of a garbling scheme. Likewise absent from the syntax of randomized encodings is anything to speak to the representation of functions; for garbling schemes, representations are explicit and central. Finally, the syntax, unlike that of a garbling scheme, does not separate the garbling of a function and the creation of a garbled input, and indeed there is nothing corresponding to the latter, the same input x being fed to f or F. The minimalist syntax of randomized encodings works

[6] Otherwise, the definition is trivially met by setting $F(x,r) = f(x)$ and $\mathsf{De}(y) = \mathsf{Sim}(y) = y$.

well for some theory-centric applications, but does not allow one to speak of obliviousness and authenticity, to investigate the low-level efficiency of different garbling schemes, and to architect schemes to useful-in-practice abstraction boundaries.

Given the variety of related definitions, let us sketch another, the *decomposable randomized encodings* defined and used by Sahai and Seyalioglu [88]. (Despite identical names, this definition is different from that above, and different again from the decomposable randomized encodings of [56], say). The object of interest can be regarded as a pair of PT algorithms $(\mathsf{En}, \mathsf{De})$ where En maps the encoding of a boolean circuit $f\colon \{0,1\}^n \to \{0,1\}^m$ to a vector of strings $(X_1^0, X_1^1, \ldots, X_m^0, X_m^1) \leftarrow \mathsf{En}(1^k, f)$ for which decoding algorithm $\mathsf{De}(X_1^{x_1}, \ldots, X_m^{x_m})$ returns $f(x_1 \cdots x_n)$. The authors demand a PPT algorithm Sim for which the ensemble of $(X_1^{x_1}, \ldots, X_m^{x_m})$ tuples induced by $\mathsf{En}(1^k, f)$ and x is computationally indistinguishable from $\mathsf{Sim}(1^k, n, |f|, f(x))$. Translating to our language, one effectively assumes a projective scheme, a boolean circuit as input, and prv.sim security over Φ_{size}. The garbled function itself has been abstracted out of existence (in a realization, it would be dropped in the X_i^j values). Compared to a garbling scheme, one might note the lack of representation independence, granularity inadequate to speak of obliviousness, authenticity, garbled inputs, and low-level efficiency. The syntax can't handle the adaptive setting in Chapter 5, where the adversary receives the garbled circuit before it specifies the input.

Obliviousness and authenticity. Some prior papers exploit obliviousness and authenticity of garbled circuits to achieve desired applications: private medical diagnostics [10], verifiable computation and private verifiable computation [37], and correctable verifiable computation [6]. The notions are not seen as properties of a stand-alone primitive corresponding to a garbling scheme.

In the last of the works mentioned, Applebaum, Ishai, Kushilevitz [6] describe the following generic transformations from privacy to obliviousness and to authenticity. (1) Obliviousness: instead of garbling a circuit f, let g be a circuit such that $g(x \parallel r) = f(x) \oplus r$ for every $x \in \{0,1\}^n$ and $r \in \{0,1\}^m$, where $n = f.n$ and $m = f.m$. Then, choose $r \twoheadleftarrow \{0,1\}^m$,

run $(F, e, d) \leftarrow \mathsf{Gb}(g)$ and output $(F, (e, r), (d, r))$. The garbled input corresponding to x will be $X = e(x \parallel r)$. To decode, output $r \oplus \mathsf{De}(d, X)$. (2) Authenticity: instead of garbling a circuit f, let g be a circuit such that $g(x \parallel K) = f(x) \parallel \mathrm{MAC}_K(f(x))$ for any $x \in \{0,1\}^n$ and any key K. Then, choose a random key K, run $(F, e, d) \leftarrow \mathsf{Gb}(g)$, and output $(F, (e, K), (d, K))$. The garbled input corresponding to x will be $X = e(x \parallel K)$. To decode, compute $y \parallel t = \mathsf{De}(d, X)$ and output y if $t = \mathrm{MAC}_K(y)$, and output $\perp$ otherwise. Applied to Garble1, the transformations lead to schemes slightly (for (1)) or substantially (for (2)) less efficient that Garble2; and (2) requires a cryptographic assumption. More fundamentally, Applebaum *et al.* do not formalize any definition for the obliviousness or authenticity of a garbling scheme.

The only work that explicitly defines obliviousness and authenticity in this domain is a recent paper of Kamara, Mohassel, and Raykova [59]. Still, their syntax is designed specifically for their application; for example, a circuit's input is a pair (x_1, x_2), a garbled circuit's input is (X_1, X_2), and the encoding function takes an input x and an index $i \in \{1, 2\}$ and outputs the corresponding X_i. Their notion of obliviousness requires hiding only the input, while obv.ind and obv.sim require one to hide both the input and the function.

Obscuring topology. We are not the first to observe that conventional means to garble a circuit obscure each gate's function but not its topology. A 2002 paper of Pinkas [83, Section 2.3] already remarks that "In this form the representation reveals nothing but the wiring of the circuit". Later, Paus, Sadeghi, and Schneider [81] use the phrase "circuit topology" to name that which is revealed by conventional garbled circuits. Nevertheless, the topology of a circuit is never formalized, and nobody ever proves that that some particular scheme reveals only the topology. We are also the first to explain the equivalence between the prv.sim and prv.ind notions relative to Φ_{topo}.

Eclectic representations. Scattered through the literature one finds computational objects other than boolean circuits that are being garbled; examples include arithmetic circuits [7], branching programs [10], circuits with lookup tables [77], DFAs [92], and ordered

binary decision diagrams [67]. The range suggests, to us, that general-purpose definitions for garbling schemes ought not be tied to circuits.

CONCURRENT WORK. Concurrent work by Kamara and Wei (henceforth KW) investigates the garbling of *structured circuits* [60], a computational model they put forward resembling ordinary circuits except that gates perform operations on an arbitrary data structure. As part of this work, KW define what they too call a garbling scheme. Their syntax is similar to ours, but without the function ev. Over this syntax KW define IND1 and SIM1 security. These notions, unlike ours, ask only for input-hiding, not function hiding. They show these definitions are equivalent for *sampleable* circuits. KW go on to give adaptive versions of their definitions, IND2 and SIM2, and an unforgeability notion, UNF2. These definitions resemble the weaker form of the adaptive-security definitions (prv1, obv1, and aut1).

Although KW speak of circuits as finitary objects described by DAGs, they appear to have in mind families of circuits, indexed by a security parameter (otherwise, we do not know how to make sense of samplability, or phrases like *polynomial size circuits*). Unlike our treatment, circuits are not provided by the adversary; security notions are with respect to a given circuit. A garbling scheme is provided in KW, but not a "conventional" one: it garbles a structured circuit and is based on a collection of *structured encryption schemes*, a notion from Chase and Kamara [26]. For the protocol to make sense with respect to the definitions given, the latter should be reinterpreted as applying to structured circuits.

3.9 Universal circuits

Here we follow the description in Wegener's book [94, pp. 110–115]. An (n, q)-*universal circuit* is a circuit $\mathscr{U}$ having q distinguished gates $g_1, \ldots g_q$ such that:

- It takes two inputs f and x where $|x| = n$ and f is the encoding of a circuit of input length n and at most q gates.
- For any input (f, x), when we evaluate $\mathscr{U}$ on (f, x), the bit obtained at the outgoing

wire of g_i is exactly the bit obtained at the outgoing wire of gate i of f when we evaluate f on x.

A universal circuit must have size $\Omega(q \log q)$ because, by a counting argument, there are $\Omega(q2^q)$ circuits of q gates. Valiant [93] designs an (n, q)-universal circuit of fanout 4 and size $19(2q + m) \lg(2q + m) + 9q$, which is asymptotically optimal, where m is the number of outputs of the original circuit.

Other constructions of universal circuits are known [65, 87, 89], but their asymptotic size is larger. While these constructions are claimed to have better concrete efficiency than Valiant's for circuit size that GCs are still practical, this claim is recently demonstrated to be false [76] for real circuits of either small size (like AES, about 50K gates) or big size (like RSA-256, about 266M gates).

It is now folklore that one can hardwire a circuit $f = (n, m, q, A, B, G)$ to a (n, q)-universal circuit $\mathscr{U}$, and garble the resulting circuit $\mathscr{U}(f, \cdot)$ to leak nothing but the size of f. However, there is a technical annoyance, as in circuit $\mathscr{U}$ above, we do not specify the number of output wires. The obvious step is to indicate that in $\mathscr{U}(f, \cdot)$, the output wires are the outgoing wires of gates $g_{q-m+1}, \ldots, g_q$. This new circuit however violates the definition in Chapter 2, as its output wires are also incoming wires of some gates. So we need to add m more gates to have an equivalent circuit that conforms to the constraint above.

Chapter 4

Efficient Garbling

4.1 Introduction

In this chapter, we show how to construct and evaluate garbling schemes at unprecedented speeds. Our gains come from two main sources. On the cryptographic side, we describe garbling schemes that evaluate a gate using a single call to a fixed permutation, which can be instantiated by fixed-key AES. On the systems side, we exploit more efficient representations of circuits.

Recall that in Section 3.5, the workhorse of garbling schemes is a *dual-key cipher* (DKC). Many existing multiparty computation (MPC) protocols construct DKCs from a cryptographic hash. The advent of AES-NI support (AES new instructions) has made it natural to turn from hash functions to blockciphers for DKCs, and AES256 was the primitive used by Kreuter, Shelat, and Shen [66]. But we contend that the starting point best suited for exploiting AES-NI is not a blockcipher but a *cryptographic permutation*, which can be realized by fixed-key AES: $\mathrm{AES}_{\mathsf{c}}(\cdot)$ with c a fixed, non-secret key. An encryption key can be setup and, after, one has a pipeline into which 128-bit blocks can be fed.

To capitalize on this possibility, we construct garbling schemes in the random-permutation model (RPM) [86], meaning that all parties, adversary included, can access a single, fixed,

	$\mathbb{E}^{\pi}(A,B,T,X) =$	Ga			GaX			GaXR		
		T_{E}	T_{G}	K	T_{E}	T_{G}	K	T_{E}	T_{G}	K
A1	$\pi(K) \oplus K \oplus X$, with $K = A \oplus B \oplus T$	50.3	218	64.0	—	—	—	—	—	—
A2	$\pi(K) \oplus K \oplus X$, with $K = 2A \oplus 4B \oplus T$	52.1	221	64.0	**23.2**	**55.6**	11.5	23.9	56.4	8.64
A3	$\pi(K\|T)[1:k] \oplus K \oplus X$, with $K = A \oplus B$	93.7	242	40.0	—	—	—	—	—	—
A4	$\pi(K\|T)[1:k] \oplus K \oplus X$, with $K = 2A \oplus 4B$	97.9	246	40.0	34.2	62.7	7.20	35.0	63.3	**5.40**

Figure 4.1.1: **Efficiency of permutation-based garbling**. Data is from the JustGarble system, garbling a moderate-size circuit (a 36.5K gate AES circuit; 82% xor gates). Columns labeled T_{E} and T_{G} give the time to evaluate and garble using the specified protocol, measured in **cycles per gate** (**cpg**). Multiply by 0.3124 to get nanoseconds per gate on our test platform. Columns labeled K give the size of the garbled tables, measured in **bytes per gate** (**bpg**). The permutation π is always $\mathrm{AES}_{\mathsf{c}}(\cdot)$. Insecure possibilities are dashed.

random permutation, as well as its inverse. In Section 3.5, suitable DKCs can be built using a single call to the permutation, but this construction is not compatible with the free-xor trick. Here we further explore this idea, aiming for high efficiency (a single call to the permutation to evaluate a garbled gate), proven security, and the ability to incorporate existing optimizations, including free xor [64] and garbled-row reduction [84]. More specifically, we introduce a notion of a *σ-derived* DKC and then prove security of various (reasonably standard) garbling schemes under specified assumptions on the function σ. By instantiating σ in different RPM-based ways one obtains schemes that meet both our efficiency and security aims. Let us explain our main contributions in a bit more detail.

1. Garbling in the RPM. We begin by precisely specifying three garbling schemes: Ga, GaX, and GaXR. The first is based on the Garble1 scheme in Section 3.5. The scheme include the *point-and-permute* technique [85], which hijacks one bit of each token so that the agent evaluating the GC knows which "row" of the garbled gate to decrypt. GaX augments Ga with the *free-xor* technique [64], wherein XOR gates can be computed by xoring their incoming token. The savings can be large, as many circuits are rich in XOR gates, or can be refactored so. Finally, GaXR augments GaX with *garbled row reduction* [84], which reduces the size of a GC by arranging that one of the four rows of each garbled gate need not be stored: tokens are selected so as to make this ciphertext a constant.

In each of the three schemes the underlying primitive is a dual-key cipher. This is a

deterministic function $\mathbb{E}\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^k \to \{0,1\}^k$ taking keys A, B, a tweak T, and a plaintext X, returning a ciphertext $\mathbb{E}(A, B, T, X)$. All schemes (Ga, GaX, and GaXR) use at most four calls to $\mathbb{E}$ to garble a gate and at most one call to evaluate a gate. We must efficiently and securely construct the needed DKC.

Our DKC constructions are in the RPM; the DKC has oracle access to a random permutation $\pi\colon \{0,1\}^\ell \to \{0,1\}^\ell$. (An important challenge for security is that the adversary has access not only to π but also to π^{-1}.) This is the sole source of cryptographic hardness available. Our implementations set $\pi = \mathrm{AES}_{\mathsf{c}}(\cdot)$ for a fixed key c. Fig. 4.1.1 shows four constructions, with A1/A3 suitable for Ga and A2/A4 suitable for all three schemes. All of our DKC constructions employ a single call to π. We postpone a description of what $2A$ and $4B$ actually mean except to indicate that these are simple operations, a couple of shifts or the like, but *not* integer multiplication.

To validate the security of our schemes instantiated with our DKC constructions, a natural first thought is to prove security of the schemes in the random-oracle (RO) model (ROM) [20] and then show that the constructions of Fig. 4.1.1 are indifferentiable from ROs [31, 33, 73]. However, attacks show that the constructions are *not* indifferentiable from ROs. We have preferred them to constructions that are indifferentiable from ROs because the latter are less efficient. The performance gains we have achieved must accordingly be backed up by dedicated proofs.

Rather than provide many *ad hoc* proofs, we provide a unified framework that defines a class of DKCs we call σ-derived. All our instantiations fall in this class. We give conditions on σ sufficient to guarantee the security of Ga, GaX, and GaXR, all in the RPM. Our results use concrete security, giving formulas that bound an adversary's maximal advantage as a function of the effort it expends.

2. Vulnerabilities in existing constructions. It is common in this area to start from a basic, proven scheme, and then implement an instantiation, enhancement, or variant that is not itself proven. In particular, while there are proofs for some schemes that use the

free-xor method [29, 64], ours are the first proofs for schemes that simultaneously use both free xor and garbled row reduction.

Absence of proof can belie presence of error. We consider the dual-key cipher

$$\mathbb{E}^H(A,B,T,X) = H(A[1:k-1] \parallel T) \oplus H(B[1:k-1] \parallel T) \oplus X$$

for a cryptographic hash function H. This DKC was suggested for Fairplay [72], but claimed to work [64] with free xor [64]. We will later show that this not to be the case. Note that other authors have gone so far as to implement MPC using this DKC [84]; the construction has only been considered undesirable because it is less efficient than alternatives, not because its security was in doubt. Our view is that it is not possible to look at a DKC and reliably ascertain if it will work in a complex security protocol; assurance here requires proofs.

3. The JustGarble system. Prior implementation work has viewed MPC as the goal, with garbling implemented as a component. Our JustGarble system reflects our tenet, divorcing garbling from MPC to deliver a system whose goal is *just* optimized garbling. JustGarble aims to be a general-purpose tool for use not only in MPC, but also beyond.

JustGarble implements Ga, GaX, and GaXR with the DKCs of Fig. 4.1.1 and the DKCs' permutation instantiated with fixed-key AES. Among the system-level optimizations and choices in JustGarble, the most prominent is programmatically realizing the mathematical conventions for representing circuits in Chapter 2. The combination of faster DKCs and a simple representation of circuits results in impressive performance gains over previous implementations.

We have carried out a number of timing studies using JustGarble. The main one on which we report is described in Fig. 4.1.1. We built an AES128 circuit, a standard test case for this domain, and looked at the time to evaluate the circuit, T_{E}; the time to garble the circuit, T_{G}; and the size of the garbled tables of the circuit, K. Breaking with tradition for this domain, we prefer to give running times in cycles per gate (cpg), a measure that's at least a little more robust than time per gate or total time. Similarly, we report on circuit size in units of bytes per gate (bpg).

Fig. 4.1.1 highlights the best evaluation time, 23.2 cpg, and the best garbling time, 55.6 cpg. (As our processor runs at 3.201 GHz, this translates to 7.25 nsec/gate for evaluating the GC and 17.4 nsec/gate for garbling it.) The smallest garbled tables are also highlighted, 5.40 bpg. Garbled circuits themselves, which include more than garbled tables, are always 8 bpg larger.

As a point of reference, Huang, Evans, Katz, and Malka (HEKM) evaluate a similar AES circuit in around 2 μsec per gate [51, Section 7: 0.06 sec, online, about 30K gates]. They indicate 10 μsec per gate for very large circuits. Kreuter, Shelat, and Shen (KSS) [66], using a DKC based on AES256 and implemented with AES-NI processor support, report constructing a 31 Kgate AES-128 circuit in 80 msec, so 2.5 μsec per gate. These times are more than two orders of magnitude off of what we report. While such a comparison is in some ways unfair—as we have explained, HKEM and KSS build systems for MPC, not garbling schemes—the time discrepancy is vast, and prior MPC work has routinely maintained that circuit garbling and evaluation *are* key components of the total work done (and have thus been the locus of prior optimizations). We note that the HEKM and KSS figures are times spent on garbling and evaluation alone; they don't include time spent on, say, oblivious transfer or network overhead.

We obtain performance gains over previous implementations even if we drop into the JustGarble system one of the previously designed, comparatively slow DKCs. The main reason for this is our extremely simple representation of garbled circuit. Gates are not objects that communicate by sending messages, for example; they are indexes into an array. There is no queue of gates ready to be evaluated; gates are topologically ordered, so one just evaluates them in numerical order. We call the representation format we use SCD, for Simple Circuit Description. Its simplicity helps ensure that most of the work in garbling a circuit or evaluating a GC is actual cryptographic work, not overhead related to procedure invocation, message passing, bookkeeping, or the like.

We emphasize that JustGarble knows nothing of MPC, oblivious transfer, compiling

programs into circuits, or any of the other tasks associated to making a useful higher-level protocol. JustGarble is a building block. If offers but two services: garble a circuit already built by other means, and evaluate a GC on a garbled input.

4.2 Preliminaries

Dual-key ciphers. We extend the syntax of *dual-key ciphers* (DKC) in Section 3.5, providing these objects with oracles. Letting Ω be a set of functions π from $\{0,1\}^*$ to $\{0,1\}^*$, an (oracle-) DKC is a function $\mathbb{E}\colon\ \Omega \times \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^k \to \{0,1\}^k$ that associates to $\pi \in \Omega$ and $A, B \in \{0,1\}^k$ and $T \in \{0,1\}^\tau$ some permutation $\mathbb{E}^\pi(A,B,T,\cdot)\colon \{0,1\}^k \to \{0,1\}^k$.

Garbling schemes Ga, GaX, GaXR. The scheme we call Ga is based on an oracle DKC $\mathbb{E}^\pi\colon\ \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^k \to \{0,1\}^k$ whose inverse is denoted $\mathbb{D}$. We associate to $\mathbb{E}$ the RPM-model garbling scheme Ga[$\mathbb{E}$] of Fig. 4.2.1.

Scheme Ga is essentially scheme Garble1 in Section 3.5. The only difference is that (i) at output wires, the last bit of each token no longer coincides with its semantics, and consequently, the description of the decoding function d is a bit vector; the ith component is the last bit of the token of semantics 0 on the ith output wire, and (ii) each tweak of the DKC is the gate index, instead of a nonce. Garbling scheme GaX augments what we have described with the free-xor technique. Scheme GaXR additionally incorporates the row-reduction technique.

4.3 Instantiation overview

We discuss some of the challenges, and choices we make in response, with regard to garbling in the RPM.

The DKC $\mathbb{E}^H(A,B,T,X) = H(A\|B\|T) \oplus X$ is a natural starting point, where H is a

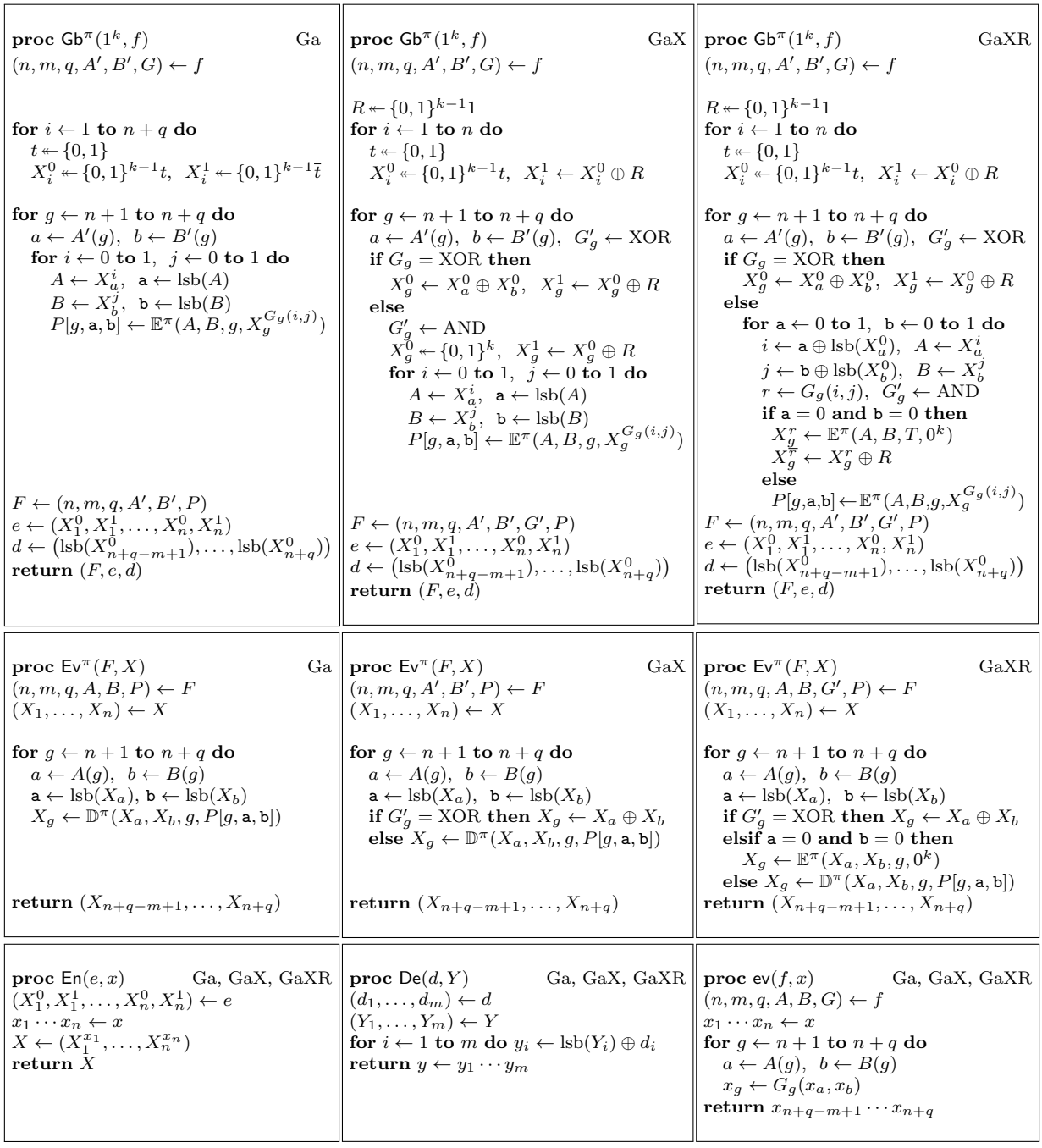

Figure 4.2.1: **Three RPM-based garbling schemes.** Schemes Ga, GaX, and GaXR have the same En, De, and ev procedures, but their own Gb and Ev procedures For a bit t, let $\{0,1\}^{k-1}t$ denote the set of k-bit strings whose last bit is t, and $\bar{t}$ the complement bit of t.

hash function. Our constructions can be seen as realizations of this approach, but based on a fixed-key blockcipher. Kreuter, Shelat, and Shen [66] had already considered $H(A\|B\|T) = \mathrm{AES256}_{A\|B}(T)$ where $|A| = |B| = |T| = 128$. Fixed-key AES provides a primitive π with only

a third the number of input bits as AES256.

One possibility is to build H from π in a manner that will render H indifferentiable from a RO [33, 73]. However, known constructions with this property will not be as efficient as we would like. We aim to use a Davies-Meyer type construction [74, 95], which applies the permutation only once. Such constructions are *not* indifferentiable from ROs [31, 33], necessitating considerable caution.

For simplicity we start by ignoring the tweak and considering the garbling of one-gate circuits. We present several natural constructions and show that they fail. We then present our constructions, and finally explain how to incorporate tweaks so as to handle circuits with an arbitrary number of gates.

INSTANTIATING GA. Consider instantiating the DKC of Ga from a permutation π by $\mathbb{E}^\pi(A, B, T, X) = \pi(A \oplus B) \oplus X$. The resulting scheme can be trivially broken, as follows. Suppose that we garble an AND gate, as illustrated on the top-left corner of Fig. 4.3.1, and suppose the adversary is given the garbled table and tokens A and C. First, it opens the third row to obtain token X. Next, let V be the ciphertext in the last row. Then the adversary can obtain token $D = \pi^{-1}(V \oplus X) \oplus A$. Likewise, it can obtain token B. Now the adversary can open every row of the garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary asks $(f_0, f_1, 00, 00)$ to GARBLE where f_0 is an AND gate and f_1 is a gate that always outputs 0. Following the idea above, the adversary open every row of the garbled table. If each row encrypts the same token then it outputs 1; otherwise, it outputs 0.

The attack arises because the adversary can invert $\pi(A \oplus D)$ to get D. To break this invertibility we employ the Davies-Meyer construction $\rho(K) = \pi(K) \oplus K$ to obtain the instantiation

$$\mathbb{E}^\pi(A, B, T, X) = \rho(A \oplus B) \oplus X \,. \tag{4.3.1}$$

We shall see in Theorem 4.4.1 that instantiation (4.3.1) indeed makes Ga secure, once the

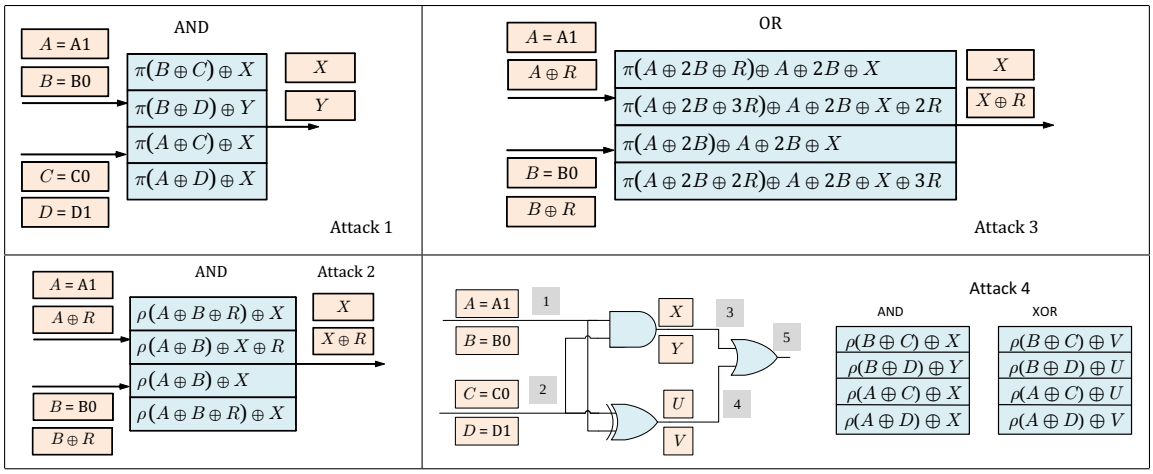

Figure 4.3.1: **Attacks on DKC instantiations.** Top-left: $\mathbb{E}^{\pi}(A, B, T, X) = \pi(A \oplus B) \oplus X$ for scheme Ga. Bottom-left: $\mathbb{E}^{\pi}(A, B, T, X) = \rho(A \oplus B) \oplus X$ for scheme GaX, with $\rho(K) = \pi(K) \oplus K$. Top-right: $\mathbb{E}^{\pi}(A, B, T, X) = \rho(A \oplus 2B) \oplus X$ for scheme GaX. Bottom-right: $\mathbb{E}^{\pi}(A, B, T, X) = \rho(A \oplus B) \oplus X$ for scheme Ga. The doubling here is multiplying in $\mathrm{GF}(2^k)$ by $\mathsf{x} = 0^{k-2}10$. In each wire, the top and bottom tokens have semantics 0 and 1 respectively.

tweaks are appropriately introduced.

Instantiating GaX. Yet instantiation (4.3.1) doesn't work for scheme GaX, even if the circuit remains a single gate. Here is an attack. Again we garble an AND gate. The illustration is given at the bottom-left corner of Fig. 4.3.1. Suppose the adversary is given the garbled table and tokens A and B. It first xors the ciphertexts in the second and third rows and obtains the string R. It then can open every row of the garbled table. Now all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary queries $(f_0, f_1, 00, 01)$ where f_0 is an AND gate and f_1 is a gate such that $f_1(a, b) = a$ for all $a, b \in \{0, 1\}$. Following the idea above, the adversary can open every row of the garbled table, regardless of the challenge bit. If there are three rows that encrypt the same token then it outputs 0; otherwise, it outputs 1.

The attack above arises because of a "symmetry" between tokens of the first and second incoming wires, leading to the use of $\rho(A \oplus B)$ twice to mask tokens of the output wire. One possible way to break this symmetry is to apply some simple operation to the token of the

second incoming wire before using it. For example, consider the instantiation

$$\mathbb{E}^{\pi}(A, B, T, X) = \rho(A \oplus 2B) \oplus X, \tag{4.3.2}$$

where doubling ($B \mapsto 2B$) is multiplying in $\mathrm{GF}(2^k)$ by the group element $\mathsf{x} = 0^{k-2}10$. The attack above is thwarted, because the ciphertext in the third row is $\rho(A \oplus 2B) \oplus X$ while that in the second row is now $\rho(A \oplus 2B \oplus 3R) \oplus X \oplus R$, where $3R$ means multiplying R by the group element $\mathsf{x} + 1 = 0^{k-2}11$ in $\mathrm{GF}(2^k)$.

Still, instantiation (4.3.2) can be broken as follows. See the illustration on the top-right corner of Fig. 4.3.1. Garble an OR gate. Suppose the adversary is given the garbled table and tokens A and B. First it opens the third row to obtain token X. Let V be the ciphertext in the first row. Query $V \oplus A \oplus 2B \oplus X$ to π^{-1}, and let K be the answer. Then, the adversary can obtain $R = K \oplus A \oplus 2B$. It can now open every row of the garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary queries $(f_0, f_1, 00, 01)$ where f_0 is an OR gate and f_1 is an AND gate. Following the idea above, the adversary can open every row of the garbled table, regardless of the challenge bit. Using the decoding function, the adversary can determine the semantics of the tokens on the output wire. If there are three rows that encrypt the token of semantics 1 then it outputs 1; otherwise, it outputs 0.

To thwart the attack above one can apply the multiplication in $\mathrm{GF}(2^k)$ to the first incoming token as well; for example, we can use the instantiation

$$\mathbb{E}^{\pi}(A, B, T, X) = \rho(2A \oplus 4B) \oplus X \tag{4.3.3}$$

where $4B$ means applying the doubling operation to B twice, that is, multiplying B by the group element $\mathsf{x}^2 = 0^{k-3}100$ in $\mathrm{GF}(2^k)$. The ciphertext in the first row will be $\pi(2A \oplus 4B \oplus 2R) \oplus 2A \oplus 4B \oplus X \oplus 3R$. Since $R \twoheadleftarrow \{0,1\}^{k-1}1$ is secret, the attack fails. We shall see in

Theorems 4.4.1 and 4.4.2 that instantiation (4.3.3) indeed makes both Ga and GaX secure, after the gate-number tweak is appropriately introduced.

THE NEED FOR THE TWEAK. Suppose now that one uses instantiation (4.3.1) for scheme Ga, but in a circuit of multiple gates. This leads to a new attack. Garble the circuit f illustrated at the bottom-right of Fig. 4.3.1. Suppose the adversary is given the garbled tables and tokens A and D. (In the illustration, only the garbled tables of the first two gates are shown.) It first opens the last rows in the first two tables to get tokens X and V. Next, it xors the ciphertexts in the third rows of the two first tables, and then xors the resulting string with X to get U. Likewise, the adversary can obtain Y. It now can open every row of the last garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security, in which the adversary queries $(f, f, 01, 11)$ to obtain (F, X, d). Following the idea above, regardless of the challenge bit, the adversary can open every row of the last garbled table. Using d, the adversary can determine the semantics of the tokens on the output wire. There is only one row of the last garbled table that encrypts the token of semantics 0. The token on wire 3 used as a key for this row must have semantics 0. The adversary then can determine the semantics of tokens on wire 3. Now evaluate F on X. If the token obtained on wire 3 during the evaluation has semantics 0 then output 0. Otherwise, output 1.

The attack above arises if the circuit contains two gates that have the same pair of incoming wires. We therefore introduce the tweak-based variants $\mathbb{E}^\tau(A, B, T, X) = \rho(A \oplus B \oplus T) \oplus X$ and $\mathbb{E}^\tau(A, B, T, X) = \rho(2A \oplus 4B \oplus T) \oplus X$ of instantiations (4.3.1) and (4.3.3), respectively, with the tweak being the gate index. We shall see in Theorems 4.4.1 and 4.4.2 that these tweak-based instantiations indeed make Ga secure, and the second one makes GaX secure.

Alternatively, for scheme Ga, one can avoid using tweaks by demanding that no two gates have the same pair of incoming wires. However, this condition is not sufficient when the free-xor trick is used, because one can arrange for distinct wires to carry the *same* pair

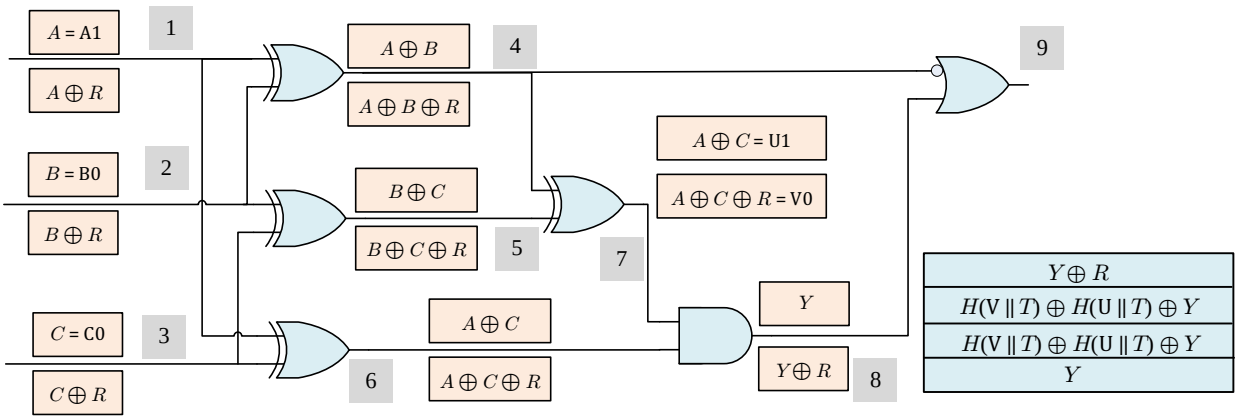

Figure 4.3.2: **An attack on GaX with DKC** $\mathbb{E}(A, B, T, X) = H(A[1\colon k-1] \parallel T) \oplus H(B[1\colon k-1] \parallel T) \oplus X$**.** In each wire, the top token has semantics 0, the bottom one has semantics 1. The table on the right is the garbled table of gate 8. Gate 9 negates the bit on wire 4, then ORs it with the bit on wire 8.

of tokens. For example, consider the circuit in Fig. 4.3.2. Wires 6 and 7 there have the same pair of tokens. This kind of subtle degeneracy serves to emphasize the need for proofs.

Other ways to double. Besides the multiplication in $\mathrm{GF}(2^k)$ (named D1 below) doubling may have several other interpretations, setting $2A$ to any of the following:

D1: $(A \ll 1) \oplus (A[1] \cdot \mathrm{const})$ *Finite field multiply*

D2: $A \ll 1$ *Logical left shift*

D3: $A \gg 1$ *Logical right shift*

D4: $A \lll 1$ *Circular left shift*

D5: $A \ggg 1$ *Circular right shift*

D6: $(A[1\colon \lfloor k/2 \rfloor] \ll 1) \,\|\, (A[\lfloor k/2 \rfloor + 1\colon k] \ll 1)$ *SIMD left*

D7: $(A[1\colon \lfloor k/2 \rfloor] \gg 1) \,\|\, (A[\lfloor k/2 \rfloor + 1\colon k] \gg 1)$ *SIMD right*

We will later show that all of these methods "work" for the schemes in this chapter, although the security bounds differ by a constant. In particular, we will identify a sufficient condition for the doubling map and a real number r associated to it, this number showing up in our bounds. The reason for attending to these different doubling methods is that "true" doubling has the best security bound, but its implementation is a bit slower than alternatives with slightly inferior bounds.

AN INSECURITY ISSUE IN PRIOR WORKS. Besides proposing free-xor, Kolesnikov and Schneider (KS) [64] propose two instantiations of a DKC, suggesting to set $\mathbb{E}^H(A, B, T, X)$ as either

$$H(A[1:k-1] \parallel B[1:k-1] \parallel T) \oplus X \quad \text{or} \tag{4.3.4}$$

$$H(A[1:k-1] \parallel T) \oplus H(B[1:\, k-1] \parallel T) \oplus X \tag{4.3.5}$$

where $H:\ \{0,1\}^* \to \{0,1\}^k$ is a hash function, to be modeled as a random oracle. KS effectively show that GaX, built on top of instantiation (4.3.4), leads to a secure two-party SFE protocol. They claim that one can use instantiation (4.3.5) as well. Pinkas, Schneider, Smart, and Williams (PSSW) [84] implement both instantiations; their garbling schemes are variants of Ga/GaX/GaXR, where each DKC's tweak is a nonce instead of the gate index. Subsequent works [49, 51, 57] use only (4.3.4) because of efficiency issues, but the authors apparently continue to believe that (4.3.5) works fine; see, for example, [27, p. 5] and [57, p. 7].

We now show that an adversary can completely break GaX if the DKC is instantiated by (4.3.5). Our attack also applies to the GaX/GaXR variants of PSSW based on (4.3.5). The key idea of the attack is that, as mentioned previously, when one uses free-xor trick, different wires in the circuit can be forced to share the same pair of tokens. Observe that if $A = B$ then instantiation (4.3.5) sends the plaintext in the clear, as $H(A[1:k-1] \parallel T) \oplus H(B[1:\, k-1] \parallel T) \oplus X = X$. Suppose that we garble the circuit f in Fig. 4.3.2. Wires 6 and 7 have the same pair of tokens. As shown in the garbled table of gate 8, we send both Y and $Y \oplus R$ in the clear, and there is no security whatsoever.

To translate the above to an attack of advantage 1 on prv security, the adversary queries $(f, f, 000, 100)$ to obtain (F, X, d). Following the idea above, the adversary obtains all tokens and opens every row of every garbled table. Using d, it can determine the semantics of the tokens on the output wire. There is only one row of the garbled table of gate 9 that

DKC	A1	A2				A3	A4			
doubling	—	D1	D2, D3	D4, D5	D6, D7	—	D1	D2, D3	D4, D5	D6, D7
regularity	1	1	4	1	16	1	1	4	1	16
strong regularity	—	1	4	4	16	—	1	4	4	16
inject indicator	1	1				0	0			

Figure 4.4.1: **Parameters for DKC instantiations.** The strong regularity of A1 and A3 is huge ($\delta = 2^k$); the corresponding entries are dashed.

encrypts the token of semantics 0. The token on wire 4 used as a key for this row must have semantics 1. The adversary therefore can determine the semantics of the tokens on wire 4. Now evaluate F on X. If the token obtained on wire 4 has semantics 0 then output 1, otherwise output 0.

4.4 Security of Ga, GaX and GaXR

We will justify the security of our schemes in a common framework. We define a class of DKCs that we call σ-derived. Under various conditions on the map σ, we prove security for our schemes.

σ-DERIVED DKCs. Let $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$ be a function. We say that $\mathbb{E}$ is σ-derived DKC if $\mathbb{E}^\pi(A,B,T,X) = (\pi(K) \oplus K)[1\colon k] \oplus X$ for $K = \sigma(A,B,T)$ and the function σ satisfies the following two conditions:

(i) $\sigma(A \oplus A^*, B \oplus B^*, T \oplus T^*) = \sigma(A,B,T) \oplus \sigma(A^*,B^*,T^*)$ for every $A, A^*, B, B^* \in \{0,1\}^k$ and $T, T^* \in \{0,1\}^\tau$, and

(ii) $\sigma(0^k, 0^k, T) \neq 0^\ell$ unless $T = 0^\tau$.

The *injectivity indicator* of σ is a number $\delta \in \{0,1\}$; it is 0 if and only if σ is tweak-wise injective, that is, $\sigma(A,B,T) \neq \sigma(A^*,B^*,T^*)$ whenever $T \neq T^*$. The *regularity* of σ is the smallest $r \in \mathbb{Z}^+$ such that

(iii) $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(x,0^k,0^\tau) = s] \leq r/2^k$ and also $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(0^k,x,0^\tau) = s] \leq r/2^k$ for every string $s \in \{0,1\}^\ell$.

The *strong regularity* of σ is the smallest $r \in \mathbb{Z}^+$ such that (iii) is satisfied and

(iv) $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(a \cdot x, b \cdot x, 0^\tau) \oplus x0^{\ell-k} = s] \le r/2^k$ and $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(x, x, 0^\tau) = s] \le r/2^k$ for every string $s \in \{0,1\}^\ell$ and every $(a,b) \in \{0,1\}^2$, where $0 \cdot x = 0^{|x|}$ and $1 \cdot x = x$.

Each of our DKC instantiations is a σ-derived DKC; the regularity, strong regularity, and injectivity indicator of its σ are shown in Fig. 4.4.1. This claim can be verified by a simple but tedious analysis. For example, consider scheme A2 with the doubling method D2. Its function σ is $\sigma(A,B,T) = 2A \oplus 4B \oplus T$, satisfying both (i) and (ii), and the injectivity indicator of this σ is 1. The regularity is 4, as $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ x \ll 1 = s] \le 2/2^k$ and $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ x \ll 2 = s] \le 4/2^k$ for every string $s \in \{0,1\}^k$. To verify that the strong regularity is also 4, suppose one wants to show that, say $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ (x \ll 1) \oplus x = s] \le 4/2^k$ for every string $s \in \{0,1\}^k$. Let $x = x_1 \cdots x_k$. Note that function $f(x) = (x \ll 1) \oplus x$ returns

$$(x_1 \oplus x_2) \parallel (x_2 \oplus x_3) \parallel \cdots \parallel (x_{k-1} \oplus x_k) \parallel x_k,$$

and thus it is a permutation on $\{0,1\}^k$. Since $x \twoheadleftarrow \{0,1\}^k$, it follows that $f(x)$ is also uniformly distributed over $\{0,1\}^k$. Hence the chance that $f(x) = s$ is at most $1/2^k$. See Section 4.7 for the complete analysis.

Security of Ga. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small regularity, then Ga[$\mathbb{E}$] is prv-secure over Φ_{topo}.

Theorem 4.4.1. Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the regularity and injectivity indicators of σ, respectively. Then

$$\mathbf{Adv}^{\text{prv.ind},\,\Phi_{\text{topo}}}_{\text{Ga}[\mathbb{E}]}(\mathcal{A},k) \le \frac{6qQ + 15q^2}{2^\ell} + \frac{30rQ + 84rq}{2^k} + \frac{\delta(42rQq + 69rq^2)}{2^k}$$

In the advantage formula above, we use the injectivity indicator δ to "safeguard" the term

$(Qq + q^2)/2^k$. For the DKC instantiation A3, our implementation uses $k = 80$, and in practice, q may go up to 2^{32}, say, as in recent works [51, 66]. The presence of the term $(Qq + q^2)/2^k$ for A3 would result in a poor bound. Fortunately, this term vanishes, because $\delta = 0$ for A3. The advantage for A3 is about $(Qq+q^2)/2^\ell + (Q+q)/2^k$, which is satisfactory for $\ell = 128$ and $k = 80$. In the DKC instantiation A1, for example, $\delta = 1$, but there we'll use $k = \ell = 128$, and the advantage becomes about $(Qq + q^2)/2^\ell$, which is very good.

To obtain the desirable bound above, the proof for Theorem 4.4.1, given in Section 4.6.1, is complex. Without care the advantage formula for $\mathbb{E} = \text{A3}$, for example, might easily include the term $Qq/2^k$ (without the guard of δ), which results in a poor bound for the choice $k = 80$.

Security of GaX. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small strong regularity, then GaX[$\mathbb{E}$] is prv-secure over Φ_{xor}. The proof is in Section 4.6.2.

Theorem 4.4.2. Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the strong regularity and injectivity indicators of σ, respectively. Then

$$\mathbf{Adv}^{\text{prv.ind},\,\Phi_{\text{xor}}}_{\text{GaX}[\mathbb{E}]}(\mathcal{A}, k) \le \frac{6qQ + 15q^2}{2^\ell} + \frac{36rQ + 108rq}{2^k} + \frac{\delta(48rQq + 84rq^2)}{2^k}$$

Security of GaXR. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small strong regularity, then GaXR[$\mathbb{E}$] is prv-secure over Φ_{xor}. The proof is in Section 4.6.3.

Theorem 4.4.3. Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the strong regularity and injectivity indicators of σ, respectively.

Then

$$\mathbf{Adv}^{\text{prv.ind},\,\Phi_{\text{xor}}}_{\text{GaXR}[\mathbb{E}]}(\mathcal{A}, k) \leq \frac{10qQ + 20q^2}{2^\ell} + \frac{36rQ + 123rq}{2^k} + \frac{\delta(48rQq + 94rq^2)}{2^k} \quad .$$

4.5 JustGarble and its performance

We have built a system, JustGarble, to realize the ideas described so far. The high speeds it achieves come from use of a fixed-key blockcipher and various implementation optimizations. We explore these factors here.

ARCHITECTURE. JustGarble follows our tenet that garbling should be decoupled from MPC, oblivious transfer, and the compilation of programs into circuits. The separation of concerns facilitates construction of an efficient tool, but it also necessitates caution when comparing reported speeds.

To facilitate speed and interoperability, JustGarble uses a circuit representation that is simple and easy to work with: SCD, for Simple Circuit Description. SCD closely follows the formulation of circuits in Chapter 2. An SCD file starts with values n, m, q, followed by arrays A, B, and G. If G is absent the file represents a topological circuit. For cross-language and cross-platform compatibility, values are encoded with MessagePack [36].

JustGarble consists of modules BUILD, GARBLE, and EVALUATE for building circuits, garbling them, and evaluating garbled circuits, respectively. The BUILD module can be used to construct circuits, working at the level of individual gates or collections of them. Constructed circuits are written to SCD files. The GARBLE module realizes the Gb algorithm of Ga, GaX, or GaXR. It can use any of the DKCs specified in this chapter. GARBLE takes in an SCD-described circuit $f = (n, m, q, A, B, G)$ and produces the garbled tables P that comprise the final component of the associated garbled circuit $F = (n, m, q, A, B, P)$. The EVALUATE module takes in a topological circuit $f^- = (n, m, q, A, B)$, the garbled tables P needed to complete this, and a garbled input X. It produces the garbled output Y.

JustGarble also includes simple routines to realize De, which maps the garbled output Y to the corresponding output y with the help of d.

The garbling module does not use the operating system to generate the pseudorandom bits needed for tokens; such a choice would not be cryptographically secure. Instead, pseudorandom bits are also generated by fixed-key AES, now operating in counter mode. At present, we use a different AES key than that employed for the random permutation underlying the selected DKC. We have verified that it would also work, cryptographically, to employ the same key for these conceptually distinct tasks. But there would be a small quantitative security loss, and the proofs would need to deal with this complication. With GaX-A2, the measured time savings from using the same permutation is at most 0.3 cpg.

JustGarble utilizes hardware AES support through AES-NI [48]. The system is written in C and employs compiler intrinsics to access SSE4 [53] instructions and 128-bit registers, which hold and manipulate the tokens. JustGarble is entirely open-source and freely available for download [58].

Experimental methodology. We run our experiments on an x86-64 Intel Core i7-970 processor clocked at 3.201 GHz with a 12MB L3 cache. Tests are compiled with `gcc` version 4.6, optimization level `-O3`, with support for SSE4 and AES-NI instructions through the `-sse4` and `-maes` flags. The tests are run in isolation, with processor frequency scaling turned off. We use the `rdtsc` instruction to count cycles.

We run tests in batches of 1000 runs each, noting the median of the times recorded in the runs. This process is repeated for 1000 batches, and the final time reported is the mean of the batch medians. The cache is warm during the tests from initial runs. The standard deviation of the batch medians does not exceed 0.25 cpg in any of the experiments.

AES-circuit benchmarks. We measure garbling and evaluation speeds on a circuit computing $\mathrm{AES128}_K(X)$ (hereafter simply AES) for a particular key K. This corresponds to a GC-based SFE of AES where the first party holds K and prepares a circuit for the second party, who holds X and wants to compute $\mathrm{AES}_K(X)$. We choose this setting because it has

Tool	$\mathbb{E}(A,B,T,X) =$	Ga		GaX		GaXR	
		T_{E}	T_{G}	T_{E}	T_{G}	T_{E}	T_{G}
Perm	$\pi(K) \oplus K \oplus X$, with $K = 2A \oplus 4B \oplus T$	52.1	221	23.2	55.6	23.9	56.4
Cipher	$E(K,T) \oplus X$, with $K = A\|\|B$	256	991	60.1	172	58.7	171
Hash	$H(K\|T)[1:k] \oplus X$, with $K = A\|\|B$	875	3460	161	566	160	568

Figure 4.5.1: **Permutation-based, blockcipher-based, and hash-based garbling**. The T_{E} (time to evaluate) and T_{G} (time to garble) values are in mean **cycles per gate (cpg)** using the subject AES circuit. The first method, A2, is based on a permutation $\pi\colon \{0,1\}^k \to \{0,1\}^k$. The permutation chosen is fixed-key AES128. The second method, from KSS [66], uses a blockcipher $E\colon \{0,1\}^{2k} \times \{0,1\}^k \to \{0,1\}^k$. The selected blockcipher is AES256. The last method, employed in [51], builds a DKC from a hash $H\colon \{0,1\}^* \to \{0,1\}^k$. The hash function chosen is SHA-1.

been used as a benchmark in prior work [49, 51, 66, 72], and hence helps compare our system with existing ones.

We build the AES circuit as described in HEKM [51]. The key is first expanded into 1280 bits. Conceptually, this is done locally by the party holding the key. We use a different S-box circuit [25] than HEKM, which results in a smaller AES circuit. This is not significant; as we measure speed in cycles per gate, small differences in circuit size are unlikely to have a noticeable effect on speed as long as the fraction of xor gates is little changed. Overall, our AES circuit has 36,480 gates, of which 29,820 (82%) are xor.

The evaluation and garbling speeds of A1, A2, A3, and A4 are listed in Fig. 4.1.1. For A2 we use doubling method D7; for A4, we use D3. These choices will be explained shortly. The fastest among our constructions, GaX with A2, evaluates the AES circuit at 23.2 cpb (7.25 ns/gate) and garbles it at 55.6 cpg (17.4 ns/gate). Overall, this comes to 637 μs for garbling the AES circuit and 264 μs for evaluating.

Schemes A3 and A4 are a little slower than A1 and A2. Part of the speed difference may be due to JustGarble being better optimized for 128-bit tokens. There may be memory-alignment overheads in dealing with 10-byte tokens: SSE4 instructions can have higher read and write latencies when data is not 16-byte aligned [53].

The sizes S_P we report in Fig. 4.1.1 measure only the contribution from the garbled tables: $S_P = |P|/8q$. Focusing on this value is justifiable because, in MPC applications, the

Circuit	Gates	Xor gates	T_{E}	T_{G}
MEXP-16	0.21 M	0.14 M	44.1	91.6
MEXP-32	1.75 M	1.15 M	45.3	96.3
MEXP-64	14.3 M	9.31 M	44.6	95.8
EDT-255	15.5 M	9.11 M	48.4	101.3

Figure 4.5.2: **Performance on larger circuits.** Evaluation times (T_{E}) and garbling times (T_{G}) are in median **cycles per gate** using GaX-A2. The modular exponentiation (MEXP) and edit distance (EDT) circuits are described in text. Gate counts are in **millions of gates** (1M = 1 million gates).

other components of the GC, its topology, will be known and need not be communicated. Regardless, the size of the GC that JustGarble makes will always be $S_F = S_P + 8$ bytes, as gates are represented as four-byte numbers and we need to record two of these per gate—one for each of arrays A and B. Here we ignore the space to store n, m, q.

For the DKC A2, we implement doubling in many ways; see the definition for methods D1–D7 in Section 4.3. We find D6 and D7 the fastest, followed by D2 and D3, then D4 and D5, and finally D1. The speed of D6 and D7 (SIMD shift) is due to the availability of a matching SSE4 instruction. The speed difference between the fastest and slowest doubling methods is $\Delta T_E \approx 7$ cpg and $\Delta T_G \approx 11$ cpg. We find this significant enough to trade a small quantity in the security bound, which is why we select A2 with D7 doubling. For the DKC A4, which uses 10-byte tokens, similar experiments lead us to select the doubling scheme D3.

Larger circuits. The size of the garbled table for each non-xor gate ranges from 30 bytes (GaXR with A3, A4) to 64 bytes (GaX with A1, A2). This means that even circuits with hundreds of thousands of gates can fit in the processor's L3 cache during evaluation. However, if the circuit is too big to fit entirely in the cache, per-gate garbling and evaluation times will increase.

To understand the performance of JustGarble on circuits larger than the cache size, we measured garbling and evaluation times of the modular exponentiation (MEXP) ("RSA circuits") and edit distance (EDT) circuits of KSS with various input sizes. We used GaX

with A2 (henceforth GaX-A2); see Fig. 4.5.2. The MEXP-ℓ circuit takes inputs a and b and returns $a^b \bmod c$ for $c = 1^8 0^{\ell-9} 1$. The EDT-m circuit takes as inputs two m-bit strings and returns their edit distance as a $(\lg m)$-bit integer. We obtained these circuits by patching the KSS compiler to produce outputs in SCD format. The garbling and evaluation times (in cycles per gate) are higher than the measured values for the AES circuit due to higher latencies involved in reading data directly from main memory. However, JustGarble is still several times faster than what KSS report. Taking RSA-32 as an example, KSS report a garbling time of 4.53 seconds, which translates to 6546 cpg, while JustGarble uses 91.6 cpg, a 70x speedup.

We can draw a quick estimate for the threshold beyond which circuits fill out of the cache, by computing the amount of memory that the evaluation module needs for evaluating a circuit of a given size. Consider the GaX-A2 evaluation of a circuit with n inputs and q gates, and λq of these XOR. In the evaluation module, gates and wires are represented by 32 byte and 16 byte data structures. Evaluating a circuit with q gates will use about $(64+32+16)(1-\lambda)q+(32+16)q\lambda = 112(1-\lambda)q+48q\lambda$, ignoring input and output labels and other small objects in memory. If the evaluating machine has (lowest level) cache size C bytes, then circuits with $q < C/(112 - 64\lambda)$ gates can reside in the cache. This matches with what we observe experimentally: in our test machine with a 12MB L3 cache, we run tests with families of circuits composed of a chain of AND gates. We find that the cost per gate remained fairly constant up until $q = 10^5$, followed by a rather sudden jump of 30 cpg (n was fixed at 1000 and $m = 1$).

At present, JustGarble cannot handle circuits that are too big to fit in main memory. An obvious direction for future work is extending JustGarble with a streaming mode of operation that can garble and evaluate large circuits by keeping only a small portion in memory at any given point.

Comparisons. JustGarble garbles and evaluates moderately-sized circuits about two orders of magnitude faster than what recent MPC implementations of HKEM and KSS

report [51, 66]. For evaluating an AES circuit, the best previously-reported figure comes from KSS [66], garbling the circuit in 80 ms. The fastest among our own constructions, GaX using A2, does the job in 638 μs. We note that both systems use AES-NI and SSE4 instructions and the free-xor optimization, and that, in both cases, the reported times are for garbling alone, excluding other operations and network overhead. One reason JustGarble performs better is that it spends less time on non-cryptographic operations, by which we mean all operations other than the DKC computations. Moreover, using a fixed-key DKC like A2 results in a sizable gain in performance, in spite of the large percentage of xor gates (82%) in the AES circuit. We measured the contributions of both of these factors as below.

JustGarble spends about 23% and 43% of its time on non-cryptographic operations when GaXR-A2 does garbling and garbled-circuit evaluation, respectively. In contrast, KSS measure AES256 (with AES-NI) overhead at 225 cycles per invocation but report an overall GaXR garbling time of over 6000 cpg, suggesting that close to 95% of the garbling time is non-cryptographic overhead. The reduced overhead is likely connected to our simple representation of circuits, one consequence of which is the absence of a need to maintain a queue of ready gates. A downside of this simple circuit representation is that, unlike HEKM and KSS, JustGarble cannot handle circuits that do not fit in memory.

To measure the contribution of the DKC itself we implemented within JustGarble the blockcipher-based DKC from KSS and the hash-function based DKC from HEKM; see Fig. 4.5.1. Let us focus on GaXR, as free-xor and garbled-row reduction are both employed in the MPC systems of KSS and HEKM. Comparing the first and second rows, the DKC-attributable speedup we get by using a permutation instead of a blockcipher is 2.5-fold improvement in evaluation time and 3-fold improvement in garbling time. Comparing the first and the third rows, the DKC-attributable speedup we get by using a permutation instead of a cryptographic hash function is 6.7-fold improvement in evaluation time and 10-fold improvement in garbling time. One may conclude that the improved DKCs play a large role in our performance gains—a factor of about 2.5 to 10—yet more mileage is obtained through

other aspects of JustGarble.

4.6 Postponed proofs

4.6.1 Proof of Theorem 4.4.1

In our code, a procedure with the keyword "private" is a local code of the caller, and thus cannot be invoked by the adversary. It can be viewed as a function-like macro in C/C++ programming language. That is, it still has read/write access to the variables of the caller, even if these variables are not its parameters. Consider games G_0–G_2 in Fig. 4.6.1. They share the same code for procedure Garble, but each has a different implementation of a local procedure GarbleRow. The adversary $\mathcal{A}$ makes queries to procedures Π and Π^{-1} to access an ideal permutation π, which is implemented lazily. Wlog, assume that $q + Q \leq 2^{k-2}/r$; otherwise the theorem is trivially true.

We reformulate game $\mathrm{PrvInd}_{\mathrm{Ga},\Phi_{\mathrm{topo}},k}$ as game G_0. Recall that in the scheme Ga, each wire i carries tokens X_i^0 and X_i^1 with semantics 0 and 1 respectively. If wire i ends up having value (semantics) v_i in the computation $v \leftarrow \mathsf{ev}(f_c, x_c)$, where c is the challenge bit, then token $X_i^{v_i}$ becomes visible to $\mathcal{A}$ while $X_i^{\overline{v}_i}$ stays invisible. Game G_0 makes this explicit. It picks for each wire i a "visible" token and an "invisible" one. Each garbled row that can be opened by visible tokens will be built directly in Garble. To construct each other garbled row, we invoke the "private" procedure GarbleRow, which inherits all variables of Garble.

We explain the game chain up until the terminal game. $\triangleright \mathrm{G}_0 \to \mathrm{G}_1$: the two games are identical until either game sets *bad*. In these games, we sample a uniformly random string S and want to set $\pi(K)$ to $K \oplus S$. This may cause inconsistency if $\pi(K)$ or $\pi^{-1}(K \oplus S)$ is already defined, triggering *bad*. In this case, G_0 resets S to the consistent value, but game G_1 does nothing. Hence in game G_1, a point $v \in \mathrm{Ran}(\pi)$ may have several preimages, and in that case $\pi^{-1}[v]$ means an arbitrary preimage.

proc GARBLE(f_0, f_1, x_0, x_1)
$(n, m, q, A', B', G) \leftarrow f_c$
for $i \leftarrow 1$ **to** $n + q$ **do**
  $v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$, $t_i \twoheadleftarrow \{0,1\}$, $X_i^{v_i} \twoheadleftarrow \{0,1\}^{k-1} t_i$, $X_i^{\overline{v}_i} \twoheadleftarrow \{0,1\}^{k-1} \overline{t}_i$
for $g \leftarrow n+1$ **to** $n+q$, $i \leftarrow 0$ **to** 1, $j \leftarrow 0$ **to** 1 **do**
  $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
  $A \leftarrow X_a^i$, $B \leftarrow X_b^j$, $\mathtt{a} \leftarrow \mathrm{lsb}(A)$, $\mathtt{b} \leftarrow \mathrm{lsb}(B)$, $K \leftarrow \sigma(A, B, g)$
  if $i = v_a$ **and** $j = v_b$ **then** $P[g, \mathtt{a}, \mathtt{b}] \leftarrow (\Pi(K) \oplus K)[1\colon k] \oplus X_g^{v_g}$
  else $P[g, \mathtt{a}, \mathtt{b}] \leftarrow$ GARBLEROW()
$F \leftarrow (n, m, q, A', B', P)$, $X \leftarrow (X_1^{v_1}, \ldots, X_n^{v_n})$
$d \leftarrow \big(\mathrm{lsb}(X_{n+q-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0)\big)$
return (F, X, d)

---

private proc GARBLEROW()
$S \twoheadleftarrow \{0,1\}^\ell$
if $K \in \mathrm{Dom}(\pi)$ **or** $S \oplus K \in \mathrm{Ran}(\pi)$ **then**
  $bad \leftarrow \mathsf{true}$
  $S \leftarrow \Pi(K) \oplus K$ ⟵ Use in game G_0
$Y \leftarrow S[1\colon k] \oplus X_g^{G_g(i,j)}$, $\pi[K] \leftarrow S \oplus K$
return Y

proc $\Pi(u)$ Game G_0 / Game G_1
if $u \notin \mathrm{Dom}(\pi)$ **then** $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \notin \mathrm{Ran}(\pi)$ **then**
  $u \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Dom}(\pi)$, $\pi[u] \leftarrow v$
return $\pi^{-1}[v]$

---

private proc GARBLEROW()
$S \twoheadleftarrow \{0,1\}^\ell$, $Y \leftarrow S[1\colon k] \oplus X_g^{G_g(i,j)}$
$\mathrm{BadDom} \leftarrow \mathrm{BadDom} \cup \{K\}$
$\mathrm{BadRan} \leftarrow \mathrm{BadRan} \cup \{K \oplus S\}$
return Y

proc $\Pi(u)$ Game G_2
if $u \in \mathrm{BadDom}$ **then** $bad \leftarrow \mathsf{true}$
if $u \notin \mathrm{Dom}(\pi)$ **then** $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \in \mathrm{BadRan}$ **then** $bad \leftarrow \mathsf{true}$
if $v \notin \mathrm{Ran}(\pi)$ **then**
  $u \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Dom}(\pi)$, $\pi[u] \leftarrow v$
return $\pi^{-1}[v]$

Figure 4.6.1: **Games for the proof of Theorem 4.4.1.** Each set is initialized to be $\emptyset$. Initially, procedure INITIALIZE() samples the challenge bit $c \twoheadleftarrow \{0,1\}$.

We now bound the chance that G_1 sets *bad*. Consider the ith invocation of GARBLEROW. It triggers *bad* to true if its string K falls into $\mathrm{Dom}(\pi)$ or $S \oplus K$ falls into $\mathrm{Ran}(\pi)$, with $S \twoheadleftarrow \{0,1\}^\ell$. Since $|\mathrm{Ran}(\pi)| \le (Q + q + i - 1)$, the latter happens with probability at most $(Q + i + q - 1)/2^\ell$. Let $K = \sigma(A, B, g)$. We claim that the chance that $K \in \mathrm{Dom}(\pi)$ is at most $6r/2^k + N_i r(2\delta + 1)/2^k$, where N_i is the size of $\mathrm{Dom}(\pi) \cap \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$, which is at most $|\mathrm{Dom}(\pi)| \le Q + q + i - 1$. By union bound, the chance that G_1 sets *bad* is at most

$$\begin{aligned} & \sum_{i=1}^{3q} \frac{Q+q+i-1}{2^\ell} + \frac{6r}{2^k} + \frac{rN_i(2\delta+1)}{2^k} \\ \leq \quad & \frac{3qQ+7.5q^2}{2^\ell} + \frac{3rQ+30rq}{2^k} + \frac{\delta(9rqQ+22.5rq^2)}{2^k} \ . \end{aligned}$$

The last inequality is obvious if $\delta = 1$, as $N_i \leq Q+q+i-1$. To justify it for the case $\delta = 0$, note that for each string s, there is at most one value g such that $s \in \{\sigma(x,y,g) \mid x,y \in \{0,1\}^k\}$. Hence when we sum up the numbers N_i, because the invocations of GarbleRow use each tweak value at most 3 times, we count each point in $\mathrm{Dom}(\pi)$ at most 3 times, and thus the sum is at most $3|\mathrm{Dom}(\pi)| \leq 3(Q+4q)$.

We now justify the claim above. Consider the moment that procedure Garble makes the ith call to GarbleRow. Let D_1 be the set of points in $\mathrm{Dom}(\pi)$ created by adversarial queries before its querying Garble, and let D_2 be the set of points in $\mathrm{Dom}(\pi)$ created by procedure Garble so far. Then $D_1 \cup D_2 = \mathrm{Dom}(\pi)$. Recall that $K = \sigma(A,B,g) = \sigma(A,0^k,0^\tau) \oplus \sigma(0^k,B,0^\tau) \oplus \sigma(0^k,0^k,g)$. Because $A \twoheadleftarrow \{0,1\}^k$ and r is the regularity of σ, it follows that $\Pr[\sigma(A,0^k,0^\tau) = s] \leq r/2^k$ for any string $s \in \{0,1\}^\ell$. Since A is independent of B and all points in D_1, the chance that $K \in D_1$ is at most $rN_i/2^k$.

What remains is to show that $\Pr[K \in D_2] \leq 6r/2^k + 2N_i r\delta/2^k$. Consider an arbitrary point $K^* \in D_2$. Let $K^* = \sigma(A^*,B^*,g^*)$. If $A \equiv A^*$ and $B \equiv B^*$ then K and K^* belong to different gates, and thus $g \neq g^*$. Hence $K \oplus K^* = \sigma(A,B,g) \oplus \sigma(A,B,g^*) = \sigma(0^k,0^k,g \oplus g^*) \neq 0^\ell$. Otherwise, wlog, suppose that $A[1\colon k-1]$ is independent of B, A^*, and B^*. For any string $s \in \{0,1\}^\ell$, as r is the regularity of σ, there are at most r strings x such that $\sigma(x,0^k,0^\tau) = s$. Given B, A^*, and B^*, because each but the last bit of A is still uniformly random, the conditional probability that A falls into one of the r strings above is at most $2r/2^k$, and thus the conditional probability that $\sigma(A,0^k,0^\tau) = s$ is at most $2r/2^k$. Hence $\Pr[K = K^*] \leq 2r/2^k$. Moreover, if the injectivity indicator $\delta = 0$ and $g \neq g^*$ then $K \neq K^*$. In other words, $\Pr[K = K^*] \leq 2r/2^k$ if $g = g^*$, and $\Pr[K = K^*] \leq 2r\delta/2^k$ otherwise.

Summing up, $\Pr[K \in D_2] \leq 6r/2^k + 2N_i r\delta/2^k$, because there are most three elements of D_2 using the tweak g.

$\triangleright \mathrm{G}_1 \rightarrow \mathrm{G}_2$: in game G_1 we write $\pi[K] \leftarrow S \oplus K$, but game G_2 omits this step. In addition, we maintain two sets BadDom and BadRan that are initialized to the empty sets. Each call to GarbleRow will add K to BadDom and $S \oplus K$ to BadRan. The two games are identical until G_2 sets *bad*, that is, when $\mathcal{A}$ happens to query $\Pi(u)$ with $u \in$ BadDom, or $\Pi^{-1}(v)$ with $v \in$ BadRan. Since G_2 samples S at random, and doesn't store it in π, the output of GarbleRow() is uniformly random, independent of the token that S masks.

We now bound the chance that G_2 sets *bad*. Consider an arbitrary point $K \in$ BadDom. It has a corresponding point $K \oplus S \in$ BadRan. Let $K = \sigma(A, B, g)$. Either A or B must be invisible. Wlog, suppose that A is invisible. Condition on the output of Garble. Initially, as each but the last bit of A is still uniformly random and the regularity of σ is r, the conditional probability that $K = s$ is at most $2r/2^k$ for any string $s \in \{0,1\}^\ell$. Consider a query u to Π. Each prior adversarial query to Π or Π^{-1} removes at most a value of K. Hence since there are at most $q+Q$ queries to Π and Π^{-1} (procedure Garble only queries Π for q rows that can be opened by visible tokens), the chance that u hits K is at most

$$\frac{2r/2^k}{1 - 2(Q+q)r/2^k} = \frac{2r}{2^k - 2r(Q+q)} \leq 4r/2^k,$$

where the last inequality is due to the assumption $Q + q \leq 2^{k-2}/r$. By union bound, the chance that $u \in$ BadDom is at most $12rq/2^k$. However, if the injectivity indicator $\delta = 0$, then there is at most one possible value of g such that $u \in \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$, and consequently, $\Pr[u \in \text{BadDom}] \leq 12r/2^k$ because each tweak value is used at most three times in BadDom. Hence in general, $\Pr[u \in \text{BadDom}] \leq 12r(q\delta + 1)/2^k$. Likewise, for each query v to Π^{-1}, the chance that $v \in$ BadRan is at most $12r(q\delta + 1)/2^k$. By union bound, the chance that game G_2 sets *bad* is at most

$$12r(Q+q)(q\delta+1)/2^k = (12rQ + 12rq)/2^k + \delta(12rQq + 12rq^2)/2^k \ .$$

Analysis of game G_2. The output of game G_2 is independent of the challenge bit c.

proc GARBLE(f_0, f_1, x_0, x_1)
$(n, m, q, A', B') \leftarrow \Phi_{\text{topo}}(f_0)$, $v_{q+n-m+1} \cdots v_{q+n} \leftarrow \mathsf{ev}(f_0, x_0)$
for $i \leftarrow 1$ **to** $n + q$ **do**
  $t_i \twoheadleftarrow \{0,1\}$, $V_i \twoheadleftarrow \{0,1\}^{k-1} t_i$, $I_i \twoheadleftarrow \{0,1\}^{k-1} \overline{t}_i$
for $i \leftarrow n + q - m + 1$ **to** $n + q$ **do**
  $X_i^{v_i} \leftarrow V_i$, $X_i^{\overline{v}_i} \leftarrow I_i$
for $g \leftarrow n + 1$ **to** $n + q$ **do**
  $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
  for $(A, B) \in \{V_a, I_a\} \times \{V_b, I_b\}$ **do**
    $\mathtt{a} \leftarrow \text{lsb}(A)$, $\mathtt{b} \leftarrow \text{lsb}(B)$, $K \leftarrow \sigma(A, B, g)$
    if $A = V_a$ **and** $B = V_b$ **then** $Y \leftarrow (\Pi(K) \oplus K)[1\colon k] \oplus V_g$ **else** $Y \twoheadleftarrow \{0,1\}^k$
    $P[g, \mathtt{a}, \mathtt{b}] \leftarrow Y$
$F \leftarrow (n, m, q, A', B', P)$, $X \leftarrow (V_1, \ldots, V_n)$
$d \leftarrow \big(\text{lsb}(X^0_{n+q-m+1}), \ldots, \text{lsb}(X^0_{n+q})\big)$
return (F, X, d)

Figure 4.6.2: **Rewritten game G_2 of the proof of Theorem 4.4.1.** This game depends solely on the topological circuit $f^- = \Phi_{\text{topo}}(f_0) = \Phi_{\text{topo}}(f_1)$ and the output $v = \mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. Procedures Π and Π^{-1} lazily implement a random permutation and its inverse, respectively.

Hence $\Pr[\mathrm{G}_2^{\mathcal{A}}(k)] = 1/2$. To justify this, from a topological circuit f^- and the final output $v = \mathsf{ev}(f_c, x_c)$, which is independent of c, we can rewrite the code of procedure GARBLE of game G_2 as shown in Fig. 4.6.2. There, we refer to the visible token of wire i as V_i, and its invisible counterpart as I_i, omitting the semantics of these tokens. Each garbled row is an independent, uniformly random string, except for rows that can be opened by visible tokens. Summing up,

$$\begin{aligned}\mathbf{Adv}^{\text{prv.ind}, \Phi_{\text{topo}}}_{\text{Ga}[\mathbb{E}]}(\mathcal{A}, k) &= 2(\Pr[\mathrm{G}_0^{\mathcal{A}}(k)] - \Pr[\mathrm{G}_2^{\mathcal{A}}(k)]) \\ &\leq \frac{6qQ + 15q^2}{2^\ell} + \frac{30rQ + 84rq}{2^k} + \frac{\delta(42rQq + 69rq^2)}{2^k} \ .\end{aligned}$$

4.6.2 Proof of Theorem 4.4.2

Wlog, assume that $Q + q \leq 2^{k-2}/r$; otherwise the theorem is trivially true. The proof is similar to that of Theorem 4.4.1. Consider games G_0–G_2 in Fig. 4.6.3. Each game has exactly the same procedures GARBLEROW, Π, and Π^{-1} as the corresponding game in Fig. 4.6.1 of the proof of Theorem 4.4.1. The only change is to add free-xor trick to the common procedure

proc GARBLE(f_0, f_1, x_0, x_1)
$(n, m, q, A', B', G) \leftarrow f_c$, $R \twoheadleftarrow \{0,1\}^{k-1}1$
for $i \leftarrow 1$ **to** $n+q$ **do** $v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$
for $i \leftarrow 1$ **to** n **do** $X_i^{v_i} \twoheadleftarrow \{0,1\}^k$, $X_i^{\overline{v}_i} \twoheadleftarrow X_i^{v_i} \oplus R$
for $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
  if $G_g = \text{XOR}$ **then** $G'_g \leftarrow \text{XOR}$, $X_g^{v_g} \leftarrow X_a^{v_a} \oplus X_b^{v_b}$, $X_g^{\overline{v}_g} \leftarrow X_g^{v_g} \oplus R$
  else $G'_g \leftarrow \text{AND}$, $X_g^{v_g} \twoheadleftarrow \{0,1\}^k$, $X_g^{\overline{v}_g} \leftarrow X_g^{v_g} \oplus R$
    for $i \leftarrow 0$ **to** 1, $j \leftarrow 0$ **to** 1 **do**
      $A \leftarrow X_a^i$, $B \leftarrow X_b^j$, $\mathtt{a} \leftarrow \text{lsb}(A)$, $\mathtt{b} \leftarrow \text{lsb}(B)$, $K \leftarrow \sigma(A, B, g)$
      if $i = v_a$ **and** $j = v_b$ **then** $P[g, \mathtt{a}, \mathtt{b}] \leftarrow (\Pi(K) \oplus K)[1:k] \oplus X_g^{v_g}$
      else $P[g, \mathtt{a}, \mathtt{b}] \leftarrow$ GARBLEROW()
$F \leftarrow (n, m, q, A', B', G', P)$, $X \leftarrow (X_1^{v_1}, \ldots, X_n^{v_n})$
$d \leftarrow \big(\text{lsb}(X_{n+q-m+1}^0), \ldots, \text{lsb}(X_{n+q}^0)\big)$
return (F, X, d)

---

private proc GARBLEROW()
$S \twoheadleftarrow \{0,1\}^\ell$
if $K \in \text{Dom}(\pi)$ **or** $S \oplus K \in \text{Ran}(\pi)$ **then**
  $bad \leftarrow \mathsf{true}$
  $S \leftarrow \Pi(K) \oplus K$ ⟵ Use in game G_0
$Y \leftarrow S[1:k] \oplus X_g^{G_g(i,j)}$, $\pi[K] \leftarrow S \oplus K$
return Y

proc $\Pi(u)$ Game G_0 / Game G_1
if $u \notin \text{Dom}(\pi)$ **then** $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \text{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \notin \text{Ran}(\pi)$ **then**
  $u \twoheadleftarrow \{0,1\}^\ell \backslash \text{Dom}(\pi)$, $\pi[u] \leftarrow v$
return $\pi^{-1}[v]$

---

private proc GARBLEROW()
$S \twoheadleftarrow \{0,1\}^\ell$, $Y \leftarrow S[1:k] \oplus X_g^{G_g(i,j)}$
$\text{BadDom} \leftarrow \text{BadDom} \cup \{K\}$
$\text{BadRan} \leftarrow \text{BadRan} \cup \{K \oplus S\}$
return Y

proc $\Pi(u)$ Game G_2
if $u \in \text{BadDom}$ **then** $bad \leftarrow \mathsf{true}$
if $u \notin \text{Dom}(\pi)$ **then** $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \text{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \in \text{BadRan}$ **then** $bad \leftarrow \mathsf{true}$
if $v \notin \text{Ran}(\pi)$ **then**
  $u \twoheadleftarrow \{0,1\}^\ell \backslash \text{Dom}(\pi)$, $\pi[u] \leftarrow v$
return $\pi^{-1}[v]$

Figure 4.6.3: **Games for the proof of Theorem 4.4.2.** Each set is initialized to be $\emptyset$. Initially, procedure INITIALIZE() samples the challenge bit $c \twoheadleftarrow \{0,1\}$.

GARBLE. Let L be the union of $\{1, \ldots, n\}$ and $\{g \mid n+1 \le g \le n+q \text{ and } G_g \neq \text{XOR}\}$. Visible tokens on wires $i \in L$ are chosen at random, and thus are independent. For each visible token V, there is a unique subset $\mathcal{V}$ of L such that V is the checksum of visible tokens of wires $i \in \mathcal{V}$. Then, the string R is independent of all visible tokens. Below, for any random variable $Z \in \{0,1\}^k$, if there is $\widetilde{Z} \in \{Z, Z \oplus R\}$ such that $\widetilde{Z}$ is the checksum of some visible tokens then we call $\widetilde{Z}$ the *visible match* of Z. Define the *flip bit* of Z to be the bit z such that $\widetilde{Z} = Z \oplus z \cdot R$. We call each string K that procedure GARBLE creates a *seed*.

```
proc GARBLE(f_0, f_1, x_0, x_1)
(n, m, q', A', B', G') ← Φ_xor(f_0)
v_{q+n−m+1} ··· v_{q+n} ← ev(f_0, x_0),  R ↞ {0,1}^{k−1}1
for i ← 1 to n + q do V_i ↞ {0,1}^k,  I_i ↞ V_i ⊕ R
for g ← n + 1 to n + q do
  a ← A'(g),  b ← B'(g)
  if G'_g = XOR then V_g ← V_a ⊕ V_b,  I_g ← V_g ⊕ R
  else
    for (A, B) ∈ {V_a, I_a} × {V_b, I_b} do
      a ← lsb(A),  b ← lsb(B),  K ← σ(A, B, g)
      if A = V_a and B = V_b then Y ← (Π(K) ⊕ K)[1: k] ⊕ V_g else Y ↞ {0,1}^k
      P[g, a, b] ← Y
for i ← n + q − m + 1 to n + q do X_i^{v_i} ← V_i,  X_i^{v̄_i} ← I_i
F ← (n, m, q, A', B', P),  X ← (V_1, ..., V_n)
d ← (lsb(X^0_{n+q−m+1}), ..., lsb(X^0_{n+q}))
return (F, X, d)
```

Figure 4.6.4: **Rewritten game G_2 of the proof of Theorem 4.4.2.** This game depends solely on $f' = \Phi_{\mathrm{xor}}(f_0) = \Phi_{\mathrm{xor}}(f_1)$ and the output $v = \mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. Procedures Π and Π^{-1} lazily implement a random permutation and its inverse, respectively.

The output of G_2 is independent of the challenge bit, and thus $\Pr[\mathrm{G}_2^{\mathcal{A}}(k)] = 1/2$. To justify this, from $f' = \Phi_{\mathrm{xor}}(f_c)$ and the final output $v = \mathsf{ev}(f_c, x_c)$, which is independent of c, we can rewrite the code of procedure GARBLE of game G_2, as shown in Fig. 4.6.4. There, we refer to the visible token of wire i as V_i, and its invisible counterpart as I_i, omitting the semantics of these tokens. Each garbled row is an independent, uniformly random string, except for rows that can be opened by visible tokens.

Hence by union bound and Lemmas 4.6.1 and 4.6.3 below,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{xor}}}_{\mathrm{GaX}[\mathbb{E}]}(\mathcal{A}, k) &= 2(\Pr[\mathrm{G}_0^{\mathcal{A}}(k)] - \Pr[\mathrm{G}_2^{\mathcal{A}}(k)]) \\ &\leq \frac{6qQ + 15q^2}{2^\ell} + \frac{36rQ + 108rq}{2^k} + \frac{\delta(48rQq + 84rq^2)}{2^k}.\end{aligned}$$

Lemma 4.6.1. The chance G_1 sets *bad* is at most $(3qQ + 7.5q^2)/2^\ell + (6rQ + 42rq)/2^k + \delta(12rQq + 30rq^2)/2^k$.

Proof. Consider the ith invocation of GARBLEROW. It triggers *bad* to true if its seed K falls into $\mathrm{Dom}(\pi)$ or $S \oplus K$ falls into $\mathrm{Ran}(\pi)$, with $S \twoheadleftarrow \{0,1\}^\ell$. The chance that $K \oplus S \in \mathrm{Ran}(\pi)$ is at most $|\mathrm{Ran}(\pi)|/2^\ell \leq (Q+q+i-1)/2^\ell$. Let D_1 be the set of points in $\mathrm{Dom}(\pi)$ created by

adversarial queries before its querying to GARBLE, and let D_2 be the set of points in $\mathrm{Dom}(\pi)$ created by procedure GARBLE so far. Then $D_1 \cup D_2 = \mathrm{Dom}(\pi)$. Let $K = \sigma(A, B, g)$, and let N_i be the size of $\mathrm{Dom}(\pi) \cap \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$, which is at most $Q + q + i - 1$. Below, we'll show that $\Pr[K \in D_1] \leq 2rN_i/2^k$ and $\Pr[K \in D_2] \leq 6r/2^k + 2rN_i\delta/2^k$. By union bound, the chance that G_1 sets *bad* is at most

$$\begin{aligned} & \sum_{i=1}^{3q} \frac{Q+q+i-1}{2^\ell} + \frac{6r}{2^k} + \frac{2rN_i(\delta+1)}{2^k} \\ \leq \; & \frac{3qQ + 7.5q^2}{2^\ell} + \frac{6rQ + 42rq}{2^k} + \frac{\delta(12rQq + 30rq^2)}{2^k} \; . \end{aligned}$$

The last inequality is obvious if $\delta = 1$, since $N_i \leq Q + q + i - 1$. To justify it for the case $\delta = 0$, note that for each string s, there is at most one value g such that $s \in \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$. Hence when we sum up the numbers N_i, because the GARBLEROW calls use each tweak value at most 3 times, we count each point in $\mathrm{Dom}(\pi)$ at most 3 times, and thus the sum is at most $3|\mathrm{Dom}(\pi)| \leq 3(Q + 4q)$.

First, we'll show that $\Pr[K \in D_1] \leq 2rN_i/2^k$. Let $\widetilde{A}, \widetilde{B}$ be the visible matches and a, b be the flip bits of A and B respectively. Since either A or B must be invisible, $(a, b) \neq (0, 0)$. We claim that $\Pr[\sigma(a \cdot R, b \cdot R, 0^\tau) = s] \leq 2r/2^k$ for any string $s \in \{0,1\}^\ell$. To justify this claim, note that as the strong regularity of σ is r, there are at most r strings x in $\{0,1\}^k$ such that $\sigma(a \cdot x, b \cdot x, 0^\tau) = s$. Because every bit of R, except the last, is uniformly random, the chance that R is one of the r strings above is at most $2r/2^k$. Since $K = \sigma(\widetilde{A} \oplus a \cdot R, \widetilde{B} \oplus b \cdot R, g) = \sigma(\widetilde{A}, \widetilde{B}, g) \oplus \sigma(a \cdot R, b \cdot R, 0^\tau)$, and R is independent of $\widetilde{A}, \widetilde{B}$ and all points in D_1, the chance that $K \in D_1$ is at most $2rN_i/2^k$. To bound the chance that $K \in D_2$, we'll show that any two seeds are unlikely to be equal.

Lemma 4.6.2. For any two seeds that procedure GARBLE creates, the chance that they are equal is at most $2r/2^k$.

Proof. Consider two seeds $K = \sigma(A, B, g)$ and $K^* = \sigma(A^*, B^*, g^*)$. Then $K \oplus K^* = \sigma(A \oplus A^*, B \oplus B^*, g \oplus g^*)$. Let C_0 and C_1 be the visible matches and c_0 and c_1 be the flip bits of $A \oplus A^*$ and $B \oplus B^*$ respectively. Suppose that $(c_0, c_1) \neq (0, 0)$. Then

$$\begin{aligned} K \oplus K^* &= \sigma(C_0 \oplus c_0 \cdot R, C_1 \oplus c_1 \cdot R, g \oplus g^*) \\ &= \sigma(C_0, C_1, g \oplus g^*) \oplus \sigma(c_0 \cdot R, c_1 \cdot R, 0^\tau) \ . \end{aligned}$$

As the strong regularity of σ is r and every bit of R, except the last, is uniformly random, the chance that $\Pr[\sigma(c_0 \cdot R, c_1 \cdot R, 0^\tau) = s] \leq 2r/2^k$ for any string $s \in \{0,1\}^\ell$. Since R is independent of all visible tokens, the chance that $K \oplus K^* = 0^\ell$ is at most $2r/2^k$. On the other hand, consider the case that $c_0 = c_1 = 0$. Let $\mathcal{A}$ be the subset of L such that C_0 is the checksum of visible tokens of wires $i \in \mathcal{A}$, and define $\mathcal{B}$ for C_1 likewise. If $\mathcal{A} = \mathcal{B} = \emptyset$ then $A \equiv A^*$ and $B \equiv B^*$, and thus K and K^* must belong to different gates. Then $g \neq g^*$ and $K \oplus K^* = \sigma(0^k, 0^k, g \oplus g^*) \neq 0^\ell$. Otherwise, if $\mathcal{A} \cup \mathcal{B} \neq \emptyset$ then let j be an arbitrary element of $\mathcal{A} \cup \mathcal{B}$. Let $a = 1$ if $j \in \mathcal{A}$, and let $a = 0$ otherwise. Likewise, let $b = 1$ if $j \in \mathcal{B}$, and let $b = 0$ otherwise. Then

$$K \oplus K^* = \sigma(a \cdot V_j, b \cdot V_j, 0^\tau) \oplus \sigma\Big(\bigoplus_{i \in \mathcal{A} \setminus \{j\}} V_i, \bigoplus_{i \in \mathcal{B} \setminus \{j\}} V_i, g \oplus g^* \Big),$$

where V_i is the visible token on wire i. As $(a, b) \neq (0, 0)$, every bit of V_j is uniformly random, and the strong regularity of σ is r, it follows that $\Pr[\sigma(a \cdot V_j, b \cdot V_j, 0^\tau) = s] \leq r/2^k$ for any string $s \in \{0,1\}^\ell$. Hence $\Pr[K = K^*] \leq 2r/2^k$ as claimed. □

What remains is to show that $\Pr[K \in D_2] \leq 6r/2^k + 2rN_i\delta/2^k$. Consider an arbitrary seed $K^* \in D_2$. Let $K^* = \sigma(A^*, B^*, g^*)$. From Lemma 4.6.2, $\Pr[K = K^*] \leq 2r/2^k$. On the other hand, if the injectivity indicator $\delta = 0$ and $g \neq g^*$ then $\Pr[K = K^*] = 0$. Thus $\Pr[K = K^*] \leq 2r/2^k$ if $g = g^*$, and $\Pr[K = K^*] \leq 2r\delta/2^k$ otherwise. By union bound, $\Pr[K \in D_2] \leq 6r/2^k + 2rN_i\delta/2^k$, because there are most three elements of D_2 using the

tweak g. □

Lemma 4.6.3. The chance G_2 sets *bad* is at most $(12rQ + 12rq)/2^k + \delta(12rQq + 12rq^2)/2^k$

Proof. Consider an arbitrary seed $K \in$ BadDom. It has a corresponding point $K \oplus S \in$ BadRan. Let $K = \sigma(A, B, g)$ and $S[1:k] = Y \oplus Z$, where Y is the value of the garbled row corresponding to K, and Z is the token that $S[1:k]$ masks. Let $\widetilde{A}, \widetilde{B}, \widetilde{Z}$ be the visible matches and a, b, z be flip bits of A, B, Z respectively. Since either A or B is invisible, $(a, b) \neq (0, 0)$. Then $K = \sigma(\widetilde{A}, \widetilde{B}, g) \oplus \sigma(a \cdot R, b \cdot R, 0^\tau)$ and

$$K \oplus S = \sigma(\widetilde{A}, \widetilde{B}, g) \oplus \sigma(a \cdot R, b \cdot R, 0^\tau) \oplus z \cdot R0^{\ell-k} \oplus \left((Y \oplus \widetilde{Z}) \parallel S[k+1:\ell]\right) .$$

Since the strong regularity of σ is r, a value of K or $K \oplus S$ corresponds to at most r possible values of R. Initially, there are 2^{k-1} equally likely values of R. Each query to Π or Π^{-1} removes at most r values of R. Hence, as there are at most $Q + q$ queries to Π and Π^{-1}, for each query to Π, the chance that it hits K is at most $r/(2^{k-1} - r(Q+q)) \leq r/2^{k-2}$, where the last inequality is due to the assumption that $Q + q \leq 2^{k-2}/r$. Thus, the chance that this query hits a point in BadDom is at most $3r(q\delta + 1)/2^{k-2}$. This claim is obvious if $\delta = 1$, as $|\text{BadDom}| \leq 3q$. To justify this for $\delta = 0$, note that there is at most one tweak whose corresponding seeds K can be hit by the query, and in BadDom, each tweak is used for at most three points. Likewise, for each query to Π^{-1}, the chance that it hits a point in BadRan is at most $3r(q\delta + 1)/2^{k-2}$. Hence, the chance that G_2 sets *bad* is at most $12(Q + q)r(q\delta + 1)/2^k$. □

4.6.3 Proof of Theorem 4.4.3

The proof is similar to that of Theorem 4.4.2. Consider games $G_0 - G_3$ in Figures 4.6.5 and 4.6.6. We reformulate game $\text{PrvInd}_{\text{GaXR},\Phi_{\text{xor}},k}$ as game G_0, with visible tokens and invisible ones. Here garbled rows that can be opened by visible tokens require using procedure EncodeRow.

proc GARBLE(f_0, f_1, x_0, x_1)
$(n, m, q, A', B', G) \leftarrow f_c$, $R \twoheadleftarrow \{0,1\}^{k-1}1$
for $i \leftarrow 1$ **to** $n+q$ **do** $v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$
for $i \leftarrow 1$ **to** n **do** $X_i^{v_i} \twoheadleftarrow \{0,1\}^k$, $X_i^{\overline{v}_i} \twoheadleftarrow X_i^{v_i} \oplus R$
for $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A'(g)$, $b \leftarrow B'(g)$, $G'_g \leftarrow$ AND
  if $G_g = $ XOR **then** $G'_g \leftarrow$ XOR, $X_g^{v_g} \leftarrow X_a^{v_a} \oplus X_b^{v_b}$, $X_g^{\overline{v}_g} \leftarrow X_g^{v_g} \oplus R$
  else
    for $\mathtt{a} \leftarrow 0$ **to** 1, $\mathtt{b} \leftarrow 0$ **to** 1 **do**
      $i \leftarrow \mathtt{a} \oplus \mathrm{lsb}(X_a^0)$, $j \leftarrow \mathtt{b} \oplus \mathrm{lsb}(X_b^0)$, $A \leftarrow X_a^i$, $B \leftarrow X_b^j$, $K \leftarrow \sigma(A, B, g)$
      if $i = v_a$ **and** $j = v_b$ **then** $S \leftarrow$ ENCODEROW()
else $S \leftarrow$ GARBLEROW()
      if $\mathtt{a} \neq 0$ **or** $\mathtt{b} \neq 0$ **then** $P[g, \mathtt{a}, \mathtt{b}] \leftarrow S[1\colon k] \oplus X_g^{G_g(i,j)}$
      else $X_g^{G_g(i,j)} \leftarrow S[1\colon k]$, $X_g^{1-G_g(i,j)} \leftarrow S[1\colon k] \oplus R$
$F \leftarrow (n, m, q, A', B', G', P)$, $X \leftarrow (X_1^{v_1}, \ldots, X_n^{v_n})$
$d \leftarrow \big(\mathrm{lsb}(X_{n+q-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0)\big)$
return (F, X, d)

private proc ENCODEROW()
$S \leftarrow \Pi(K) \oplus K$
if $K \in$ Seeds **then** *bad* $\leftarrow$ true, $\boxed{S \twoheadleftarrow \{0,1\}^\ell}$
elsif $K \in$ Coll **then**
  bad $\leftarrow$ true, $\boxed{S \leftarrow \mathrm{Map}[K] \oplus K}$
$\pi[K] \leftarrow S \oplus K$, Seeds $\leftarrow$ Seeds $\cup \{K\}$
return S

private proc GARBLEROW()
$S \twoheadleftarrow \{0,1\}^\ell$
if $K \in \mathrm{Dom}(\pi)$ **or** $S \oplus K \in \mathrm{Ran}(\pi)$ **then**
  bad $\leftarrow$ true
  $S \leftarrow \Pi(K) \oplus K$ $\longleftarrow$ Use in game G_0
$\pi[K] \leftarrow S \oplus K$, Seeds $\leftarrow$ Seeds $\cup \{K\}$
return S

proc $\Pi(u)$ Game G_0 / $\boxed{\text{Game } \mathrm{G}_1}$
if $u \notin \mathrm{Dom}(\pi)$ **then**
  $v \twoheadleftarrow \{0,1\}^\ell$
  if $v \in \mathrm{Ran}(\pi)$ **then**
    Coll $\leftarrow$ Coll $\cup \{u\}$, $\mathrm{Map}[u] \leftarrow v$
    $v \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Ran}(\pi)$
$\pi[u] \leftarrow v$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \notin \mathrm{Ran}(\pi)$ **then**
  $u \twoheadleftarrow \{0,1\}^\ell \backslash \mathrm{Dom}(\pi)$, $\pi[u] \leftarrow v$
return $\pi^{-1}[v]$

Figure 4.6.5: **Games for the proof of Theorem 4.4.3.** Each set is initialized to be $\emptyset$. Initially, procedure INITIALIZE() samples the challenge bit $c \twoheadleftarrow \{0,1\}$. Games G_1 and G_2 include the corresponding boxed statements.

We explain the game chain up until the terminal game. $\triangleright \mathrm{G}_0 \to \mathrm{G}_1$: We maintain two sets Coll and Seeds, which are initialized to be $\emptyset$. In procedure $\Pi(u)$, if $\pi[u]$ is not previously defined then we attempt to choose $\pi[u]$ uniformly, pretending that π is a random function, instead of a random permutation. Of course it may create inconsistency with prior points in $\mathrm{Dom}(\pi)$. If this happens, the "failure" point u is added to Coll, and we'll sample $\pi[u]$ again, according to the correct distribution. The set Seeds keeps track of the seeds K that we write to $\pi[K]$. The two games are identical until either game sets *bad*.

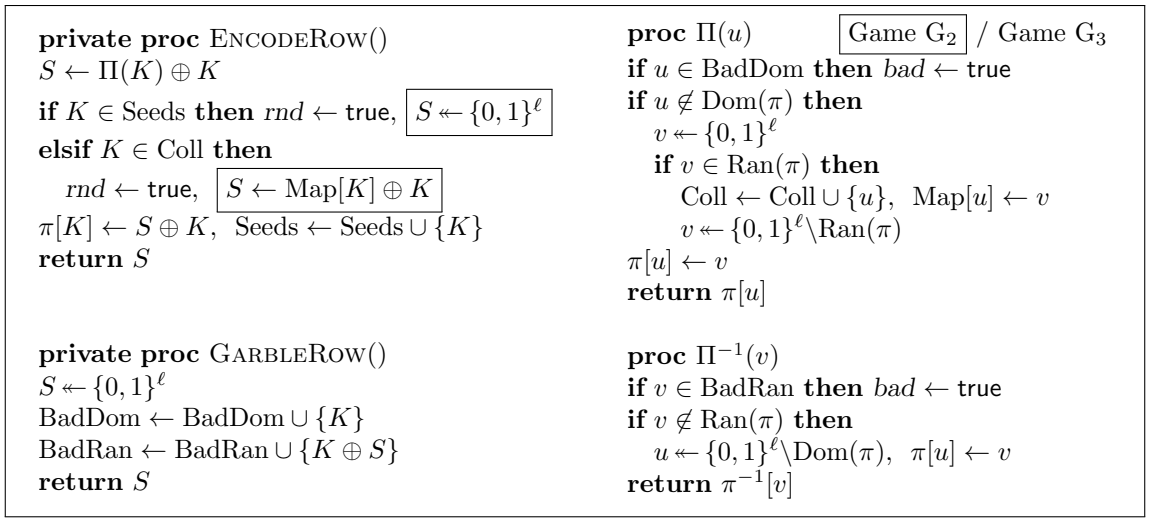

Figure 4.6.6: **Games for the proof of Theorem 4.4.3.** Each set is initialized to be $\emptyset$. Initially, procedure Initialize() samples the challenge bit $c \twoheadleftarrow \{0,1\}$. Games G_1 and G_2 include the corresponding boxed statements.

We claim that in game G_1, the visible token of the outgoing wire of each non-XOR gate is chosen uniformly, independent of R and other visible token created before. Such a visible token is either (i) $S[1:k] \oplus R$, with $S \leftarrow$ GarbleRow(), (ii) $S[1:k]$, with $S \leftarrow$ GarbleRow(), or (iii) $S[1:k]$, with $S \leftarrow$ EncodeRow(). Since GarbleRow always outputs a fresh $S \twoheadleftarrow \{0,1\}^\ell$, it suffices to show that the same holds for EncodeRow. Let K be a seed created in procedure EncodeRow. If K is equal to some prior seeds then game G_1 explicitly samples S uniformly. Otherwise, we let $S \leftarrow v \oplus K$, where v is the value sampled in $\Pi(K)$ at the *first* attempt. Since v is uniformly distributed over $\{0,1\}^\ell$, so is S.

We now bound the chance that G_1 sets *bad*. By using exactly the same arguments in the proof of Lemma 4.6.1, the chance that GarbleRow sets *bad* is at most $(3qQ + 7.5q^2)/2^\ell + (6rQ + 42rq)/2^k + \delta(12rQq + 30rq^2)/2^k$. What's left is to bound the chance that procedure EncodeRow triggers *bad* to true. Consider the ith call of EncodeRow. Let $K = \sigma(A, B, g)$ be the seed of this call, and let $K^* = \sigma(A^*, B^*, g^*)$ be an arbitrary point in Seeds then. By using exactly the same arguments of Lemma 4.6.2, the chance that $K = K^*$ is at most $2r/2^k$. Moreover, if $\delta = 0$ and $g \neq g^*$ then $\Pr[K = K^*] = 0$. In other

words, if $g \neq g^*$ then $\Pr[K = K^*] \leq 2r\delta/2^k$. By union bound,

$$\Pr[K \in \text{Seeds}] \leq 6r/2^k + 2|\text{Seeds}| \cdot \delta r/2^k \leq 6r/2^k + \delta r(8i-2),$$

because there are most three elements of Seeds using the tweak g. On the other hand, the chance that $K \in \text{Coll}$ is at most $|\text{Ran}(\pi)|/2^k \leq (Q+4i-1)/2^\ell$. Hence the chance that procedure EncodeRow triggers *bad* to true is at most

$$\sum_{i=1}^{q} \frac{Q+4i-1}{2^\ell} + \frac{6r}{2^k} + \frac{\delta r(8i-2)}{2^k} \leq \frac{Qq+2q^2+q}{2^\ell} + \frac{6rq}{2^k} + \frac{\delta(4rq^2+2rq)}{2^k} \quad .$$

$\triangleright \mathrm{G}_1 \to \mathrm{G}_2$: In procedure GarbleRow of game G_1, we write $S \oplus K$ to $\pi[K]$, but game G_2 drops this assignment. Since Seeds is used to keep track of seeds K that we write to $\pi[K]$, game G_2 doesn't modify Seeds in procedure Garble. In addition, we maintain two sets BadDom and BadRan that are initialized to the empty sets. Each call to GarbleRow will add K to BadDom and $S \oplus K$ to BadRan. The two games are identical until G_2 sets *bad*, that is, when $\mathcal{A}$ happens to query $\Pi(u)$ with $u \in \text{BadDom}$, or $\Pi^{-1}(v)$ with $v \in \text{BadRan}$.

We now bound the chance that G_2 sets *bad*. In this game, the visible token of the outgoing wire of each non-XOR gate is also chosen uniformly, independent of R and other visible token created before. By using exactly the same arguments of the proof of Lemma 4.6.3, the chance that G_2 sets *bad* is at most $(12rQ+12rq)/2^k + \delta(12rQq+12rq^2)/2^k$. (In the proof of Lemma 4.6.3, we let $S[1:k] = Y \oplus Z$, where Y is the value of the garbled row corresponding to K, and Z is the token that $S[1:k]$ masks. Here, if $\mathtt{a} = \mathtt{b} = 0$ then the row is blank, so let $Y = 0^k$ and $Z = S[1:k]$, which is also a token.)

$\triangleright \mathrm{G}_2 \to \mathrm{G}_3$: game G_3 drops the re-sampling of S in procedure EncodeRow, so S is always $\Pi(K) \oplus K$, and the assignment $\pi[K] \leftarrow S \oplus K$ is redundant, because it writes $\Pi(K)$ to $\pi[K]$. The two games are identical until G_2 sets *rnd*.

We now bound the chance that G_2 sets *rnd*. Consider the ith call of EncodeRow. Let $K = \sigma(A, B, g)$ be the seed of this call, and let $K^* = \sigma(A^*, B^*, g^*)$ be an arbitrary point in Seeds then. By using exactly the same arguments of Lemma 4.6.2, the chance that $K = K^*$ is at most $2r/2^k$. Moreover, if $\delta = 0$ and $g \neq g^*$ then $\Pr[K = K^*] = 0$. In other

proc GARBLE(f_0, f_1, x_0, x_1)
$(n, m, q', A', B', G') \leftarrow \Phi_{\text{xor}}(f_0)$
$v_{q+n-m+1} \cdots v_{q+n} \leftarrow \mathsf{ev}(f_0, x_0)$, $R \twoheadleftarrow \{0,1\}^{k-1}1$
for $i \leftarrow 1$ **to** $n+q$ **do** $V_i \twoheadleftarrow \{0,1\}^k$, $I_i \twoheadleftarrow V_i \oplus R$
for $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
  if $G'_g = \text{XOR}$ **then** $V_g \leftarrow V_a \oplus V_b$, $I_g \leftarrow V_g \oplus R$
  else
    $P[g,0,1] \twoheadleftarrow \{0,1\}^k$, $P[g,1,0] \twoheadleftarrow \{0,1\}^k$, $P[g,1,1] \twoheadleftarrow \{0,1\}^k$
    $\mathsf{a} \leftarrow \text{lsb}(V_a)$, $\mathsf{b} \leftarrow \text{lsb}(V_b)$
    $K \leftarrow \sigma(V_a, V_b, g)$, $Y \leftarrow (\Pi(K) \oplus K)[1\colon k]$
    if $\text{lsb}(V_a) = 0$ **and** $\text{lsb}(V_b) = 0$ **then** $V_g \leftarrow Y$, $I_g \twoheadleftarrow V_g \oplus R$
    else $V_g \twoheadleftarrow \{0,1\}^k$, $I_g \leftarrow V_g \oplus R$, $P[g, \mathsf{a}, \mathsf{b}] \leftarrow Y \oplus V_g$
for $i \leftarrow n+q-m+1$ **to** $n+q$ **do** $X_i^{v_i} \leftarrow V_i$, $X_i^{\overline{v}_i} \leftarrow I_i$
$F \leftarrow (n, m, q, A', B', P)$, $X \leftarrow (V_1, \ldots, V_n)$
$d \leftarrow \big(\text{lsb}(X^0_{n+q-m+1}), \ldots, \text{lsb}(X^0_{n+q})\big)$
return (F, X, d)

Figure 4.6.7: **Rewritten game G_3 of the proof of Theorem 4.4.3.** This game depends solely on $f' = \Phi_{\text{xor}}(f_0) = \Phi_{\text{xor}}(f_1)$ and the output $v = \mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. Procedures Π and Π^{-1} lazily implement a random permutation and its inverse, respectively.

words, if $g \neq g^*$ then $\Pr[K = K^*] \leq 2r\delta/2^k$. By union bound,

$$\Pr[K \in \text{Seeds}] \leq 2|\text{Seeds}|\delta r/2^k \leq \delta r(2i-2),$$

because there is *no* element of Seeds using the tweak g. On the other hand, the chance that $K \in \text{Coll}$ is at most $|\text{Ran}(\pi)|/2^k \leq (Q+i-1)/2^\ell$. Hence the chance that G_2 sets *rnd* is at most

$$\sum_{i=1}^{q} \frac{Q+i-1}{2^\ell} + \frac{\delta r(2i-2)}{2^k} \leq \frac{Qq + 0.5q^2 - 0.5q}{2^\ell} + \frac{\delta(rq^2 - rq)}{2^k} \ .$$

ANALYSIS OF GAME G_3. We claim that the output of game G_3 is independent of the challenge bit c. Hence $\Pr[\mathrm{G}_3^{\mathcal{A}}(k)] = 1/2$. To justify the claim above, from $f' = \Phi_{\text{xor}}(f_c)$ and the final output $v = \mathsf{ev}(f_c, x_c)$, which is independent of c, we can rewrite the code of procedure GARBLE of G_3 as shown in Fig. 4.6.7. There, we refer to the visible token of wire i as V_i, and its invisible counterpart as I_i, omitting the semantics of these tokens. Consider an arbitrary non-XOR gate g. Each ciphertext in the rows $P[g,0,1]$, $P[g,1,0]$, and $P[g,1,1]$ is chosen at random, unless the row can be opened by visible tokens. The visible token on wire g is chosen uniformly at random, unless both visible tokens of g's incoming wires end with 0. The invisible token on wire g is obtained by xoring R to the visible counterpart.

Summing up,

$$\begin{aligned}
& \mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{xor}}}_{\mathrm{GaXR}[\mathbb{E}]}(\mathcal{A},k) \\
= \;& 2(\Pr[\mathrm{G}_0^{\mathcal{A}}(k)] - \Pr[\mathrm{G}_3^{\mathcal{A}}(k)]) \\
\leq \;& \frac{10qQ+20q^2+q}{2^\ell} + \frac{36rQ+120rq}{2^k} + \frac{\delta(48rQq+94rq^2+2q)}{2^k} \\
\leq \;& \frac{10qQ+20q^2}{2^\ell} + \frac{36rQ+123rq}{2^k} + \frac{\delta(48rQq+94rq^2)}{2^k} \;\; .
\end{aligned}$$

4.7 Accounting for parameters in Fig. 4.4.1

For completeness, we give an analysis to justify the parameters used in Fig. 4.4.1.

SCHEME A1. The σ function is $\sigma(A,B,T) = A \oplus B \oplus T$. Since $\sigma(0^k,0^k,0^k) = \sigma(1^k,0^k,1^k)$, it follows that $\delta = 1$. Next, $\sigma(x,0^k,0^k) = \sigma(0^k,x,0^k) = x$. As $\Pr[x \twoheadleftarrow \{0,1\}^k : x = s] = 1/2^k$ for any string $s \in \{0,1\}^k$, the regularity is 1. On the other hand, since $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k,x,0^k) \oplus x = 0^k] = 1$, the strong regularity is the trivial 2^k.

SCHEME A3. The σ function is $\sigma(A,B,T) = (A \oplus B) \parallel T$, and thus $\delta = 0$. Next, $\sigma(x,0^k,0^{\ell-k}) = \sigma(0^k,x,0^{\ell-k}) = x0^{\ell-k}$. As $\Pr[x \twoheadleftarrow \{0,1\}^k : x0^{\ell-k} = s] \leq 1/2^k$ for any string $s \in \{0,1\}^\ell$, the regularity is 1. On the other hand, since $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k,x,0^{\ell-k}) \oplus x0^{\ell-k} = 0^\ell] = 1$, the strong regularity is the trivial 2^k.

SCHEME A2, WITH D1. The σ function is $\sigma(A,B,T) = 2A \oplus 4B \oplus T$. Since $\sigma(0^k,0^k,0^k) = \sigma(A,0^k,2A)$ for any $A \in \{0,1\}^k$, and there exists $A \in \{0,1\}^k$ such that $2A \neq 0^k$, it follows that $\delta = 1$. Let $*$ denote the multiplication operator in $\mathrm{GF}(2^k)$. Note that $f(x) = c * x$ is bijective, for any $c \in \mathrm{GF}(2^k)\backslash\{0\}$. Note that $\sigma(x,0^k,0^k) = 2*x$, $\sigma(0^k,x,0^k) = 4*x$, $\sigma(0^k,0^k,0^k) \oplus x = x$, $\sigma(x,0^k,0^k) \oplus x = 3*x$, $\sigma(0^k,x,0^k) \oplus x = 5*x$, $\sigma(x,x,0^k) \oplus x = 7*x$, and $\sigma(x,x,0^k) = 6*x$. Hence both the regularity and strong regularity are 1.

SCHEME A2, WITH D2/D3. Again, $\delta = 1$. We give an analysis for D2; the case of D3 is similar.

- Note that $\sigma(x, 0^k, 0^k) = x \ll 1$, and $\sigma(0^k, x, 0^k) = x \ll 2$, and $\sigma(0^k, 0^k, 0^k) \oplus x = x$. Hence $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) = s] \leq 2/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) = s] \leq 4/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, 0^k, 0^k) \oplus x = s] = 1/2^k$, for any $s \in \{0,1\}^k$.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) \oplus x = s] = 1/2^k$ for any $s \in \{0,1\}^k$. Let $f_0(x) = (x \ll 1) \oplus x$. To justify this claim, let $x = x_1 \cdots x_k$. Then

  $$f_0(x) = (x_1 \oplus x_2) \| \cdots \| (x_{k-1} \oplus x_k) \| x_k$$

  is bijective. Indeed, given $y = y_1 \cdots y_k$, we can compute $x = x_1 \cdots x_k = f_0^{-1}(y)$ by way of $x_k = y_k$, and recursively, $x_i = x_{i+1} \oplus y_i$, for $i = k-1, k-2, \ldots, 1$. As $x \twoheadleftarrow \{0,1\}^k$, it follows that $\sigma(x, 0^k, 0^k) \oplus x = f_0(x)$ is also uniformly distributed over $\{0,1\}^k$, and the claim follows.
- Note that $\sigma(x, x, 0^k) = (x \ll 1) \oplus (x \ll 2) = f_0(x) \ll 1$. Since $f_0(x)$ is a permutation on $\{0,1\}^k$, it follows that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) = s] \leq 2/2^k$ for any $s \in \{0,1\}^k$.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) \oplus x = s] = 1/2^k$ for any $s \in \{0,1\}^k$. Let $f_1(x) = (x \ll 2) \oplus x$. To justify this, let $x = x_1 \cdots x_k$. Then

  $$f_1(x) = (x_1 \oplus x_3) \| \cdots \| (x_{k-2} \oplus x_k) \| x_{k-1} \| x_k$$

  is bijective. Given $y = y_1 \cdots y_k$, we can compute $x = x_1 \cdots x_k = f_1^{-1}(y)$ by way of $x_k = y_k$, $x_{k-1} = y_{k-1}$, and recursively, $x_i = x_{i+2} \oplus y_i$, for $i = k-2, k-3, \ldots, 1$. As $x \twoheadleftarrow \{0,1\}^k$, it follows that $\sigma(0^k, x, 0^k) \oplus x = f_1(x)$ is uniformly distributed over $\{0,1\}^k$, and the claim follows.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) \oplus x = s] = 1/2^k$ for any $s \in \{0,1\}^k$. Consider $f_2(x) = (x \ll 2) \oplus (x \ll 1) \oplus x$. To justify this claim, let $x = x_1 \cdots x_k$. Then

  $$f_2(x) = (x_1 \oplus x_2 \oplus x_3) \| \cdots \| (x_{k-2} \oplus x_{k-1} \oplus x_k) \| (x_{k-1} \oplus x_k) \| x_k$$

  is bijective. Indeed, given $y = y_1 \cdots y_k$, we can compute $x = x_1 \cdots x_k = f_2^{-1}(y)$ by way of $x_k = y_k$, $x_{k-1} = y_{k-1} \oplus x_k$, and recursively, $x_i = x_{i+1} \oplus x_{i+2} \oplus y_i$, for $i = k-2, k-3, \ldots, 1$. As $x \twoheadleftarrow \{0,1\}^k$, it follows that $\sigma(x, x, 0^k) \oplus x = f_2(x)$ is also uniformly distributed over $\{0,1\}^k$, and the claim follows.

Hence both the regularity and strong regularity are at most 4. On the other hand, note that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) = 0^k] = 4/2^k$, and thus both the regularity and strong regularity are exactly 4.

SCHEME A2, WITH D4/D5. Again, $\delta = 1$. We give an analysis for D4; the case of D5 is similar.

- Note that $\sigma(x, 0^k, 0^k) = x \lll 1$, and $\sigma(0^k, x, 0^k) = x \lll 2$, and $\sigma(0^k, 0^k, 0^k) \oplus x = x$. Hence $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) = s] = 1/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) = s] = 1/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, 0^k, 0^k) \oplus x = s] = 1/2^k$, for any $s \in \{0,1\}^k$.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) \oplus x = s] \leq 2/2^k$ for any $s \in \{0,1\}^k$. Let $g_0(x) = (x \lll 1) \oplus x$. To justify this claim, let $x = x_1 \cdots x_k$. Then

$$g_0(x) = (x_1 \oplus x_2) \| \cdots \| (x_{k-1} \oplus x_k) \| (x_k \oplus x_1) \ .$$

  Given $y = y_1 \cdots y_k$, there are at most two pre-images $x = x_1 \cdots x_k$, since $x_i = x_{i-1} \oplus y_{i-1}$, for $i \in \{2, \ldots, k\}$. Hence for any $s \in \{0,1\}^k$, there are at most two values x such that $\sigma(x, 0^k, 0^k) = g_0(x) = s$, and the claim follows.
- Note that $\sigma(x, x, 0^k) = (x \lll 1) \oplus (x \lll 2) = g_0(x) \lll 1$. Then $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) = s] \leq 2/2^k$ for any $s \in \{0,1\}^k$.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) \oplus x = s] \leq 4/2^k$ for any $s \in \{0,1\}^k$. Let $g_1(x) = (x \lll 2) \oplus x$. To justify this claim, let $x = x_1 \cdots x_k$. Then

$$g_1(x) = (x_1 \oplus x_3) \| \cdots \| (x_{k-2} \oplus x_k) \| (x_{k-1} \oplus x_1) \| (x_k \oplus x_2) \ .$$

  Given $y = y_1 \cdots y_k$, there are at most 4 pre-images $x = x_1 \cdots x_k$, since $x_i = x_{i-2} \oplus y_{i-2}$ for any $i \in \{3, \ldots, k\}$. Hence for any $s \in \{0,1\}^k$, there are at most four values x such that $\sigma(x, 0^k, 0^k) = g_1(x) = s$, and the claim follows.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) \oplus x = s] \leq 4/2^k$ for any $s \in \{0,1\}^k$. Consider $g_2(x) = (x \lll 2) \oplus (x \lll 1) \oplus x$. Let $x = x_1 \cdots x_k$. Then

$$g_2(x) = (x_1 \oplus x_2 \oplus x_3) \| \cdots \| (x_{k-2} \oplus x_{k-1} \oplus x_k) \| (x_{k-1} \oplus x_k \oplus x_1) \| (x_k \oplus x_1 \oplus x_2).$$

  Given $y = y_1 \cdots y_k$, there are at most four pre-images $x = x_1 \cdots x_k$, as $x_i = y_{i-2} \oplus$

$x_{i-1} \oplus x_{i-2}$, for any $i \in \{3, \ldots, k\}$. So for any $s \in \{0,1\}^k$, there are at most four values x such that $\sigma(x, 0^k, 0^k) = g_2(x) = s$, and the claim follows.

Hence the regularity is exactly 1 and strong regularity is at most 4.

SCHEME A2, WITH D6/D7. Again, $\delta = 1$. We give an analysis for D6; the case of D7 is similar. Let $x = x_1 \cdots x_k$ and $n = \lfloor k/2 \rfloor$.

- Note that $\sigma(x, 0^k, 0^k) = x_2 \cdots x_n 0 \parallel x_{n+2} \cdots x_k 0$, and $\sigma(0^k, 0^k, 0^k) \oplus x = x$, and $\sigma(0^k, x, 0^k) = x_3 \cdots x_n 00 \parallel x_{n+3} \cdots x_k 00$. Hence $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) = s] \leq 4/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) = s] \leq 16/2^k$, and $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, 0^k, 0^k) \oplus x = s] = 1/2^k$, for any $s \in \{0,1\}^k$.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, 0^k, 0^k) \oplus x = s] \leq 1/2^k$ for any $s \in \{0,1\}^k$. Let

  $$h_0(x) = (x_1 \oplus x_2) \| \cdots \| (x_{n-1} \oplus x_n) \| x_n \parallel (x_{n+1} \oplus x_{n+2}) \| \cdots \| (x_{k-1} \oplus x_k) \| x_k \ .$$

  Given $y = y_1 \cdots y_k \in S$, there is at most one pre-image $x = x_1 \cdots x_k$, as $x_i = y_i$ if $i \in \{n, k\}$, and $x_i = x_{i+1} \oplus y_i$ otherwise. So for any $s \in \{0,1\}^k$, there is at most one value x such that $\sigma(x, 0^k, 0^k) = h_0(x) = s$, and the claim follows.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(0^k, x, 0^k) \oplus x = s] \leq 1/2^k$ for any $s \in \{0,1\}^k$. Let

  $$\begin{aligned} h_1(x) = \quad & (x_1 \oplus x_3) \| \cdots \| (x_{n-2} \oplus x_n) \| x_{n-1} \| x_n \\ \parallel \quad & (x_{n+1} \oplus x_{n+3}) \| \cdots \| (x_{k-2} \oplus x_k) \| x_{k-1} \| x_k \ . \end{aligned}$$

  Given $y = y_1 \cdots y_k$, there is at most one pre-image $x = x_1 \cdots x_k$, as $x_i = y_i$ if $i \in \{n-1, n, k-1, k\}$ and $x_i = x_{i+2} + y_i$ otherwise. So for any $s \in \{0,1\}^k$, there is at most one value x such that $\sigma(x, 0^k, 0^k) = h_1(x) = s$, and the claim follows.
- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) \oplus x = s] \leq 1/2^k$ for any $s \in \{0,1\}^k$. Let

  $$\begin{aligned} h_2(x) = \quad & (x_1 \oplus x_2 \oplus x_3) \| \cdots \| (x_{n-2} \oplus x_{n-1} \oplus x_n) \| (x_{n-1} \oplus x_n) \| x_n \\ \parallel \quad & (x_{n+1} \oplus x_{n+2} \oplus x_{n+3}) \| \cdots \| (x_{k-2} \oplus x_{k-1} \oplus x_k) \| (x_{k-1} \oplus x_k) \| x_k \ . \end{aligned}$$

Given $y = y_1 \cdots y_k$, there is at most one pre-image $x = x_1 \cdots x_k$, since $x_i = y_i$ if $i \in \{n, k\}$, $x_i = y_i \oplus x_{i+1}$ if $i \in \{n-1, k-1\}$, and $x_i = y_i \oplus x_{i+1} \oplus x_{i+2}$ otherwise. Hence for any $s \in \{0,1\}^k$, there is at most one value x such that $\sigma(x, 0^k, 0^k) = h_2(x) = s$, and the claim follows.

- We claim that $\Pr[x \twoheadleftarrow \{0,1\}^k : \sigma(x, x, 0^k) = s] \leq 4/2^k$ for any $s \in \{0,1\}^k$. Let

$$h_3(x) = (x_2 \oplus x_3) \| \cdots \| (x_{n-1} \oplus x_n) \| x_n 0 \parallel (x_{n+2} \oplus x_{n+3}) \| \cdots \| (x_{k-1} \oplus x_k) \| x_k 0 \ .$$

  Given $y = y_1 \cdots y_k$, there are at most four pre-images $x = x_1 \cdots x_k$, since $x_i = y_{i-1}$ if $i \in \{n, k\}$, $x_i = y_{i-1} \oplus x_{i+1}$ if $i \in \{2, \ldots, n-1, n+1, \ldots, k-1\}$. Hence for any $s \in \{0,1\}^k$, there are at most four values x such that $\sigma(x, 0^k, 0^k) = h_2(x) = s$, and the claim follows.

Hence both the regularity and strong regularity are at most 16.

SCHEME A4. The σ function is $\sigma(A, B, T) = (2A \oplus 4B) \parallel T$, and thus $\delta = 0$. The analysis for the regularity and strong regularity is similar to that of scheme A2.

Chapter 5

Adaptively Secure Garbling

5.1 Introduction

Overview. The notions we have formalized so far, while still suitable for a variety of applications, provide only *static* security, meaning the input x is not allowed to depend on the garbled circuit F. But some applications, notably one-time programs [45] and secure outsourcing [37], require *adaptive* security, where x may depend on F. In such cases Yao's technique can be enhanced in *ad hoc* ways, the resulting protocol then incorporated into the higher-level application.

In this chapter, we provide a different approach, by investigating the adaptive security of garbling schemes. We show how adaptively secure garbling schemes support simple solutions for one-time programs and secure outsourcing, with privacy being the goal in the first case and obliviousness and authenticity the goal in the second. Let's look at two of the applications that motivate our work.

Two applications. *One-time programs* are due to Goldwasser, Kalai, and Rothblum (GKR) [45]. The authors aim to compile a program into one that can be executed just once, on an input of the user's choice. Unachievable in any "standard" model of computation, GKR assume a model providing what they call *one-time memory*. Their solution makes crucial use

of Yao's garbled-circuit technique. Recognizing that this does not support adaptive queries, GKR embellish the method by a technique involving output-masking and n-out-of-n secret sharing.

In a different direction, *secure outsourcing* was formalized and investigated by Gennaro, Gentry, and Parno (GGP) [37]. Here a *client* transforms a function f into a function F that is handed to a *worker*. When, later, the client would like to evaluate f at x (and various such inputs may arise), he should be able to quickly map x to a garbled input X and give this to the worker, who will compute and return $Y = F(X)$. The client must be able to quickly reconstruct from this $y = f(x)$. He should be sure that the correct value was computed—the computation is *verifiable*—while the server shouldn't learn anything significant about x, including $f(x)$.[1] GGP again make use of circuit garbling, and they again realize that they need something from it—its authenticity—that is a *novum* for this domain.

Issues. Assuming the existence of a one-way function, GKR [45] claim that their construction turns a (statically-secure) garbled circuit into a secure one-time program. We point to a gap in their proof, namely, the absence of a reduction showing that their simulator works based on the one-way function assumption. By presenting an example of a statically-secure garbled circuit that, under their transform, yields a program that is not one-time, we also show that the gap cannot be filled without changing either the construction or the assumption. The problem is that the GKR transform fails to ensure adaptive security of garbled circuits under the stated assumption.

Lindell and Pinkas (LP) [70] prove static security of a version of Yao's protocol assuming a semantically secure encryption scheme satisfying some extra properties (an elusive and efficiently verifiable range). GGP [37] build a one-time outsourcing scheme from the LP protocol, claiming to prove its security based on the same assumption as used in LP. Again, there is a gap in this proof arising from an implicit assumption of adaptive security of the LP construction.

[1] This is *input privacy*. One could go further and ask that the server not learn anything it shouldn't about f itself. Our definitions and constructions will encompass this stronger goal.

We do not believe these are major problems for either work. In both cases, alternative ways to establish the the authors' main results already existed. Goyal, Ishai, Sahai, Venkatesan and Wadia [46] present an unconditional one-time compiler (no complexity-theoretic assumption is used at all), while Chung, Kalai and Vadhan [30] present secure outsourcing schemes based solely on FHE (garbled circuits are not employed). Our interpretation of the stated gaps is that they are symptoms of something else—a missing abstraction boundary. The applications we point to motivate the study of adaptive security for garbling schemes, while the gaps indicate that the issues may be more subtle than recognized.

Definitions. We now discuss our contributions in more depth. We extend the our security notions for garbling schemes to adaptive ones, considering two flavors of adaptive security. With *coarse-grained* adaptive security the input x can depend on the garbled function F but x itself is atomic, provided all at once. With *fine-grained* adaptive security not only may x depend on the garbled function F, but individual bits of x can depend on the "tokens" the adversary has so-far learned. We will see that coarse-grained adaptive security is what's needed for GGP's approach to secure outsourcing, while fine-grained adaptive security is what's needed for GKR's approach to one-time programs.

Orthogonal to adaptive security's granularity are the security aims themselves. We again consider three different notions: privacy, obliviousness, and authenticity. This gives rise to six new security notions: $\{\text{prv}, \text{obv}, \text{aut}\} \times \{\text{coarse}, \text{fine}\}$. We compactly denote these prv1, prv2, obv1, obv2, aut1, aut2. Our primary definitions for adaptive secrecy (prv1, prv2, obv1, obv2) are simulation-based. In Section 5.4 we give indistinguishability-based counterparts as well. For simplicity, we write prv for prv.sim and obv for obv.sim.

Relations. We explore the provable-security relationships among our definitions. As expected, the simulation-based definitions imply indistinguishability-based ones (namely, prv1 $\Rightarrow$ prv1.ind, prv2 $\Rightarrow$ prv2.ind, obv1 $\Rightarrow$ obv1.ind, and obv2 $\Rightarrow$ obv2.ind). But none of the converse statements hold. For the static setting, the converse statements *do* hold as long as the associated side-information function is efficiently invertible. In contrast, we show that,

for adaptive privacy, this condition still won't guarantee equivalence of simulation-based and indistinguishability-based notions. (For obliviousness, it is true that obv1.ind $\Rightarrow$ obv1 and obv2.ind $\Rightarrow$ obv2 if Φ is efficiently invertible.) The results are our main reason to focus on simulation-based definitions for adaptive privacy. Section 5.5 paints a complete picture of the relations among our basic definitions. Apart from the trivial relations (prv2 $\Rightarrow$ prv1 $\Rightarrow$ prv, obv2 $\Rightarrow$ obv1 $\Rightarrow$ obv, and aut2 $\Rightarrow$ aut1 $\Rightarrow$ aut) nothing implies anything else.

Achieving adaptive security. Basic garbling-scheme constructions [39, 42, 78] either do not achieve adaptive security or present difficulties in proving adaptive security that we do not know how to overcome. One could give new constructions and directly prove them xxx1 or xxx2 secure, for xxx $\in \{\text{prv}, \text{obv}, \text{aut}\}$. An alternative is to provide generic ways to transform statically secure garbling schemes to adaptively secure ones. Combined with the results in Section 3.6, this would yield adaptively-secure garbling schemes.

The aim of the GKR construction was to add adaptive security to statically-secure garbled circuit constructions. We reformulate it as a transform, OMSS (Output Masking and Secret Sharing), aiming to turn a prv secure garbling scheme to a prv2 secure one. We show, by counterexample, that OMSS does not achieve this goal.

To give transforms that work we make two steps, first passing from static security to coarse-grained adaptive security, and thence to fine-grained adaptive security. We design these transformations first for privacy (prv-to-prv1, prv1-to-prv2) and then for simultaneously achieving all three goals (all-to-all1 and all1-to-all2). Our prv-to-prv1 transform uses a one-time-padding technique from [46], while our prv1-to-prv2 transform uses the secret-sharing component of OMSS.

Applications. We treat the two applications above, one-time programs and secure outsourcing. We show that adaptive garbling schemes yield these applications easily and directly. Specifically, we show that a prv2 projective garbling scheme can be turned into a secure one-time program by simply putting the garbled inputs into the one-time memory. We also show how to easily turn an obv1+aut1 secure garbling scheme into a secure one-time outsourc-

ing scheme. (GGP [37] show how to lift one-time outsourcing schemes to many-time ones using FHE.) The simplicity of these transformations underscores our tenet that abstracting garbling schemes and treating adaptive security for them enables modular and rigorous applications of the garbled-circuit technique. Basing the applications on garbling schemes also allows instantiations to inherit efficiency features of future schemes.

Applying our prv-to-prv1 and then prv1-to-prv2 transforms to scheme Garble1 in Section 3.5 yields a prv2-secure scheme based on any one-way function. Combining this with the above yields one-time programs based on one-way functions, recovering the claim of GKR [45]. Similarly, applying our all-to-all1 transform to scheme Garble2 in Section 3.6 yields an obv1+aut1 secure garbling scheme based on a one-way function, and combining this with the above yields a secure one-time outsourcing scheme based on one-way functions.

Efficiency. Let us say a garbling scheme has *short* garbled inputs if their length depends only on the security parameter k, the length n of f's input, and the length m of f's output. It does not depend on the length of f. The schemes Garble1 and Garble2, as with all classical garbled-circuit constructions, have short garbled inputs. But our prv-to-prv1 and all-to-all1 transforms result in long garbled inputs. In the ROM (random-oracle model) we are able to provide schemes producing short garbled inputs, as illustrated in Fig. 5.1.1. Constructing an adaptively secure garbling scheme with short garbled inputs under standard assumptions remains open.[2]

Short garbled inputs are particularly important for the application to secure outsourcing, for in their absence the outsourcing scheme may fail to be non-trivial. (Non-trivial means that the client effort is less than the effort needed to directly compute the function [37].) In particular, the one-time outsourcing scheme we noted above, derived by applying all-to-all1 to Garble2, fails to be non-trivial. ROM schemes do not fill the gap because of the use of FHE in upgrading one-time schemes to many-time ones [37]. Thus, a secure and non-

[2]Intuitively, the underlying encryption appears to need some kind of security against selective-opening attacks that reveal decryption keys (SOA-K), and this is hard without long keys [12]. However, there is some hope because full-fledged SOA-K security does not seem to be needed.

Transform	Model	Cost	See
prv-to-prv1	standard model	$\|F\| + \|d\| + \|X\|$	Theorem 5.2.2
prv1-to-prv2	standard model	$(n+1)\,\|X\|$	Theorem 5.2.3
all-to-all1	standard model	$\|F\|+\|d\|+\|X\|+k$	Theorem 5.3.1
all1-to-all2	standard model	$(n+1)\,\|X\|$	Theorem 5.3.2
rom-prv-to-prv1	random-oracle model	$\|X\| + k$	Theorem 5.2.4
rom-prv1-to-prv2	random-oracle model	$\|X\| + nk$	Theorem 5.2.5
rom-all-to-all1	random-oracle model	$\|X\| + 2k$	Theorem 5.3.3
rom-all1-to-all2	random-oracle model	$\|X\| + nk$	Theorem 5.3.4

Figure 5.1.1: **Achieving adaptive security**. The name of each transform specifies its relevant property. The word all means that prv, obv, and aut are all upgraded. Column "Cost" specifies the length of the garbled input in the constructed scheme in terms of the lengths of the input scheme's garbled function F, decoding function d, garbled input X, the number of input bits n, and security parameter k.

trivial instantiation of the GGP method is still lacking. (However, as we have noted before, non-trivial secure outsourcing may be achieved by entirely different means [30].)

Further related work. Applebaum, Ishai, and Kushilevitz [6] investigate ideas similar to obliviousness and authenticity. Their approach to obtaining these ends from privacy can be lifted and formalized in our settings; one could specify transforms prv1-to-all1 and prv2-to-all2, effectively handling the constructive story "horizontally" instead of "vertically." The line of work on *randomized encodings* that the same authors have been at the center of provides an alternative to garbling schemes [54] but lacks the granularity to speak of adaptive security.

Concurrent work by Kamara and Wei (KW) investigates the garbling what they call *structured circuits* [60] and, in the process, give definitions somewhat resembling prv1, obv1, and aut1, although circuit-based, not function-hiding, and not allowing the adversary to specify the initial function. KW likewise draw motivation from GKR and GGP, indicating that, in these two settings, the adversary can choose the inputs to the computation as a function of the garbled circuit, motivating adaptive notions of privacy and unforgeability.

proc GARBLE(f)	**proc** INPUT(x) $\quad \mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$	**if** $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
if $b = 1$ **then** $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$	**if** $b = 1$ **then** $X \leftarrow \mathsf{En}(e, x)$
else $(F, d) \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$	**else** $y \leftarrow \mathsf{ev}(f, x)$, $X \leftarrow \mathcal{S}(y, 1)$
return (F, d)	**return** X

proc GARBLE(f)	**proc** INPUT(i, c) $\quad \mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$; $n \leftarrow f.n$; $Q \leftarrow \emptyset$; $\tau \leftarrow \varepsilon$	**if** $i \notin \{1, \ldots, n\} \setminus Q$ **then return** $\perp$
if $b = 1$ **then**	$x_i \leftarrow c$; $Q \leftarrow Q \cup \{i\}$
$(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$	**if** $\lvert Q \rvert = n$ **then**
else	$x \leftarrow x_1 \cdots x_n$; $y \leftarrow \mathsf{ev}(f, x)$; $\tau \leftarrow y$
$(F, d) \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$	**if** $b = 1$ **then** $X_i \leftarrow X_i^{x_i}$ **else** $X_i \leftarrow \mathcal{S}(\tau, i, \lvert Q \rvert)$
return (F, d)	**return** X_i

Figure 5.2.1: **Adaptive privacy: prv1 and prv2.** Games to define the coarse-grained and fine-grained privacy of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Each game starts with INITIALIZE() that samples a bit $b \twoheadleftarrow \{0,1\}$, and its FINALIZE($b'$) returns the predicate $(b = b')$.

5.2 Adaptive privacy and one-time programs

In this section we define coarse and fine-grained adaptive privacy for garbling schemes. We show that some natural approaches to achieve these aims fail. We provide alternatives that work, and more efficient ones in the ROM. We apply this to get secure one-time programs.

5.2.1 Definitions for adaptive privacy

In the privacy notion from Section 3.2, the adversary is *static*, in the sense it must commit to its initial function f and its input x at the same time. Thus the latter is independent of the garbled function F (and the decoding function d) derived from f. It is natural to consider stronger privacy notions, ones where the adversary obtains F and *then* selects x. Two formulations for this are specified in Fig. 5.2.1. We call these *adaptive* security. The notion in the top panel, denoted by prv1, is *coarse-grained* adaptive security. The notion in the bottom panel denoted by prv2, is *fine-grained* adaptive security. This notion is only applicable for projective garbling schemes.

In detail, let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme and let Φ be a side-information function. We define two simulation-based notions of privacy via the games $\mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$ and $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$ of Fig. 5.2.1. Here $\mathcal{S}$, the *simulator*, is an always-terminating algorithm that

maintains state across invocations. An adversary $\mathcal{A}$ interacting with any of these games must make exactly one GARBLE query. For game Prv1 it is followed by a single INPUT query. For game Prv2 it is followed by multiple INPUT queries. There, the garbling scheme must be projective. The advantage the adversary gets is defined by

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A},k) &= 2\Pr[\mathrm{Prv1}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k)]-1\\ \mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A},k) &= 2\Pr[\mathrm{Prv2}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k)]-1\ .\end{aligned}$$

For xxx $\in \{\mathrm{prv1},\mathrm{prv2}\}$ we say that $\mathcal{G}$ is xxx secure with respect to (or over) Φ if for every PT adversary $\mathcal{A}$ there exists a PT simulator $\mathcal{S}$ such that $\mathbf{Adv}^{\mathrm{xxx},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A},\cdot)$ is negligible. We let $\mathsf{GS}(\mathrm{xxx},\Phi)$ be the set of all garbling schemes that are xxx secure over Φ.

Let us now explain the two games. For coarse-grained adaptive privacy, we begin by letting the adversary pick f. Either we garble it to $(F,e,d) \leftarrow \mathsf{Gb}(1^k,f)$ and give the adversary (F,d); or else we ask the simulator to devise a fake (F,d) based solely on k and the partial information $\Phi(f)$ about f. Only after the adversary has received (F,d) do we ask it to provide an input x. Corresponding to the two choices we either encode x to $X = \mathsf{En}(e,x)$ or ask the simulator to produce a fake X, assisting it only by providing $\mathsf{ev}(f,x)$. The adversary has to guess if the garbling was real or fake.

Coarse-grained adaptive privacy is arguably not all *that* adaptive, as the adversary specifies its input x all in one shot. This is unavoidable as long as the encoding function e operates on x atomically, using (all of) x to generate (all of) X. But if the encoding function e is projective, then we can dole out the garbled input component-by-component. In a garbling scheme that enjoys fine-grained adaptive privacy, the adversary may, for example, specify the second bit x_2 of the input x, receive the corresponding token $X_2^{x_2}$, then specify the first bit x_1 of x, and so on. Only after the adversary specifies all n bits, one by one, is the input fully determined. At that point the simulator is handed y, which might be needed for constructing the final token $X_i^{x_i}$.

5.2.2 The OMSS transform

In the process of constructing one-time programs from garbled circuits, GKR [45] recognize the need for adaptive privacy of the garbled circuits. Their construction incorporates a technique to provide it. This technique is easily abstracted to provide, in our terminology, a transform that aims to convert a projective, prv garbling scheme into a projective, prv2 garbling scheme. Instead of garbling f we pick $r \twoheadleftarrow \{0,1\}^m$ and garble the circuit g defined by $g(x) = f(x) \oplus r$ for every $x \in \{0,1\}^n$ where $n = f.n$ and $m = f.m$. Then we secret share r as $r = r_1 \oplus \cdots \oplus r_n$ and include r_i in the i-th token, so that evaluation reconstructs r and it can be xored back at decoding time to recover $\mathsf{ev}(f, x)$ as $\mathsf{ev}(g, x) \oplus r$. Intuitively, this should work because the simulator can garble a dummy constant function with random output s and does not have to commit to r until it gets the target output value y of f and needs to provide the last token, at which point it can pick $r = s \oplus y$ so that the final output is y as desired [45]. Just the same, we show by counterexample that the OMSS does not work, in general, to convert a prv secure scheme to a prv2 secure one: we present a prv secure $\mathcal{G}$ such that $\mathsf{OMSS}[\mathcal{G}]$ is not prv2 secure.[3]

Now proceeding formally, we associate to circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{proj})$ the circuit-garbling scheme $\mathsf{OMSS}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_2, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}(\mathrm{proj})$ defined at the top of Fig. 5.2.2. For simplicity we are assuming that the decoding rule d in $\mathcal{G}$ is always vacuous, meaning $d = \varepsilon$. (Recall that we do not need non-trivial d to achieve privacy, and this lets us stay closer to GKR [45], whose garbled circuits have no analogue of our decoding rule.) In the code, $g(\cdot) \leftarrow f(\cdot) \oplus r$ means that we construct from f, r a circuit g such that $\mathsf{ev}(g, x) = \mathsf{ev}(f, x) \oplus r$ for all $x \in \{0,1\}^{f.n}$. (Note we can do this in such a way that $\Phi_{\mathrm{topo}}(g) = \Phi_{\mathrm{topo}}(f)$.)

The claim under consideration is that if $\mathcal{G}$ is prv secure relative to $\Phi = \Phi_{\mathrm{topo}}$ then $\mathcal{G}_2$ is prv2 secure relative to $\Phi = \Phi_{\mathrm{topo}}$. To prove this, we would need to let $\mathcal{A}_2$ be an arbitrary

[3] In Section 5.2.7 we extend this to show that the OMSS-based one-time compiler of GKR [45] is not secure. The underlying technical issues, are, however in our view easier understood in terms of garbling, divorced from the application to one-time programs.

proc $\mathsf{Gb}_2(1^k, f)$
$n \leftarrow f.n$, $r_1, \ldots, r_n \twoheadleftarrow \{0,1\}^{f.m}$
$r \leftarrow r_1 \oplus \cdots \oplus r_n$, $g(\cdot) \leftarrow f(\cdot) \oplus r$
$(G, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), \varepsilon) \twoheadleftarrow \mathsf{Gb}(1^k, g)$
for $i \in \{1, \ldots, n\}$ **do**
  $T_i^0 \leftarrow (X_i^0, r_i)$, $T_i^1 \leftarrow (X_i^1, r_i)$
return $(G, (T_1^0, T_1^1, \ldots, T_n^0, T_n^1), \varepsilon)$

proc $\mathsf{En}_2((T_1^0, T_1^1, \ldots, T_n^0, T_n^1), x)$
$x_1 \cdots x_n \leftarrow x$
return $(T_1^{x_1}, \ldots, T_n^{x_n})$

proc $\mathsf{Ev}_2(G, (T_1, \ldots, T_n))$
for $i \in \{1, \ldots, n\}$ **do** $(X_i, r_i) \leftarrow T_i$
$Y \leftarrow \mathsf{Ev}(G, (X_1, \ldots, X_n))$
$r \leftarrow r_1 \oplus \cdots \oplus r_n$
return (Y, r)

proc $\mathsf{De}_2(\varepsilon, (Y, r))$
return $\mathsf{De}(\varepsilon, Y) \oplus r$

proc $\mathsf{Gb}(1^k, g)$
$(n, m) \leftarrow (g.n, g.m)$
$(G', (Z_1^0, Z_1^1, \ldots, Z_n^0, Z_n^1), \varepsilon) \twoheadleftarrow \mathsf{Gb}'(1^k, g)$
for $i \in \{1, \ldots, n\}$ **do** $V_i^0, V_i^1 \twoheadleftarrow \{0,1\}^m$
$v_1 \cdots v_n \leftarrow v \twoheadleftarrow \{0,1\}^n$, $V \twoheadleftarrow \{0,1\}^m$
if $n \geq k$ **then**
  $V \leftarrow \mathsf{ev}(g, \overline{v}) \oplus V_1^{v_1} \oplus \cdots \oplus V_n^{v_n}$
for $i \in \{1, \ldots, n\}$ **do**
  $X_i^0 \leftarrow (Z_i^0, V_i^0)$, $X_i^1 \leftarrow (Z_i^1, V_i^1)$
$G \leftarrow (G', v, V)$
return $(G, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), \varepsilon)$

proc $\mathsf{Ev}(G, (X_1, \ldots, X_n))$
for $i \in \{1, \ldots, n\}$ **do** $(Z_i, V_i) \leftarrow X_i$
$(G', v, V) \leftarrow G$
return $\mathsf{Ev}'(G', (Z_1, \ldots, Z_n))$

proc $\mathsf{En}((X_1^0, X_1^1, \ldots, X_n^0, X_n^1), x)$
$x_1 \cdots x_n \leftarrow x$
return $(X_1^{x_1}, \ldots, X_n^{x_n})$

Figure 5.2.2: **OMSS definition (top).** Scheme $\mathsf{OMSS}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_2, \mathsf{Ev}_2, \mathsf{ev})$ where $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. **OMSS counterexample (bottom).** The garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ obtained from $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ is prv secure when $\mathcal{G}'$ is, but $\mathsf{OMSS}[\mathcal{G}]$ is not prv2 secure. We assume the decoding rule of $\mathcal{G}'$ is vacuous, a feature inherited by $\mathcal{G}$. We are letting $\overline{v}$ denote the bitwise complement of a string v.

PT adversary and build a PT simulator $\mathcal{S}_2$ such that $\mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2, \cdot)$ is negligible. GKR suggest a plausible strategy for the simulator that, in particular, explains the intuition for the transform. We present here our understanding of this strategy adapted to our setting. In its first phase the simulator $\mathcal{S}_2$ has input $1^k, \phi, 0$ where $\phi = \Phi(f)$, with f being the query made by the adversary to Garble. Simulator $\mathcal{S}_2$ picks $s \twoheadleftarrow \{0,1\}^n$ and lets f_s be the circuit that has output s on all inputs and $\Phi_{\mathrm{topo}}(f_s) = \phi$. It also picks random m-bit strings $s_1, \ldots, s_n$ and a random input $w \twoheadleftarrow \{0,1\}^n$. It lets $(G, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), \varepsilon) \twoheadleftarrow \mathsf{Gb}(1^k, f_s)$ and returns G to the adversary. In the second phase, when given input τ, i, j, for $j \leq n-1$, the simulator lets $T_i \leftarrow (X_i^{w_i}, s_i)$ and returns T_i to the adversary as the token for bit i of the input. In the case that $j = n$, the simulator obtains (from τ as per our game) the output

$y = \mathsf{ev}(f, x)$ of the function on input x, the latter defined by the adversary's queries to INPUT. It now resets $s_i = y \oplus s \oplus s_i \oplus s_1 \oplus \cdots \oplus s_n$ and returns $(X_i^{w_i}, s_i)$, so that evaluation of the garbled function indeed results in output y.

This simulation strategy is intuitive, but trying to prove it correct runs into problems. We have to show that $\mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2, \cdot)$ is negligible. We must utilize the assumption of prv security to do this, which means we must perform a reduction. The only plausible path towards this is to construct from $\mathcal{A}_2$ an adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$ and then exploit the existence of a simulator $\mathcal{S}$ such that $\mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible. However, it is not clear how to construct $\mathcal{A}$, let alone how its simulator comes into play.

The problem turns out to be more than technical, for we will see that the transform itself does not work in general. By this we mean that we can exhibit a (projective) circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ that is prv secure relative to $\Phi = \Phi_{\mathrm{topo}}$ but the transformed scheme $\mathcal{G}_2 = \mathsf{OMSS}[\mathcal{G}]$ is subject to an attack showing that it is not prv2 secure. This means, in particular, that the above simulation strategy does not in general work.

To carry this out, we start with an arbitrary projective circuit-garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ assumed to be prv secure relative to $\Phi = \Phi_{\mathrm{topo}}$. We then transform it into the projective circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ shown at the bottom of Fig. 5.2.2. The idea is as follows. We choose m-bit random shares V_i^0, V_i^1 for every $i \le n$, and distribute them to the tokens. Next, choose a "poisoned" point $v = v_1 \cdots v_n$ at random, and append it to the garbled function, making it trivial for an adaptive adversary to query $x = v$. Since v is random, a static adversary can guess v with probability only 2^{-n}. To make sure this probability is negligible in terms of k, we only do the following trick if $n \ge k$. Let V be the encryption of $\mathsf{ev}(g, \overline{v})$ by using the one-time pad constructed from the shares corresponding to v, namely, the pad is the checksum of $V_1^{v_1}, \ldots, V_n^{v_n}$. Append V to the garble function as well. So if the adversary queries $x = v$ then it will learn $\mathsf{ev}(g, \overline{v})$ in addition to $\mathsf{ev}(g, v)$; while if $x \ne v$ then the shares the adversary receives won't allow it to decrypt V. The following proposition says that $\mathcal{G}$ continues to be prv secure but an attack shows that $\mathsf{OMSS}[\mathcal{G}]$ is not

prv2 secure. (The proof shows it is in fact not even prv1 secure.)

Proposition 5.2.1. Let ev be the canonical circuit-evaluation function. Assume $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj})$ and let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{proj})$ be the garbling scheme shown at the bottom of Fig. 5.2.2. Then (1) $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj})$, but (2) $\mathsf{OMSS}[\mathcal{G}] \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$.

Proof. First let us justify (1). Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}$. Assume that the circuit f in $\mathcal{A}$'s query satisfies $f.n \geq k$; otherwise $\mathcal{G}$ will inherit the prv security from $\mathcal{G}'$, as it only appends to garbled function and each token a random string independent of anything else. Let the garbled function be (G', v, V). Unless $\mathcal{A}$ manages to query $x = v$, the same argument applies and $\mathcal{G}$ will again inherit the prv security of $\mathcal{G}'$. Since $v \twoheadleftarrow \{0,1\}^n$, the chance that $x = v$ is $2^{-n} \leq 2^{-k}$.

Now, we justify (2) via the following attack. Adversary $\mathcal{A}_2(1^k)$ picks bits $R_0, R_1 \twoheadleftarrow \{0,1\}$, and lets f_{R_0,R_1} be a circuit such that $f_{R_0,R_1}.n = k, f_{R_0,R_1}.m = 1$ and $\mathsf{ev}(f_{R_0,R_1}, x) = R_{x_1}$ where x_1 is the first bit of x. (We note that we construct the circuit in such a way that the topology is independent of R_0, R_1 and depends only on k.) It queries f_{R_0,R_1} to Garble to get back (G, ε). It parses $(G', v, V) \leftarrow G$ and $v_1 \cdots v_n \leftarrow v$. Next for $i = 1, \ldots, n$ it queries (i, v_i) to Input to get back T_i and lets $(X_i, r_i) \leftarrow T_i$ and $(Z_i, V_i) \leftarrow X_i$. It lets $y \leftarrow \mathsf{De}_2\big(\varepsilon, \mathsf{Ev}_2(G, (T_1, \ldots, T_n))\big)$ and $y' \leftarrow V \oplus V_1 \oplus \cdots \oplus V_n$ and $r \leftarrow r_1 \oplus \cdots \oplus r_n$. If $y \oplus y' \oplus r = R_0 \oplus R_1$ then it returns 1 else it returns 0.

Let $\mathcal{S}_2$ be *any* PT simulator and consider game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$. We claim that $\mathcal{A}_2(1^k)$ returns 1 with probability 1 if the challenge bit b in the game is 1. This is because in this case we have $y = \mathsf{ev}(f_{R_0,R_1}, v)$ and $y' = r \oplus \mathsf{ev}(f_{R_0,R_1}, \overline{v})$ so by definition of f_{R_0,R_1} we have $y \oplus y' \oplus r = R_0 \oplus R_1$. Next we claim that $\mathcal{A}_2(1^k)$ returns 1 with probability at most 1/2 if the challenge bit b is 0. (We emphasize that this claim is made regardless of the strategy of the simulator, showing that no simulator could possibly do well.) In the first phase, the simulator $\mathcal{S}_2$ is given $1^k, \Phi_{\mathrm{topo}}(f), 0$ as input and can obtain no information on R_0 or R_1 beyond their length because the topology of f_{R_0,R_1} is by construction independent of R_0, R_1.

$\mathbf{proc}\ \mathsf{Gb}_1(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
$Z_{\mathtt{F}} \twoheadleftarrow \{0,1\}^{|F|},\ \ Z_{\mathtt{d}} \twoheadleftarrow \{0,1\}^{|d|}$
$F_1 \leftarrow F \oplus Z_{\mathtt{F}},\ \ d_1 \leftarrow d \oplus Z_{\mathtt{d}}$
$e_1 \leftarrow (e, Z_{\mathtt{d}}, Z_{\mathtt{F}})$
$\mathbf{return}\ (F_1, e_1, d_1)$

$\mathbf{proc}\ \mathsf{Ev}_1(F_1, X_1)$
$(X, Z_{\mathtt{d}}, Z_{\mathtt{F}}) \leftarrow X_1,\quad F \leftarrow F_1 \oplus Z_{\mathtt{F}}$
$Y \leftarrow \mathsf{Ev}(F, X)$
$\mathbf{return}\ (Y, Z_{\mathtt{d}})$

$\mathbf{proc}\ \mathsf{En}_1(e_1, x)$
$(e, Z_{\mathtt{d}}, Z_{\mathtt{F}}) \leftarrow e_1,\quad X \leftarrow \mathsf{En}(e, x)$
$\mathbf{return}\ (X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$

$\mathbf{proc}\ \mathsf{De}_1(d_1, Y_1)$
$(Y, Z_{\mathtt{d}}) \leftarrow Y_1,\ \ d \leftarrow d_1 \oplus Z_{\mathtt{d}}$
$\mathbf{return}\ \mathsf{De}(d, Y)$

$\mathbf{proc}\ \mathsf{Gb}_2(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}_1(1^k, f)$
$(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
$N \leftarrow |\mathsf{En}_1(e, 0^n)|$
$\mathbf{for}\ i \in \{1, \ldots, n\}\ \mathbf{do}$
$\quad Z_i \twoheadleftarrow \{0,1\}^{|X_i^0|},\quad S_i \twoheadleftarrow \{0,1\}^N$
$Z \leftarrow (Z_1, \ldots, Z_n)$
$S_n \leftarrow Z \oplus S_1 \oplus \cdots \oplus S_{n-1}$
$\mathbf{for}\ i \in \{1, \ldots, n\}\ \mathbf{do}$
$\quad T_i^0 \leftarrow (X_i^0 \oplus Z_i, S_i),\ \ T_i^1 \leftarrow (X_i^1 \oplus Z_i, S_i)$
$\mathbf{return}\ (F, (T_1^0, T_1^1, \ldots, T_n^0, T_n^1), d)$

$\mathbf{proc}\ \mathsf{Ev}_2(F, X_2)$
$\big((U_1, S_1), \ldots, (U_n, S_n)\big) \leftarrow X_2$
$Z \leftarrow S_1 \oplus \cdots \oplus S_n$
$(Z_1, \ldots, Z_n) \leftarrow Z$
$X \leftarrow (U_1 \oplus Z_1, \ldots, U_n \oplus Z_n)$
$\mathbf{return}\ \mathsf{Ev}_1(F, X)$

$\mathbf{proc}\ \mathsf{En}_2(e_2, x)$
$(T_1^0, X_1^1, \ldots, T_n^0, X_n^1) \leftarrow e_2$
$x_1 \cdots x_n \leftarrow x$
$\mathbf{return}\ (T_1^{x_1}, \ldots, T_n^{x_n})$

Figure 5.2.3: **Transform prv-to-prv1 (top):** Scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$ obtained by applying the prv-to-prv1 transform to $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi)$. **Transform prv1-to-prv2 (bottom):** Projective garbling scheme $\mathcal{G}_2 = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv2}, \Phi)$ obtained by applying the prv1-to-prv2 transform to projective garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$.

In the second phase, the only useful information that the sender gets is $y = \mathsf{ev}(f_{R_0,R_1}, v)$. It thus learns R_{v_1} but it has no information about R_{1-v_1} and thus the probability that the $y' \oplus r$ computed by the adversary equals $y \oplus R_0 \oplus R_1$ is at most $1/2$. □

GKR had stated their transform only for circuits with boolean output, meaning $f.m = 1$. We have accordingly presented our counter-example above for this case.

5.2.3 Achieving prv1 security

We now describe a transform prv-to-prv1 that successfully turns a prv secure circuit garbling scheme into a prv1 secure one. Combined with established results in Section 3.5, this yields prv1 secure schemes based on standard assumptions. The idea (cf. [46]) is to use one-time

pads to mask F and d, and then append the pads to X. This will ensure that the adversary learns nothing about F and d until it fully specifies function f and x. Given a (not necessarily projective) garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, the prv-to-prv1 transform returns the garbling scheme $\textsf{prv-to-prv1}[\mathcal{G}] = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ at the top of Fig. 5.2.3. We claim:

Theorem 5.2.2. For any Φ, if $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi)$ then $\textsf{prv-to-prv1}[\mathcal{G}] \in \mathsf{GS}(\mathrm{prv1}, \Phi)$.

The proof sketch is as follows. Given any PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$, we build a PT adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$. Now the assumption of prv security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible. Now we build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$ we have $\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k)$. This yields the theorem. In Section 5.6.1 we provide a full proof that shows how to build $\mathcal{A}$ and $\mathcal{S}_1$. The idea for the latter is that in its first stage, $\mathcal{S}_1$, given $(1^k, \phi, 0)$, returns random F_1 and d_1. In the second phase, given y, it lets $(F, X, d) \twoheadleftarrow \mathcal{S}(1^k, \phi, y)$, $Z_{\mathtt{F}} \leftarrow F_1 \oplus F$, and $Z_{\mathtt{d}} \leftarrow d_1 \oplus d$. It returns $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$. The formal proof must attend to some pesky issues connected with the need for the simulator to know what length it must pick for F_1 and d_1.

Transform prv-to-prv1 does not require the starting scheme $\mathcal{G}$ to be projective. However, it is important that if $\mathcal{G}$ is projective, so is $\textsf{prv-to-prv1}[\mathcal{G}]$. Seeing this requires a slight re-interpretation of certain quantities in the algorithms at the top of Fig. 5.2.3. Specifically, e will now have the form $(X^0_1, X^1_1, \ldots, X^0_n, X^1_n)$ and Gb_1 will let

$$e_1 = ((X^0_1, Z_{\mathtt{d}}, Z_{\mathtt{F}}), (X^1_1, Z_{\mathtt{d}}, Z_{\mathtt{F}}), X^0_2, X^1_2, \ldots, X^0_n, X^1_n).$$

Also X in En_1 will have the form $(X_1, \ldots, X_n)$ and En_1 will return $((X_1, Z_{\mathtt{d}}, Z_{\mathtt{F}}), X_2, \ldots, X_n)$.

A potentially simpler transform of a prv secure garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ into a prv1 secure garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ is as follows. Algorithm $\mathsf{Gb}_1(1^k, f)$ lets $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f)$ and returns $(\varepsilon, (e, F, d), \varepsilon)$. Let $\mathsf{En}_1((e, F, d), x) = (\mathsf{En}(e, x), F, d)$. Let $\mathsf{Ev}_1(\varepsilon, (X, F, d)) = (\mathsf{Ev}(F, X), d)$. Let $\mathsf{De}_1(\varepsilon, (Y, d)) = \mathsf{De}(d, Y)$. This works, but the scheme does not meet the non-degeneracy requirement. The prv-to-prv1 transform can be seen as a way to effectively implement this trivial transform while avoiding degeneracy.

5.2.4 Achieving prv2 security

Next we show how to transform a prv1 scheme into a prv2 one. Formally, given a projective garbling scheme $\mathcal{G} = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$, the prv1-to-prv2 transform returns the projective garbling scheme $\mathsf{prv1\text{-}to\text{-}prv2}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev})$ shown at the bottom of Fig. 5.2.3. The idea is to mask the garbled input and then use the second part of GKR's idea as represented by OMSS, namely secret-share the mask, putting a piece in each token, so that unless one has all tokens, one learns nothing about the garbled input. The formal proof of the following is in Section 5.6.2.

Theorem 5.2.3. For any side-information function Φ, if $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then $\mathsf{prv1\text{-}to\text{-}prv2}[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$.

The proof sketch is as follows. We first build, from a given prv2 adversary $\mathcal{A}_2$, a prv1 adversary $\mathcal{A}_1$, and then, from the simulator $\mathcal{S}_1$ for the latter, a simulator $\mathcal{S}_2$ for $\mathcal{A}_2$. The prv2 simulator $\mathcal{S}_2$ can return random tokens for the first $n-1$ bits of the input. Just before it must provide a token for the very last input bit, it gets the final output y. Now, it can run the prv1 simulator on y to get the real tokens and create the last piece of the secret mask and thence its last token so that the shares unmask the real tokens.

5.2.5 Efficient ROM transforms

We say that garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ has *short garbled inputs* if there is a polynomial s such that $|\mathsf{En}(e, x)| \leq s(k, f.n, f.m)$ for all $k \in \mathbb{N}$, $f \in \{0,1\}^*$, $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$, and $x \in \{0,1\}^{f.n}$. Let T be a transform that maps a garbling scheme $\mathcal{G}$ to a garbling scheme $\mathsf{T}[\mathcal{G}]$. We say that T *preserves* short garbled inputs if $\mathsf{T}[\mathcal{G}]$ has short garbled inputs when $\mathcal{G}$ does.

The prv-to-prv1 transform does not preserve short garbled inputs, meaning even if $\mathcal{G}$ has short garbled inputs, $\mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}]$ may not. The prv1-to-prv2 transform preserves short garbled inputs, but we usually want to apply the two transforms in sequence. We do not

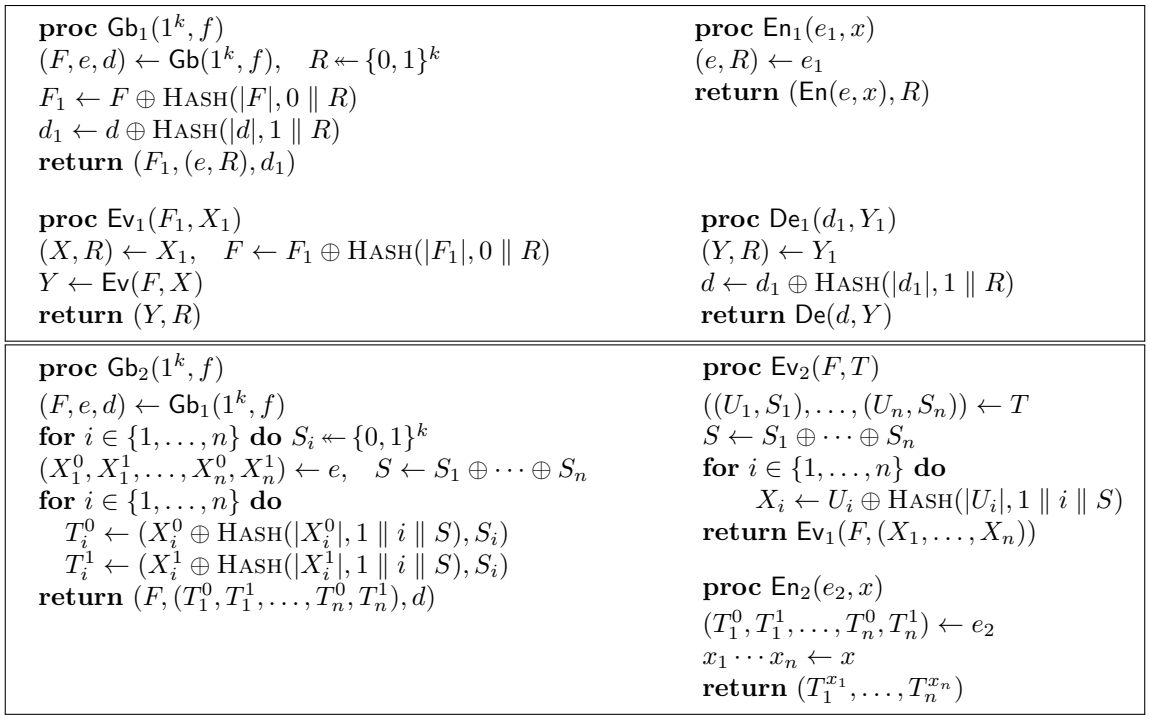

Figure 5.2.4: **Transform rom-prv-to-prv1 (top):** Garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv1}, \Phi)$ obtained by applying the ROM rom-prv-to-prv1 transform to garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi)$. **Transform rom-prv1-to-prv2 (bottom):** Projective garbling scheme $\mathcal{G}_2 = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv2}, \Phi)$ obtained by applying the ROM rom-prv1-to-prv2 transform to projective garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$. The advantage of these transforms over the ones of Fig. 5.2.3 is that they preserve short garbled inputs.

know how to fill this gap in the standard model under standard assumptions. We will now provide a simple way to do it in the ROM (random-oracle model). Recall Section 3.3.4 for the extension of garbling-scheme privacy in the ROM. For xxx $\in \{\mathrm{prv}, \mathrm{prv1}, \mathrm{prv2}\}$ we let $\mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx}, \Phi)$ be the set of all garbling schemes that are xxx secure over Φ in the ROM.

The rom-prv-to-prv1 transform at the top of Fig. 5.2.4 generates the mask of the prv-to-prv1 transform by applying the RO to a random k-bit seed R, and includes R in the encoding function and garbled input and output in place of the full mask, thereby saving space. As a consequence, it preserves short garbled inputs. We claim:

Theorem 5.2.4. If $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi)$ then $\textsf{rom-prv-to-prv1}[\mathcal{G}] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv1}, \Phi)$, for any side-information function Φ.

The proof is in Section 5.6.3. The idea is standard. The simulator can pick F_1, d_1 at

random just as in the proof of Theorem 5.2.2. Then, once it has F, d, it will pick R at random and program the RO so that $F_1 = F \oplus \textsc{Hash}(|F|, 0\|R)$ and $d_1 = d \oplus \textsc{Hash}(|d|, 1\|R)$. Security relies on the fact that the probability that the adversary queries (ℓ, w) to HASH, with R being the suffix of w, prior to receiving R in the garbled input, is negligible.

As with prv-to-prv1, we note that the starting scheme is not assumed projective, but a suitable re-interpretation of the notation is enough to ensure that if the starting scheme is projective, so is the constructed one.

Our prv1-to-prv2 already preserves short garbled inputs, but the size of a token in the constructed scheme is n times the size of a token in the original scheme. The rom-prv-to-prv1 transform at the bottom of Fig. 5.2.4 does a little better, increasing the size of each token by an additive nk bits regardless of the length of the tokens of the starting scheme. The idea is again to generate the masks of the prv1-to-prv2 transform by applying the RO to a seed and then secret-sharing the latter instead of the entire mask. The proof of the following, in Section 5.6.4, is again standard:

Theorem 5.2.5. For any side-information function Φ, if $\mathcal{G}_1 \in \mathsf{GS}(\text{prv1}, \Phi) \cap \mathsf{GS}(\text{proj})$ then rom-prv1-to-prv2$[\mathcal{G}_1] \in \mathsf{GS}^{\text{rom}}(\text{prv2}, \Phi) \cap \mathsf{GS}(\text{proj})$.

As the statements of Theorems 5.2.4 and 5.2.5 indicate, we are assuming in both cases that the starting scheme is a standard-model one. This is for simplicity. One can apply the transform to a ROM scheme. (And, in the case of rom-prv1-to-prv2, are likely to, since the starting scheme is likely an output of rom-prv-to-prv1.) This can be handled by suitable "domain separation" of all ROs involved.

For conceptual simplicity we have presented two separate transforms but we note that one can gain efficiency by going directly from prv to prv2. We would not pick S as in rom-prv1-to-prv2 but instead apply the secret-sharing directly to the R chosen by the transform rom-prv-to-prv1.

5.2.6 "Standard" schemes are not prv2 secure

It is easy to see that prv security does not in general imply prv1 or prv2 security, meaning that there exist prv secure schemes that are not prv1 (and thus not prv2) secure (cf. Proposition 5.5.1). A more interesting question concerns the adaptive security of "standard" constructions of garbled circuits, meaning garbling schemes in the Yao style such as the Garble1 and Garble2 schemes in Sections 3.5 and 3.6 or the scheme of Lindell and Pinkas [70]. These are prv secure. But are they prv1 or prv2 secure? Here we show that they are not prv2 secure. This is for a fundamental reason, namely that they permit what we call *partial evaluation*: if certain output bits depend only on certain input bits, having the tokens for these input bits (and having the decoding rule, but not the tokens, for other input bits) allows one to compute the corresponding output bits. We will show that any scheme with this property is prv2 insecure. But the partial-evaluation property is possessed by all schemes that use the token-based, gate-encryption paradigm of Yao, in particular the ones mentioned above, and thus our results will imply that these schemes are not prv2 secure. We now proceed to formalize and prove this claim, defining what it means for a garbling scheme to permit partial evaluation and then showing that any scheme with this property fails to be prv2 secure.

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective circuit-garbling scheme, so that ev is the canonical circuit-evaluation algorithm, taking as input a circuit f and $x \in \{0,1\}^{f.n}$ to return $\mathsf{ev}(f,x) \in \{0,1\}^{f.m}$. We extend ev to a *partial circuit evaluation algorithm* $\overline{\mathsf{ev}}$ that takes f and $x \in \{0,1,\bot\}^{f.n}$ and returns $\overline{\mathsf{ev}}(f,x) \in \{0,1,\bot\}^{f.m}$ as follows:

proc $\overline{\mathsf{ev}}(f,x)$
$(n,m,q,A,B,G) \leftarrow f$
for $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A(g)$, $b \leftarrow B(g)$
  if $(x_a = \bot$ **or** $x_b = \bot)$ **then** $x_g \leftarrow \bot$ **else** $x_g \leftarrow G_g(x_a, x_b)$
return $x_{n+q-m+1} \cdots x_{n+q}$

Note that $\overline{\mathsf{ev}}(f,x) = \mathsf{ev}(f,x)$ if $x \in \{0,1\}^{f.n}$. Partial evaluation captures an inherent property of circuit evaluation, namely the ability to compute a part of the output given only the inputs on which it depends. For example if the first bit of $\mathsf{ev}(f,x)$ depends only on the first two bits of x, then this first output bit can be computed as the first bit of $\overline{\mathsf{ev}}(f, x_1x_2\bot\cdots\bot)$.

We say that $\mathcal{G}$ permits partial evaluation of the garbled function if the above property is inherited by the garbled-evaluation process. Thus if, as in the above example, the first bit of $\mathsf{ev}(f,x)$ depends only on the first two bits of x, then this first output bit can be computed given the garbled function F, the tokens $X_1^{x_1}, X_2^{x_2}$ and the decoding rule d, meaning tokens corresponding to the other bits of the input are not necessary. Formally we say that $\overline{\mathsf{Ev}}$ is a *partial garbled-evaluation algorithm* for $\mathcal{G}$ if for any $f \in \{0,1\}^*$, any $(F, (X_1^0, X_1^1, \ldots, X_{f.n}^0, X_{f.n}^1), d) \in [\mathsf{Gb}(1^k, f)]$, and any $x \in \{0,1,\bot\}^{f.n}$, if we let $X_i^{\bot} = \bot$ for $1 \le i \le f.n$, then

$$\mathsf{De}\big(d, \overline{\mathsf{Ev}}(F, (X_1^{x_1}, \ldots, X_n^{x_n}))\big) = \overline{\mathsf{ev}}(f,x)\ .$$

In other words, tokens may now take value $\bot$, and evaluation of the garbled circuit is still possible, the result being the corresponding partial evaluation of the circuit. We say that $\mathcal{G}$ *permits partial evaluation* if it has a PT partial garbled-evaluation algorithm. The following says this condition implies that $\mathcal{G}$ is not prv2 secure:

Proposition 5.2.6. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective circuit-garbling scheme that permits partial evaluation. Then $\mathcal{G} \notin \mathsf{GS}(\mathrm{prv2}, \Phi)$ for all Φ.

The result is quite strong with regard to side-information, saying the scheme is insecure for *all* side-information functions. As we indicated above, standard garbling schemes based on the Yao paradigm of encrypted gate entries and token propagation do permit partial evaluation, so this result rules out their prv2 security.

Proof. For $k \in \mathbb{N}$ let $\mathrm{ID}_k\colon\ \{0,1\}^{k+1} \to \{0,1\}^{k+1}$ denote the identity function and let id_k denote a circuit such that $id_k.n = k+1$, $\mathsf{ev}(id_k, \cdot) = \mathrm{ID}_k(\cdot)$, and $\overline{\mathsf{ev}}(id_k, x_1\cdots x_k\bot) = x_1\cdots x_k\bot$ for every $x_1, \ldots, x_k \in \{0,1\}$. Let $\overline{\mathsf{Ev}}$ be the partial garbled-evaluation algorithm

associated to $\mathcal{G}$. Consider the following adversary:

adversary $\mathcal{A}_2(1^k)$, $(F, d) \leftarrow \textsc{Garble}(id_k)$
$x \twoheadleftarrow \{0,1\}^{k+1}$, $x_1 \cdots x_{k+1} \leftarrow x$
for $i \in \{1, \ldots, k\}$ **do** $X_i \leftarrow \textsc{Input}(i, x_i)$
$z \leftarrow \mathsf{De}(d, \overline{\mathsf{Ev}}(F, (X_1, \ldots, X_k, \bot)))$
if $z = x_1 \cdots x_k \bot$ **then return** 1 **else return** 0

Let $\mathcal{S}_2$ be any (even computationally unbounded) simulator. Then for every $k \in \mathbb{N}$, letting b be the challenge bit in game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$, we have

$$\Pr\left[\,\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G},\Phi,\mathcal{S}_2}(k) \mid b = 1\,\right] = 1 \quad \text{and} \quad \Pr\left[\,\neg\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G},\Phi,\mathcal{S}_2}(k) \mid b = 0\,\right] \leq 2^{-k}\ .$$

The first equation uses the assumption that $\overline{\mathsf{Ev}}$ is a partial garbled-evaluation algorithm. The second equation is true because $\mathcal{S}$ has no information about the input x until the very last token is requested, and the adversary stops just short of that. Subtracting we have

$$\mathbf{Adv}^{\mathrm{prv2},\,\Phi,\,\mathcal{S}_2}_{\mathcal{G}}(\mathcal{A}_2, k) \geq 1 - 2^{-k}\ ,$$

which proves the theorem. □

5.2.7 One-time programs

Security definition for a one-time compiler. The notion of a *one-time program* was put forward by Goldwasser, Kalai, and Rothblum (GKR [45]). The intent is that possession of a one-time program P for a function f should enable one to evaluate f at any single value x; but, beyond that, the one-time program should be useless. Unachievable in any standard model of computation (where possession of P would enable its repeated evaluation at multiple point), GKR suggest achieving one-time programs in a model of computation that provides *one-time memory*—tamper-resistant hardware whose read-once i-th location returns, on query $(i, b) \in \mathbb{N} \times \{0,1\}$, the string T_i^b, immediately thereafter expunging T_i^{1-b}. A *one-time compiler* probabilistically transforms the description of a function f into a one-time program P and its associated one-time memory T.

For a formal treatment, we begin by specifying two stateful oracles; see Fig. 5.2.5. The

proc $\mathrm{OTP}_f(x)$	**proc** $\mathrm{OTM}_T(i,b)$
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$	$(T_1^0, T_1^1, \ldots, T_\ell^0, T_\ell^1) \leftarrow T$
if *called* **then return** $\perp$	**if** $i \notin [1..\ell]$ **or** $used_i$ **or** $b \notin \{0,1\}$ **then return** $\perp$
$called \leftarrow \mathsf{true}$	$used_i \leftarrow \mathsf{true}$
return $\mathsf{ev}_{\mathsf{circ}}(f,x)$	**return** T_i^b

Figure 5.2.5: **Oracles model one-time programs and one-time memory.** Oracle OTP depends on a string f representing a boolean circuit. Oracle OTM depends on a list of strings T.

first, OTP_f, formalizes the desired behavior of a one-time program for f. Here f will now be regarded as a string, not a function, but this string represents a circuit computing a function $\mathsf{ev}_{\mathsf{circ}}(f) : \{0,1\}^{f.n} \to \{0,1\}^{f.m}$; we write $\mathsf{ev}_{\mathsf{circ}}$ for the canonical circuit-evaluation function. The agent calling out to OTP_f provides x and, on the first query, it gets $\mathsf{ev}_{\mathsf{circ}}(f,x)$. Subsequent queries return nothing. On the right-hand side of Fig. 5.2.5 we similarly define an oracle OTM_T, this to model possession of a one-time-memory system. Given a list of ℓ pairs of strings (establish some convention so that every string T is regarded as denoting a list of ℓ pairs of strings, for some $\ell \in \mathbb{N}$), the oracle returns at most one string from each pair satisfying each request.

Elaborating on GKR, we now define a *one-time compiler* as a pair of probabilistic algorithms $\Pi = (\mathsf{Co}, \mathsf{Ex})$ (for *compile* and *execute*). Algorithm Co, on input 1^k and a string f, produces a pair $(P,T) \leftarrow \mathsf{Co}(1^k, f)$ where P (the one-time program) is a string and T (the one-time-memory) encodes a list of 2ℓ strings, for some ℓ. Algorithm Ex, on input of strings P and x, and given access to an oracle $\mathcal{O}$, returns a string $y \leftarrow \mathsf{Ex}^{\mathcal{O}}(P,x)$. We require the following *correctness* condition of $\Pi = (\mathsf{Co}, \mathsf{Ex})$: if $(P,T) \leftarrow \mathsf{Co}(1^k, f)$ and $x \in \{0,1\}^{f.n}$ then $\mathsf{Ex}^{\mathrm{OTM}_T(\cdot,\cdot)}(P,x) = \mathsf{ev}_{\mathsf{circ}}(f,x)$.

The security of $\Pi = (\mathsf{Co}, \mathsf{Ex})$ will be relative to a side-information function Φ; the value $\phi = \Phi(f)$ captures the information about f that P is allowed to reveal.[4] So fix a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$, an adversary $\mathcal{A}$, a security parameter k, and a string f. (1) Consider the distribution $\mathrm{Real}_{\Pi,\mathcal{A},f}(k)$ determined by the following experiment: first, sample $(P,T) \leftarrow$

[4]For example, we might have $\Phi(f) = \Phi_{\mathrm{size}}(f) = (f.n, f.m, f.q)$, the number of inputs, outputs, and gates; or $\Phi(f) = \Phi_{\mathrm{topo}}(f) = f^-$, the topology of f; or $\Phi(f) = (f.n, f.m, u(f.q))$ for some monotonic u like $u(q) = 10^6\lceil 10^{-6}q \rceil$.

$\mathsf{Co}(1^k, f)$; then, run $\mathcal{A}^{\mathrm{OTM}_T(\cdot)}(1^k, P)$ and output whatever $\mathcal{A}$ outputs. (2) Alternatively, fix a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$, a side-information function Φ, a simulator $\mathcal{S}$, a security parameter k, and a string f. Consider the distribution $\mathrm{Fake}_{\Pi,\Phi,\mathcal{S},f}(k)$ determined by the following experiment: run $S^{\mathrm{OTP}_f(\cdot)}(1^k, \Phi(f))$ and output whatever S outputs. For $\mathcal{D}$ an algorithm and Π, Φ, $\mathcal{A}$, $\mathcal{S}$, and k as above, let

$$\begin{aligned}\mathbf{Adv}^{\mathrm{otc}}_{\Pi,\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k) \;=\; & \Pr[(f,\sigma) \leftarrow \mathcal{D}(1^k); v \leftarrow \mathrm{Real}_{\Pi,\mathcal{A},f}(k) \colon\ \mathcal{D}(\sigma, v) \Rightarrow 1] - \\ & \Pr[(f,\sigma) \leftarrow \mathcal{D}(1^k); v \leftarrow \mathrm{Fake}_{\Pi,\Phi,\mathcal{S},f}(k) \colon\ \mathcal{D}(\sigma, v) \Rightarrow 1]\end{aligned}$$

One-time compiler Π is said to be (OTC-) *secure* with respect to side-information function Φ if for any PPT adversary $\mathcal{A}$ there is a PPT simulator $\mathcal{S}$ such that for all PPT distinguishers $\mathcal{D}$, function $\mathbf{Adv}^{\mathrm{otc}}_{\Pi,\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k)$ is negligible.

Discussion. Let us briefly talk through the definition. The distinguisher $\mathcal{D}$ selects f and is presented with a string drawn from one of two worlds. In the first world, the distinguisher is given the output (equivalently, the view) of an adversary $\mathcal{A}$ who has the garbled program P for f and its associated one-time memory. Using the execution procedure Ex the adversary could compute $\mathsf{ev}_{\mathsf{circ}}(f, x)$, if it so wishes, but it is not compelled to do so. In the second world, the distinguisher is given output produced by a simulator $\mathcal{S}$. That simulator has no one-time memory; it has only the side-information $\Phi(f)$ about f and an ideal one-time program for f. In a protocol we deem secure, no matter what the adversary does, there will be a simulator such that the two views described will be computationally close.

To arrive at an achievable notion of security, one *must* allow that information beyond the function's value at x to be leaked; minimally, information on the size of the circuit will be revealed. Indeed the construction of GKR leaks more—it divulges the *topology* of a circuit computing f. Again a side-information function Φ comes into play, acting as a "knob" controlling just what may be learned of f.

Constructing an OTC from a garbling scheme. A projective circuit-garbling

scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ can be turned into a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$ in a natural way: let $\mathrm{OTC}[\mathcal{G}] = (\mathsf{Co}, \mathsf{Ex})$ be defined as follows. (1) $\mathsf{Co}(1^k, f)$: let $(F, e, d) \leftarrow \mathsf{Gb}(f)$ and return (P, T) where $P = (F, d)$ and $T = e$. (2) $\mathsf{Ex}^{\mathcal{O}}(P, x)$: Let $(F, d) \leftarrow P$, let $x_1 \cdots x_n \leftarrow x$, query oracle $\mathcal{O}$ on $(1, x_1), \ldots, (n, x_n)$ to obtain $X_1, \ldots, X_n$, respectively, and return $\mathsf{De}(d, \mathsf{Ev}(F, X))$ with $X = (X_1, \ldots, X_n)$. The straightforwardness of the construction and its trivial proof are, we believe, points in our favor, evidence of our claim that the garbling-scheme abstraction and appropriate security notions for it engender applications in direct, simple and less error-prone ways. A concrete one-time compiler may be obtained from any prv-secure (projective) garbling scheme by (1) using our prv-to-prv1 transform to go from the prv garbling scheme to a prv1 one (2) using our prv1-to-prv2 transform to go from the prv1 scheme to a prv2 one, and (3) applying Theorem 5.2.7.

Theorem 5.2.7. If $\mathcal{G}$ is a prv2-secure projective scheme over Φ then $\mathrm{OTC}[\mathcal{G}]$ is OTC-secure over Φ.

Proof. Let $\mathcal{S}'$ be a simulator to which $\mathcal{G}$ is prv2 secure. Fix an adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$. Consider the following simulator $\mathcal{S}$. On input 1^k and $\phi = \Phi(f)$, it gets $(F, d) \leftarrow \mathcal{S}'(1^k, \phi, 0)$, initializes $Q = \emptyset$ and $\tau = \perp$, and then runs $\mathcal{A}(1^k, (F, d))$. Let $n = f.n$. Whenever $\mathcal{A}$ queries (i, b), the simulator $\mathcal{S}$ proceeds as follows:

if $i \notin \{1, \ldots, n\} \backslash Q$ **then return** $\perp$
$Q \leftarrow Q \cup \{i\}, \;\; x_i \leftarrow b$
if $|Q| = n$ **then** $x \leftarrow x_1 \cdots x_n, \;\; \tau \leftarrow y \leftarrow \mathrm{OTP}_f(x)$
$X_i \leftarrow \mathcal{S}'(\tau, i, |Q|)$

and then returns X_i to $\mathcal{A}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. Consider the following adversary $\mathcal{B}(1^k)$ attacking $\mathcal{G}$. It runs $\mathcal{D}(1^k)$. When the latter queries f, the former queries f to its oracle Garble to get (F, d). It then runs $\mathcal{A}(1^k, (F, d))$. For each query (i, b) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i, b) to its oracle Input, and gives the answer to $\mathcal{A}$. Finally, $\mathcal{B}$ returns $\mathcal{A}$'s output to $\mathcal{D}$. If the challenge bit c of game $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}'}$ is 1 then $\mathcal{B}$ is giving $\mathcal{D}$ the distribution $\mathrm{Real}_{\mathrm{OTC}[\mathcal{G}],\mathcal{A},f}(k)$. Otherwise, if $c = 0$ then $\mathcal{B}$ gives $\mathcal{D}$ the distribution

$\mathrm{Fake}_{\mathrm{OTC}[\mathcal{G}],\Phi,\mathcal{S},f}(k)$. Hence $\mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}'}_{\mathcal{G}}(\mathcal{B},k) = \mathbf{Adv}^{\mathrm{otc}}_{\mathrm{OTC}[\mathcal{G}],\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k)$. □

Analysis of OTC[$\mathsf{OMSS}[\mathcal{G}]$]. The claim of GKR [45], in our language, is that if $\mathcal{G}$ is a prv-secure (projective) garbling scheme then OTC[$\mathsf{OMSS}[\mathcal{G}]$] is otc-secure. Proposition 5.2.1, showing that $\mathsf{OMSS}[\mathcal{G}]$ need not be prv2-secure, does not refute this claim, for the prv2 security of $\mathsf{OMSS}[\mathcal{G}]$, while sufficient to establish the claim, may not be necessary. Here we accordingly refute the claim by extending the counter-example of Proposition 5.2.1 to give a projective, prv-secure garbling scheme $\mathcal{G}$ for which OTC[$\mathsf{OMSS}[\mathcal{G}]$] is shown by attack to not be otc-secure. (That is, we show that this transform will yield programs that are not one-time.)

This example does not contradict the updated claim of [44], made in response to our work, of a OTC based on exponentially-hard one-way functions. The latter would correspond, in our language, to the claim that OTC[$\mathsf{OMSS}[\mathcal{G}]$] is a secure OTC if $\mathcal{G}$ has exponential prv-security.

Proceeding to the counter-example, recall that in the proof of Proposition 5.2.1, we gave a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ such that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$ but $\mathcal{G}_2 = \mathsf{OMSS}[\mathcal{G}] \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$. Now we show that OTC[$\mathcal{G}_2$] is otc-insecure, by demonstrating an attack. Distinguisher $\mathcal{D}(1^k)$ picks $R_0, R_1 \twoheadleftarrow \{0,1\}$, and lets f_{R_0,R_1} denote a circuit such that $f_{R_0,R_1}.n = k$, $f_{R_0,R_1}.m = 1$ and $\mathsf{ev}(f_{R_0,R_1}, x) = R_{x_1}$ where x_1 is the first bit of x. (We construct the circuit in such a way that the topology is independent of R_0, R_1 and depends only on k.) It queries f_{R_0,R_1}, and then outputs 1 only if the oracle's answer is $R_0 \oplus R_1$.

The adversary $\mathcal{A}(1^k)$ is given (G, ε), and parses $(G', v, V) \leftarrow G$ and $v_1 \cdots v_n \leftarrow v$. Next for every $i \le n$, it queries (i, v_i) to Input to get back T_i and lets $(X_i, r_i) \leftarrow T_i$ and $(Z_i, V_i) \leftarrow X_i$. It lets $y \leftarrow \mathsf{De}_2(\varepsilon, \mathsf{Ev}_2(G, (T_1, \ldots, T_n)))$ and $y' \leftarrow V \oplus V_1 \oplus \cdots \oplus V_n$ and $r \leftarrow r_1 \oplus \cdots \oplus r_n$. It then returns $y \oplus y' \oplus r$. Note that $y = \mathsf{ev}(f_{R_0,R_1}, v)$ and $y' = r \oplus \mathsf{ev}(f_{R_0,R_1}, \overline{v})$ so by definition of f_{R_0,R_1} we have $y \oplus y' \oplus r = R_0 \oplus R_1$. Hence given $\mathrm{Real}_{\Pi,\mathcal{A},f}(k)$, the distinguisher always outputs 1.

Let $\mathcal{S}$ be any (even computationally unbounded) simulator. It is given only $1^k, \Phi_{\mathrm{topo}}(f)$

as input and can obtain no information on R_0 or R_1 beyond their length because the topology of f_{R_0,R_1} is by construction independent of R_0, R_1. The simulator is given oracle access to $\mathrm{OTP}_{f_{R_0,R_1}}$ and can obtain either R_0 or R_1 but not both. Since R_0 is independent of $R_0 \oplus R_1$, and so is R_1, the probability that the simulator can output $R_0 \oplus R_1$ is $1/2$. Hence, given $\mathrm{Fake}_{\Pi,\Phi,\mathcal{S},f}(k)$, the distinguisher outputs 1 with probability $1/2$.

5.3 Obliviousness, authenticity and application to secure outsourcing

We define obliviousness and authenticity, both with either the coarse-grained or fine-grained adaptivity. We show how to achieve these goals, in combination with adaptive privacy, via generic transforms and in the standard model. We then give more efficient transforms for the ROM model. Finally we apply this to obtain extremely simple and modular designs, and security proofs, for verifiable outsourcing schemes based on the paradigm of GGP [37].

5.3.1 Definitions for adaptive obliviousness and authenticity

Obliviousness. We add two new definitions, to incorporate either coarse-grained or fine-grained adaptive security. See the two top panels of Fig. 5.3.1. Fine-grained adaptive security continues to require that $\mathcal{G}$ be projective. The games used for defining obliviousness closely mirror their privacy counterparts. The first important difference is that the adversary does not get the decoding function d. The second important difference is that the simulator must do without $y = \mathsf{ev}(f, x)$. For a garbling scheme $\mathcal{G}$, side-information Φ, simulator $\mathcal{S}$, adversary $\mathcal{A}$, and security parameter $k \in \mathbb{N}$, let $\mathbf{Adv}^{\mathrm{obv1},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) = 2\Pr[\mathrm{Obv1}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k)] - 1$, and $\mathbf{Adv}^{\mathrm{obv2},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) = 2\Pr[\mathrm{Obv2}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k)] - 1$. Garbling scheme $\mathcal{G}$ is obv1 secure with respect to Φ if for every PPT $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\mathbf{Adv}^{\mathrm{obv1},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k)$ is negligible. We similarly define obv2 security. For xxx $\in \{\mathrm{obv1}, \mathrm{obv2}\}$ we let $\mathsf{GS}(\mathrm{xxx}, \Phi)$

proc GARBLE(f)	**proc** INPUT(x) Obv1$_{\mathcal{G},\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$ **if** $b = 1$ **then** $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ **else** $F \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$ **return** F	**if** $x \notin \{0,1\}^{f.n}$ **then return** $\bot$ **if** $b = 1$ **then** $X \leftarrow \mathsf{En}(e, x)$ **else** $X \leftarrow \mathcal{S}(1)$ **return** X

proc GARBLE(f)	**proc** INPUT(i, c) Obv2$_{\mathcal{G},\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$; $n \leftarrow f.n$; $Q \leftarrow \emptyset$; $\sigma \leftarrow \varepsilon$ **if** $b = 1$ **then** $(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$ **else** $F \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$ **return** F	**if** $i \notin \{1, \ldots, n\} \setminus Q$ **then return** $\bot$ $x_i \leftarrow c$; $Q \leftarrow Q \cup \{i\}$ **if** $b = 1$ **then** $X_i \leftarrow X_i^{x_i}$ **else** $X_i \leftarrow \mathcal{S}(i, \lvert Q \rvert)$ **return** X_i

proc GARBLE(f)	**proc** INPUT(x) Aut1$_{\mathcal{G}}$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ **return** F	**if** $x \notin \{0,1\}^{f.n}$ **then return** $\bot$ $X \leftarrow \mathsf{En}(e, x)$ **return** X

proc GARBLE(f)	**proc** INPUT(i, c) Aut2$_{\mathcal{G}}$
$n \leftarrow f.n$; $Q \leftarrow \emptyset$; $\sigma \leftarrow \varepsilon$ $(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$ **return** F	**if** $i \notin \{1, \ldots, n\} \setminus Q$ **then return** $\bot$ $x_i \leftarrow c$; $Q \leftarrow Q \cup \{i\}$, $X_i \leftarrow X_i^{x_i}$ **if** $\lvert Q \rvert = n$ **then** $X \leftarrow (X_1, \ldots, X_n)$ **return** X_i

Figure 5.3.1: **Obliviousness (top).** Games for defining obv1 and obv2 security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. For each game, INITIALIZE() samples $b \twoheadleftarrow \{0,1\}$ and FINALIZE(b') returns $(b = b')$. **Authenticity (bottom).** Games for defining aut1 and aut2 security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Procedure FINALIZE($Y$) of each game returns 0 if $x \notin \{0,1\}^{f.n}$, otherwise it returns $(\mathsf{De}(d, Y) \neq \bot$ **and** $Y \neq \mathsf{Ev}(F, X))$.

denote the set of all garbling schemes that are xxx secure over Φ.

AUTHENTICITY. Fig. 5.3.1 also formalizes the games underlying two definitions of authenticity, capturing an adversary's inability to create from F and X a garbled output $Y \neq F(X)$ that will be deemed authentic. The static definition is strengthened either to allow the adversary to specify x subsequent to obtaining F, or, stronger, the bits of x are provided one-by-one, each corresponding token then issued. For the second case, game Aut2, the garbling scheme must once again be projective. For a garbling scheme $\mathcal{G}$, adversary $\mathcal{A}$, and security parameter $k \in \mathbb{N}$, we let $\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}}(\mathcal{A}, k) = \Pr[\mathrm{Aut1}^{\mathcal{A}}_{\mathcal{G}}(k)]$, and $\mathbf{Adv}^{\mathrm{aut2}}_{\mathcal{G}}(\mathcal{A}, k) = \Pr[\mathrm{Aut2}^{\mathcal{A}}_{\mathcal{G}}(k)]$. Garbling scheme $\mathcal{G}$ is aut1 secure if $\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}}(\mathcal{A}, k)$ is negligible for every PPT $\mathcal{A}$. We similarly define aut2 security. For xxx $\in \{\mathrm{aut1}, \mathrm{aut2}\}$ we let $\mathsf{GS}(\mathrm{xxx})$ denote

the set of all garbling schemes that are xxx secure.

5.3.2 Achieving adaptive obliviousness and authenticity

ACHIEVING OBV1 AND AUT1. It is tempting to think that the prv-to-prv1 operator in Fig. 5.2.3 will also promote xxx security, with xxx $\in \{\text{obv}, \text{aut}\}$, to xxx1 security. However, a second glance reveals that prv-to-prv1 does not promote aut to aut1, as the following counter-example illustrates. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme that is aut secure. Consider $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}', \mathsf{Ev}, \mathsf{ev})$ defined as follows. On input $(1^k, f)$, the algorithm Gb' creates $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and then returns $(F, e, 1 \,\|\, d)$. On input (d', Y), the algorithm De' parses $d' = b \,\|\, d$, and outputs $\mathsf{De}(d, Y)$ if $b = 1$, and outputs 1 otherwise. The scheme $\mathcal{G}'$ inherits aut security from $\mathcal{G}$. The scheme $\mathcal{G}_1 = \mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}'] = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ is not even aut secure. An adversary can attack $\mathcal{G}_1$ as follows. First, query an arbitrary circuit f and input $x \in \{0,1\}^{f.n}$ to receive (F_1, X_1). Let $X_1 = (X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$. Then, output $Y = (1, \overline{Z_{\mathtt{d}}})$. Let d_1 be the decoding function used to authenticate Y. Then $d_1 \oplus Z_{\mathtt{d}} = 1 \,\|\, d$, and $d_1 \oplus \overline{Z_{\mathtt{d}}} = 0 \,\|\, \overline{d}$. Hence $\mathsf{De}_1(d_1, Y) = 1$, and the adversary wins with advantage 1.

We now show how to change prv-to-prv1 to an operator all-to-all1 that promotes any xxx $\in \{\text{prv}, \text{obv}, \text{aut}\}$ to being xxx1 secure. The insecurity of the prv-to-prv1 operator arises because the adversary can forge a *fake* $Z_{\mathtt{d}}$, where $Z_{\mathtt{d}}$ is the one-time pad masking the decoding function d. To prevent this, we choose $K \twoheadleftarrow \{0,1\}^k$, and append $\mathsf{F}_K(Z_{\mathtt{d}})$ to the garbled input X, where $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ is a PRF. The decoding function will be $(Z_{\mathtt{d}} \oplus d, K)$. See Fig. 5.3.2. The proof of the following is in Section 5.6.5.

Theorem 5.3.1. (1) For any side-information function Φ and any xxx $\in \{\text{prv}, \text{obv}\}$, if $\mathcal{G} \in \mathsf{GS}(\text{xxx}, \Phi)$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\text{xxx1}, \Phi)$ (2) If $\mathcal{G} \in \mathsf{GS}(\text{aut})$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\text{aut1})$ (3) If $\mathcal{G} \in \mathsf{GS}(\text{proj})$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\text{proj})$.

ACHIEVING OBV2 AND AUT2. The transform to promote coarse-grained to fine-grained security is unchanged; we let $\mathsf{all1\text{-}to\text{-}all2} = \mathsf{prv1\text{-}to\text{-}prv2}$ be the transform at the bottom of

proc $\mathsf{Gb}_1(1^k, f)$ $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ $Z_{\mathtt{F}} \twoheadleftarrow \{0,1\}^{\lvert F\rvert}, \quad Z_{\mathtt{d}} \twoheadleftarrow \{0,1\}^{\lvert d\rvert}$ $F_1 \leftarrow F \oplus Z_{\mathtt{F}}, \quad K \twoheadleftarrow \{0,1\}^k, \quad d_1 \leftarrow (d \oplus Z_{\mathtt{d}}, K)$ $\mathrm{tag} \leftarrow \mathsf{F}_K(Z_{\mathtt{d}}), \quad e_1 \leftarrow (e, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathrm{tag})$ **return** (F_1, e_1, d_1)	**proc** $\mathsf{En}_1(e_1, x)$ $(e, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathrm{tag}) \leftarrow e_1$ **return** $(\mathsf{En}(e, x), Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathrm{tag})$
proc $\mathsf{Ev}_1(F_1, X_1)$ $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathrm{tag}) \leftarrow X_1, \quad F \leftarrow F_1 \oplus Z_{\mathtt{F}}$ $Y \leftarrow \mathsf{Ev}(F, X)$ **return** $(Y, Z_{\mathtt{d}}, \mathrm{tag})$	**proc** $\mathsf{De}_1(d_1, Y_1)$ $(Y, Z_{\mathtt{d}}, \mathrm{tag}) \leftarrow Y_1$ $(D, K) \leftarrow d_1, \ \ d \leftarrow D \oplus Z_{\mathtt{d}}$ **if** $\mathrm{tag} \neq \mathsf{F}_K(Z_{\mathtt{d}})$ **then return** $\perp$ **return** $\mathsf{De}(d, Y)$

proc $\mathsf{Gb}_1(1^k, f)$ $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ $R \twoheadleftarrow \{0,1\}^k, \ K \twoheadleftarrow \{0,1\}^k$ $F_1 \leftarrow F \oplus \textsc{Hash}(\lvert F\rvert, 0 \parallel R)$ $D \leftarrow d \oplus \textsc{Hash}(\lvert d\rvert, 1 \parallel R)$ $\mathrm{tag} \leftarrow \textsc{Hash}(k, K \parallel R), \quad d_1 \leftarrow (D, K)$ **return** $(F_1, (e, R, \mathrm{tag}), d_1)$	**proc** $\mathsf{En}_1(e_1, x)$ $(e, R, \mathrm{tag}) \leftarrow e_1$ **return** $(\mathsf{En}(e, x), R, \mathrm{tag})$
proc $\mathsf{Ev}_1(F_1, X_1)$ $(X, R, \mathrm{tag}) \leftarrow X_1$ $F \leftarrow F_1 \oplus \textsc{Hash}(\lvert F_1\rvert, 0 \parallel R)$ $Y \leftarrow \mathsf{Ev}(F, X)$ **return** (Y, R, tag)	**proc** $\mathsf{De}_1(d_1, Y_1)$ $(Y, R, \mathrm{tag}) \leftarrow Y_1, \ \ (D, K) \leftarrow d_1$ $d \leftarrow D \oplus \textsc{Hash}(\lvert D\rvert, 1 \parallel R)$ **if** $\textsc{Hash}(\lvert K\rvert, K \parallel R) \neq \mathrm{tag}$ **then** **return** $\perp$ **return** $\mathsf{De}(d, Y)$

Figure 5.3.2: **Transform all-to-all1 (top):** Scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{obv1}, \Phi) \cap \mathsf{GS}(\mathrm{aut1})$ obtained from scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi) \cap \mathsf{GS}(\mathrm{obv}, \Phi) \cap \mathsf{GS}(\mathrm{aut})$. The transform uses a PRF $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$. **Transform rom-all-to-all1 (bottom):** Garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{obv1}, \Phi) \cap \mathsf{GS}(\mathrm{aut1})$ obtained by applying the ROM rom-all-to-all1 transform to garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi) \cap \mathsf{GS}(\mathrm{obv}, \Phi) \cap \mathsf{GS}(\mathrm{aut})$. It makes use of an RO-modeled Hash. The advantage of the bottom transform over the top one is that it preserves short garbled inputs.

Fig. 5.2.3. We claim it has additional features captured by the following, whose proof is in Section 5.6.6.

Theorem 5.3.2. (1) For any side-information function Φ and any $\mathrm{xxx} \in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{xxx1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then $\mathsf{all1\text{-}to\text{-}all2}[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{xxx2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ (2) If $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{aut1}) \cap \mathsf{GS}(\mathrm{proj})$ then $\mathsf{all1\text{-}to\text{-}all2}[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{aut2}) \cap \mathsf{GS}(\mathrm{proj})$.

5.3.3 Efficient ROM transforms

Again, the all-to-all1 transform does not preserve short garbled inputs. We give the transform rom-all-to-all1 in the ROM to fill the gap. The same attack to break the aut1 security of all-to-all1 can be used to show that rom-prv-to-prv1 is inadequate to handle authenticity as well. The rom-all-to-all1 transform at the bottom of Fig. 5.3.2 generates the mask of the all-to-all1 transform by applying the RO to a random k-bit seed R, and includes R in the encoding rule and garbled input and output in place of the full mask, thereby saving space. As a consequence, it preserves short garbled inputs. Instead of using a PRF $\mathsf{F}_K : \{0,1\}^* \to \{0,1\}^k$, we call $\textsc{Hash}(k, K \parallel \cdot)$. For each xxx $\in \{\text{obv}, \text{obv1}, \text{obv2}\}$, let $\mathsf{GS}^{\text{rom}}(\text{xxx}, \Phi)$ be the set of all garbling schemes that are xxx secure over Φ in the ROM. Likewise, for each xxx $\in \{\text{aut}, \text{aut1}, \text{aut2}\}$, let $\mathsf{GS}^{\text{rom}}(\text{xxx})$ be the set of all garbling schemes that are xxx secure in the ROM. We claim:

Theorem 5.3.3. (1) For any side-information function Φ and any xxx $\in \{\text{prv}, \text{obv}\}$, if $\mathcal{G} \in \mathsf{GS}(\text{xxx}, \Phi)$ then rom-all-to-all1$[\mathcal{G}] \in \mathsf{GS}^{\text{rom}}(\text{xxx1}, \Phi)$, and (2) If $\mathcal{G} \in \mathsf{GS}(\text{aut})$ then rom-all-to-all1$[\mathcal{G}] \in \mathsf{GS}^{\text{rom}}(\text{aut})$.

The proof is in Section 5.6.7. We note that the starting scheme is not assumed projective, but a suitable re-interpretation of the notation is enough to ensure that if the starting scheme is projective, so is the constructed one.

The ROM transform to promote coarse-grained to to fine-grained security is unchanged; we let rom-all1-to-all2 = rom-prv1-to-prv2 be the transform at the bottom of Fig. 5.2.4. We claim the following theorem; the proof is in Section 5.6.8

Theorem 5.3.4. (1) For any Φ and any xxx $\in \{\text{prv}, \text{obv}\}$: If $\mathcal{G}_1 \in \mathsf{GS}(\text{xxx1}, \Phi) \cap \mathsf{GS}(\text{proj})$ then scheme rom-all1-to-all2$[\mathcal{G}_1] \in \mathsf{GS}^{\text{rom}}(\text{xxx2}, \Phi) \cap \mathsf{GS}(\text{proj})$, and (2) If $\mathcal{G}_1 \in \mathsf{GS}^{\text{rom}}(\text{aut1}) \cap \mathsf{GS}(\text{proj})$ then scheme rom-all1-to-all2$[\mathcal{G}_1] \in \mathsf{GS}^{\text{rom}}(\text{aut2}) \cap \mathsf{GS}(\text{proj})$.

5.3.4 Application to secure outsourcing

DEFINITIONS. An outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$ is a tuple of PT algorithms that, intuitively, will be run partly on a *client* and partly on a *server*. Generation algorithm Gen is run by the client on input of the unary encoding 1^k and a string f describing the function $\mathsf{ev}(f, \cdot) : \{0,1\}^{f.n} \to \{0,1\}^{f.m}$ to be evaluated (so that ev, like in a garbling scheme, is a deterministic evaluation algorithm) to get back a public key pk that is sent to the server and a secret key sk that is kept by the client. Algorithm Inp is run by the client on input pk, sk and $x \in \{0,1\}^{f.n}$ to return a garbled input X that is sent to the server. Associated state information St is preserved by the client. Algorithm Comp is run by the server on input pk, X to get a garbled output Y that is returned to the client. The latter runs deterministic algorithm Out on pk, sk, Y, St to get back $y \in \{0,1\}^{f.n} \cup \{\perp\}$. Correctness requires that for all $k \in \mathbb{N}$, all $f \in \{0,1\}^*$, and all $x \in \{0,1\}^{f.n}$, if $(pk, sk) \leftarrow \mathsf{Gen}(1^k, f)$, $(X, St) \leftarrow \mathsf{Inp}(pk, sk, x)$, $Y \leftarrow \mathsf{Comp}(pk, X)$, and $y \leftarrow \mathsf{Out}(pk, sk, Y, St)$, then $y = \mathsf{ev}(f, x)$. Our syntax is the same as that of GGP [37] except for distinguishing between functions and their descriptions, as represented the addition of ev to the list.

The games OSVF_{Π} and $\mathrm{OSPR}_{\Pi,\Phi,\mathcal{S}_{\mathrm{os}}}$ of Fig. 5.3.3 are used to define *verifiability* and *privacy* of an outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$, where Φ is a side-information function and $\mathcal{S}_{\mathrm{os}}$ is a simulator. In both games, the adversary is allowed only one GETPK query, and this must be its first oracle query. For adversaries $\mathcal{A}_{\mathrm{os}}$ and $\mathcal{B}_{\mathrm{os}}$, we define $\mathbf{Adv}^{\mathrm{osvf}}_{\Pi}(\mathcal{A}_{\mathrm{os}}, k) = \Pr[\mathrm{OSVF}^{\mathcal{A}_{\mathrm{os}}}_{\Pi}(k)]$ and $\mathbf{Adv}^{\mathrm{ospr},\Phi,\mathcal{S}_{\mathrm{os}}}_{\Pi}(\mathcal{B}_{\mathrm{os}}, k) = 2\Pr[\mathrm{OSPR}^{\mathcal{B}_{\mathrm{os}}}_{\Pi,\Phi,\mathcal{S}_{\mathrm{os}}}(k)] - 1$. We say that Π is verifiable if $\mathbf{Adv}^{\mathrm{osvf}}_{\Pi}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible for all PT adversaries $\mathcal{A}_{\mathrm{os}}$. We say that Π is private over Φ if for all PT adversaries $\mathcal{B}_{\mathrm{os}}$ there is a PT simulator $\mathcal{S}_{\mathrm{os}}$ (that maintains state across invocations) such that $\mathbf{Adv}^{\mathrm{ospr},\Phi,\mathcal{S}_{\mathrm{os}}}_{\Pi}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible. An adversary is said to be one-time if it makes only one INPUT query. We say that Π is one-time verifiable if $\mathbf{Adv}^{\mathrm{osvf}}_{\Pi}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible for all PT one-time adversaries $\mathcal{A}_{\mathrm{os}}$. We say that Π is one-time private over Φ if for all PT one-time adversaries $\mathcal{B}_{\mathrm{os}}$ there is a PT simulator $\mathcal{S}_{\mathrm{os}}$ such that $\mathbf{Adv}^{\mathrm{ospr},\Phi,\mathcal{S}_{\mathrm{os}}}_{\Pi}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible.

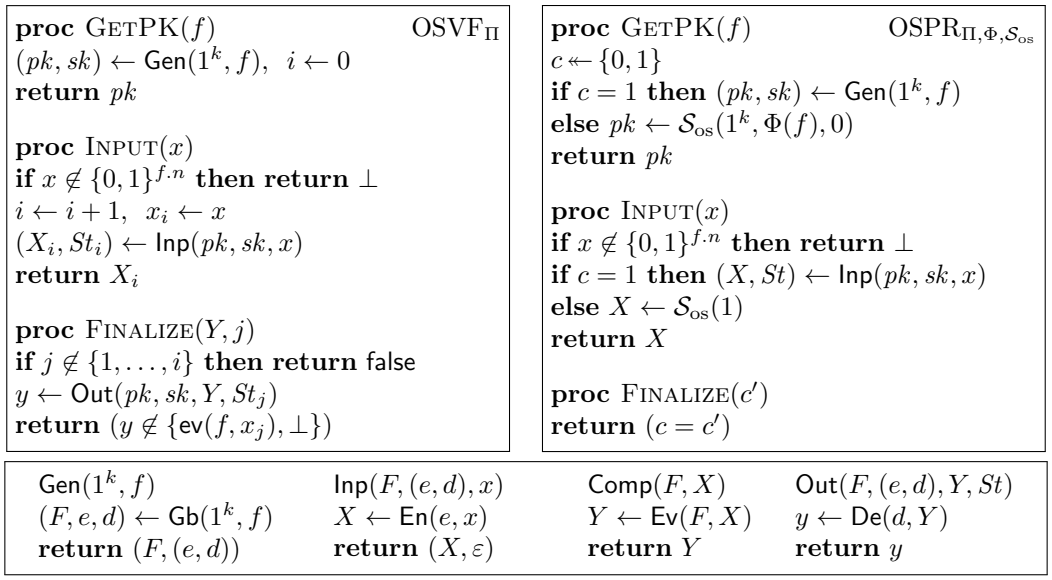

Figure 5.3.3: Games to define the **verifiability** (OSVF) (top left) and **privacy** (OSPR) (top right) of outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$, and the outsourcing scheme $\Pi[\mathcal{G}] = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$ (bottom) constructed from garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$.

Our verifiability definition coincides with that of GGP [37] but our privacy definition is stronger: it requires not just "input privacy" (concealing each input x) but, also, privacy of the function f (relative to Φ). (As in our garbling definitions this is subject to $\Phi(f)$ being revealed). Also, while GGP use an indistinguishability-style formalization, we use a simulation-style one, as this is stronger for some side-information functions.

To be "interesting" the work of the client in an outsourcing scheme should be less than the work required to compute the function directly, for otherwise outsourcing is not buying anything. An outsourcing scheme is said to be non-trivial if this condition is met.

Achieving one-time security. GGP show how to use FHE to turn any one-time verifiable and private outsourcing scheme into a fully verifiable and private one. This allows us to focus on designing the former. We show how a garbling scheme that is both aut1 and obv1 secure immediately implies a one-time verifiable and private outsourcing scheme. The construction, given in Fig. 5.3.3, is very direct, and the proof is trivial. These points reinforce our claim that the garbling scheme abstraction and adaptive security may be easily used in applications.

Theorem 5.3.5. If $\mathcal{G} \in \mathsf{GS}(\text{obv1}, \Phi) \cap \mathsf{GS}(\text{aut1})$ then outsourcing scheme $\Pi[\mathcal{G}]$ is one-time verifiable and also one-time private over Φ.

Proof. Let $\mathcal{A}_{\text{os}}$ be a PT one-time adversary attacking the verifiability of $\Pi[\mathcal{G}]$. We construct another PT adversary $\mathcal{A}_{\text{gs}}$ such that $\mathbf{Adv}^{\text{osvf}}_{\Pi[\mathcal{G}]}(\mathcal{A}_{\text{os}}, k) \leq \mathbf{Adv}^{\text{aut1}}_{\mathcal{G}}(\mathcal{A}_{\text{gs}}, k)$ for all $k \in \mathbb{N}$, which proves the first claim in the theorem. Adversary $\mathcal{A}_{\text{gs}}(1^k)$ runs $\mathcal{A}_{\text{os}}(1^k)$, answering the GETPK query via GARBLE and the (single) INPUT query via INPUT. When $\mathcal{A}_{\text{os}}$ halts with output Y, j, adversary $\mathcal{A}_{\text{gs}}$ outputs Y.

Let $\mathcal{B}_{\text{os}}$ be a PT one-time adversary attacking the privacy of $\Pi[\mathcal{G}]$. We construct another PT adversary $\mathcal{B}_{\text{gs}}$ as follows. Adversary $\mathcal{B}_{\text{gs}}(1^k)$ runs $\mathcal{B}_{\text{os}}(1^k)$, answering the GETPK query via GARBLE and the (single) INPUT query via INPUT. When $\mathcal{B}_{\text{os}}$ halts with output c', adversary $\mathcal{B}_{\text{gs}}$ outputs c'. By the assumption that $\mathcal{G} \in \mathsf{GS}(\text{obv1}, \Phi)$ there is PT simulator $\mathcal{S}_{\text{gs}}$ such that $\mathbf{Adv}^{\text{obv1},\Phi,\mathcal{S}_{\text{gs}}}_{\mathcal{G}}(\mathcal{B}_{\text{gs}}, \cdot)$ is negligible. Let $\mathcal{S}_{\text{os}} \equiv \mathcal{S}_{\text{gs}}$. Then $\mathbf{Adv}^{\text{ospr},\Phi,\mathcal{S}_{\text{os}}}_{\Pi[\mathcal{G}]}(\mathcal{B}_{\text{os}}, k) \leq \mathbf{Adv}^{\text{obv1},\Phi,\mathcal{S}_{\text{gs}}}_{\mathcal{G}}(\mathcal{B}_{\text{gs}}, k)$ for all $k \in \mathbb{N}$, which proves the second claim in the theorem. □

A benefit of our modular approach is that we may use any obv1 + aut1 garbling scheme as a starting point while GGP were tied to the scheme of [70]. However, the latter scheme is not adaptively secure, which brings us to our next point.

DISCUSSION. GGP give a proof that their outsourcing scheme is one-time verifiable assuming the encryption scheme underlying the garbled-circuit construction of Lindell and Pinkas (LP) [70] has semantic security, and elusive and verifiable range. However, their proof has a gap. Quoting [37, p. 12 of Aug 2010 ePrint version]: "For any two values x, x' with $f(x) = f(x')$, the security of Yao's protocol implies that no efficient player P_2 can distinguish if x or x' was used." This claim is correct if both x and x' are chosen independently of the randomness in the garbled circuit. But in their setting, the string x is chosen *after* the adversary sees the garbled circuit, and the security proof given by LP no longer applies.

GGP's proof effectively only shows that the garbled circuit construction of LP is (in our language, if cast as a garbling scheme) aut secure. But we show in Proposition 5.5.5

that aut security does not always imply aut1 security. One may try to give a new proof that the LP garbling scheme satisfies aut1 security. However, this seems to be difficult. Intuitively, an adaptive attack on the garbling scheme allows the adversary to mount a key-revealing selective-opening (SOA-K) attack on the underlying encryption scheme. But SOA-K secure encryption is notoriously hard to achieve [12] and not achieved by standard encryption schemes. The only known way to achieve it is via non-committing encryption [24, 28, 32], which is only possible with keys as long as the total number of bits of message ever encrypted [79], making the outsourcing scheme fail to be non-trivial.

This brings us to another discussion of non-triviality. The obv1 + aut1 secure scheme obtained via our all-to-all1 transform has long garbled inputs, so the one-time verifiable outsourcing scheme yielded by Theorem 5.3.5, while secure, is not non-trivial. Our ROM transforms yield an ROM obv1 + aut1 secure scheme with short garbled inputs and thence a non-trivial one-time outsourcing scheme but the FHE-based method of GGP of lifting it to a many-time scheme does not work in the ROM. Finding a obv1 + aut1 secure garbling scheme with short garbled inputs in the standard model under standard assumptions is an open problem. This means that right now we know of no correct way to instantiate GPP's construction to get a non-trivial and proven secure outsourcing scheme in the standard model, based on standard assumptions. We think Theorem 5.3.5 is still useful because it can be used at any point such a scheme emerges. (Indeed, an ongoing work of Bellare, Hoang, and Keelveedhi [15] gives an obv2/aut2-secure garbling scheme by defining a new assumption on keyed hash functions.) All this again is an indication of the subtleties and hidden challenges underlying adaptive security of garbled circuits that seem to have been overlooked in the literature.

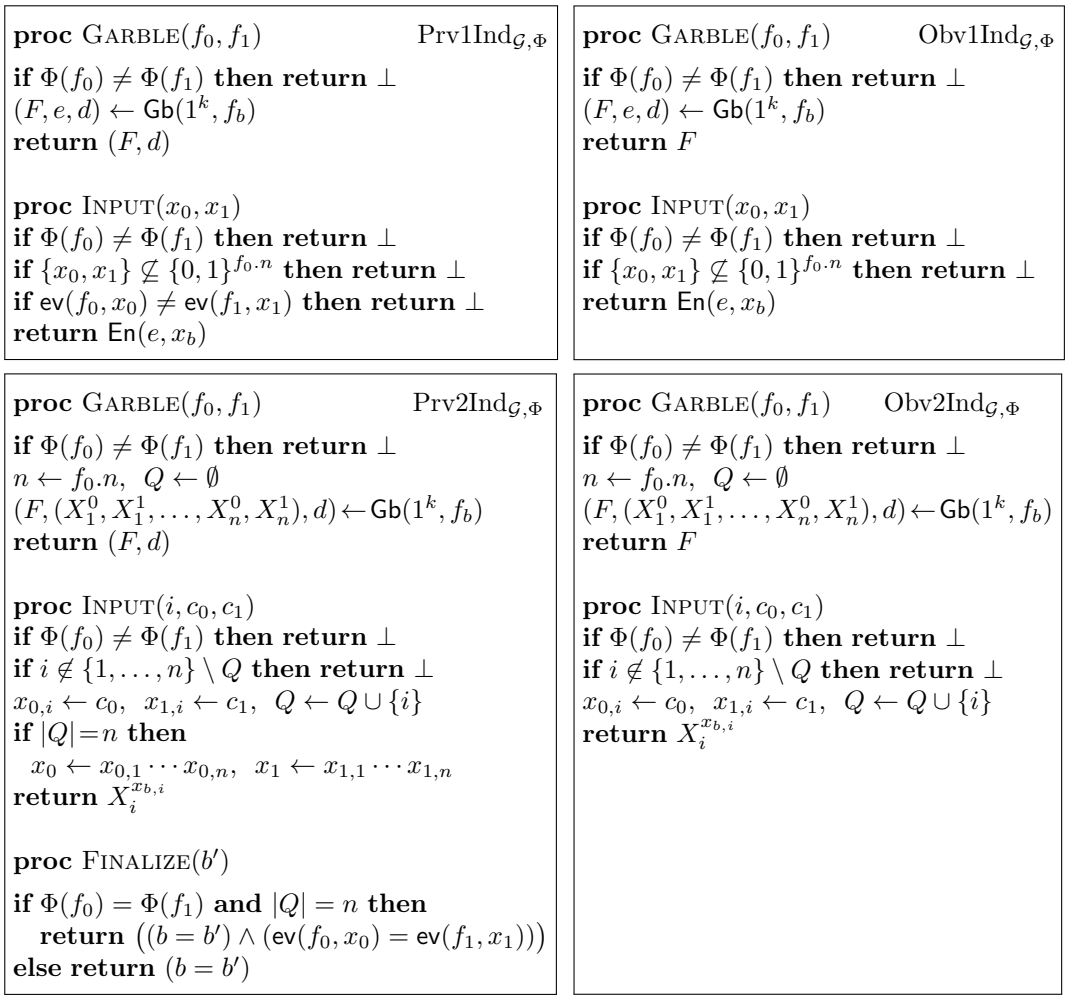

Figure 5.4.1: **Indistinguishability-based privacy notions.** Games to define the ind-based coarse-grained and fine-grained adaptive security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. In each game, INITIALIZE() samples $b \twoheadleftarrow \{0,1\}$, and when FINALIZE($b'$) is unspecified, it returns $(b = b')$.

5.4 Indistinguishability-based definitions

We define the indistinguishability-based counterparts of our prv1, prv2, obv1, and obv2 definitions in Fig. 5.4.1; the prv2.ind and obv2.ind again require garbling schemes to be projective. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme and let Φ be a side-information function. The prv1.ind advantage of an adversary $\mathcal{A}$ is defined by $\mathbf{Adv}^{\text{prv1.ind},\Phi}_{\mathcal{G}}(\mathcal{A}, k) = 2\Pr[\text{Prv1Ind}^{\mathcal{A}}_{\mathcal{G},\Phi}(k)] - 1$. For any xxx $\in \{\text{prv2.ind}, \text{obv1.ind}, \text{obv2.ind}\}$, define $\mathbf{Adv}^{\text{xxx},\Phi}_{\mathcal{G}}(\mathcal{A}, k)$ similarly. We say that $\mathcal{G}$ is xxx secure over Φ if $\mathbf{Adv}^{\text{xxx},\Phi}_{\mathcal{G}}(\mathcal{A}, k)$ is negligible, for every PT adversary $\mathcal{A}$. Let $\mathsf{GS}(\text{xxx}, \Phi)$ be the set of all garbling schemes that are xxx se-

cure over Φ. Below, we will explore the relations between ind-based and sim-based notions, as illustrated in Fig. 5.4.2. It is obvious that prv2.ind $\Rightarrow$ prv1.ind $\Rightarrow$ prv.ind and obv2.ind $\Rightarrow$ obv1.ind $\Rightarrow$ obv.ind.

DISCUSSION. Defining prv2.ind requires care, and merits some discussion. Consider a natural variant prv2.ind.bad in which procedure FINALIZE(b') of game $\text{Prv2Ind}_{\mathcal{G},\Phi}$ returns

$$(b = b') \wedge (\Phi(f_0) = \Phi(f_1)) \wedge (|Q| = n) \wedge ((\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)),$$

requiring the adversary to fully specify its input strings x_0 and x_1 and get no credit if it only gives, say, the first bits of x_0 and x_1, and makes its guess. Doing so would severely limit the adversary's choice of querying (i, c_0, c_1) to the INPUT oracle, because it needs to make sure that the bits c_0 and c_1 can end up making strings x_0 and x_1 satisfying $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. In contrast, for prv2.ind security, if the adversary does not fully specify x_0 and x_1 then the bits c_0 and c_1 can be arbitrary, and the adversary will not be "giving up" on the game.

We now show that in fact, prv2.ind.bad is "wrong", insofar as it doesn't imply prv1.ind. Fix a length-preserving permutation $P : \{0,1\}^* \to \{0,1\}^*$ that is *one-way*: for every PT adversary $\mathcal{A}$, the advantage

$$\mathbf{Adv}^{\text{ow}}_{P}(\mathcal{A}, k) = \Pr[x \leftarrow \{0,1\}^k;\ x' \leftarrow \mathcal{A}(P(x)) \colon\ x' = x]$$

is negligible. For every $f, x \in \{0,1\}^*$, let $\Phi(f) = (f.n, f.m, |f|)$, and let $\mathsf{ev}^P(f, x) = P(b \parallel x)$, where b is the last bit of f. Consider the following projective garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}^P)$. Let $\mathsf{Gb}(1^k, f) = (b, e, \varepsilon)$, where b is the last bit of f, $n = f.n$, and e is the $2n$-bit vector $(0, 1, \ldots, 0, 1)$. Let $\mathsf{En}(e, x) = x$, $\mathsf{Ev}(b, x) = P(b \parallel x)$, and $\mathsf{De}(\varepsilon, y) = y$. Let an (even computationally-unbounded) adversary $\mathcal{A}$ attack the prv2.ind.bad security of $\mathcal{G}$. Assume that $\mathcal{A}$ eventually produces (f_0, f_1, x_0, x_1) satisfying $\Phi(f_0) = \Phi(f_1)$ and $\mathsf{ev}^P(f_0, x_0) = \mathsf{ev}^P(f_1, x_1)$; otherwise $\mathcal{A}$'s advantage is 0. Since P is a permutation, $\mathsf{ev}^P(f_0, x_0) = \mathsf{ev}^P(f_1, x_1)$ implies that $x_0 = x_1$ and the last bits of f_0 and f_1 are equal, and

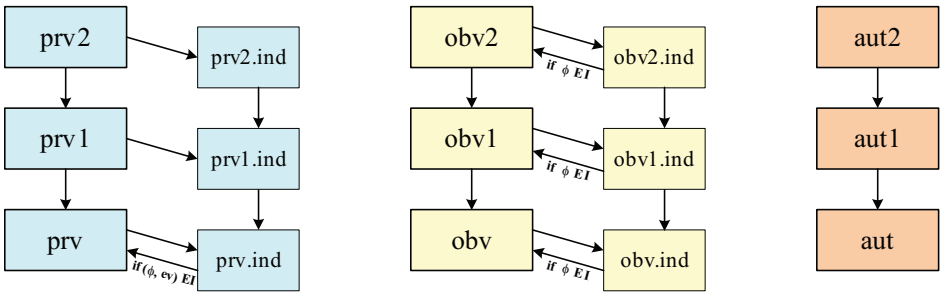

Figure 5.4.2: **Relations among security notions.** A solid arrow is an implication; an if-labeled arrow, a conditional implication. Besides the implications given by the arrows and those inferred from them, any two notions are separated.

consequently, $\mathcal{A}$'s advantage is still 0. On the other hand, consider the following adversary $\mathcal{B}$ attacking the prv1.ind security of $\mathcal{G}$. It queries $(0,1)$ to the GARBLE oracle to receive the answer b. It then outputs b without querying the INPUT oracle, and wins with advantage 1.

RELATIONS AMONG PRIVACY NOTIONS. The following says that, as expected, prv1 security always implies prv1.ind security.

Proposition 5.4.1. $\mathsf{GS}(\text{prv1}, \Phi) \subseteq \mathsf{GS}(\text{prv1.ind}, \Phi)$ for any PT Φ.

Proof. Consider a scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{prv1}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{prv1.ind}, \Phi)$. Let $\mathcal{A}$ be an adversary attacking the prv1.ind security of $\mathcal{G}$ over Φ. We construct a PT prv1-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to GARBLE, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own GARBLE oracle to get back (F, d) and returns this to $\mathcal{A}$. For the next query (x_0, x_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$ or $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ or $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$. Else it queries x_a to its own INPUT oracle to get X and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then for any $\mathcal{S}$ we have

$$\begin{aligned} \Pr\left[\,\text{Prv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 1\,\right] &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\text{prv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \\ \Pr\left[\,\neg\text{Prv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 0\,\right] &= \frac{1}{2} \end{aligned}$$

where b denotes the challenge bit in game $\mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$. The second claim is true since $\mathcal{S}$ has the same input regardless of a, and thus whatever $\mathcal{A}$ receives is independent of a. Subtracting, we obtain

$$\mathbf{Adv}^{\mathrm{prv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \leq 2 \cdot \mathbf{Adv}^{\mathrm{prv1},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k) \ .$$

By assumption there is a PT simulator $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well. □

The following says that prv2 security always implies prv2.ind security.

Proposition 5.4.2. $\mathsf{GS}(\mathrm{prv2},\Phi) \subseteq \mathsf{GS}(\mathrm{prv2.ind},\Phi)$ for any PT Φ.

Proof. Consider a scheme $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}) \in \mathsf{GS}(\mathrm{prv2},\Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv2.ind},\Phi)$. Let $\mathcal{A}$ be an adversary attacking the prv2.ind security of $\mathcal{G}$ over Φ. We construct a PT prv2-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to Garble, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$, and also returns $\perp$ to $\mathcal{A}$'s subsequent Input queries, if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own Garble oracle to get back (F,d) and returns this to $\mathcal{A}$. Then, for each query (i,c_0,c_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i,c_a) to its own Input oracle and returns the resulting token X_i to $\mathcal{A}$. The latter now returns a bit b'. Let x_0 and x_1 be the input strings resulting from the Input queries of $\mathcal{A}$. If $\Phi(f_0) = \Phi(f_1)$ and $\mathcal{A}$ makes its guess without fully specifying x_0 and x_1, then $\mathcal{B}$ returns 1 only if $b' = a$. Otherwise, $\mathcal{B}$ returns 1 only if $b' = a$ and $\mathsf{ev}(f_0,x_0) = \mathsf{ev}(f_1,x_1)$. Then for any $\mathcal{S}$ we have

$$\begin{aligned} \Pr\left[\,\mathrm{Prv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b=1\,\right] &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{prv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \\ \Pr\left[\,\neg\mathrm{Prv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b=0\,\right] &= \frac{1}{2} \end{aligned}$$

where b denotes the challenge bit in game $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$. The second claim is true since $\mathcal{S}$ has the same input regardless of a, and thus whatever $\mathcal{A}$ receives is independent of a. Subtracting,

we see that

$$\mathbf{Adv}^{\mathrm{prv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \le 2 \cdot \mathbf{Adv}^{\mathrm{prv2},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k)\ .$$

By assumption there is a PT simulator $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well. □

Recall that for garbling schemes $(\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})$ with (Φ,ev) efficiently invertible, prv.ind security over Φ implies prv security over Φ. But analogous claims do not hold for adaptive privacy. Below, we will show that, prv2.ind security does not imply prv1 security even when (Φ,ev) is efficiently invertible.

Let $P : \{0,1\}^* \to \{0,1\}^*$ be a length-preserving permutation. Recall that P has a *hard-core predicate* $h : \{0,1\}^* \to \{0,1\}$ if advantage

$$2\Pr[x \twoheadleftarrow \{0,1\}^k;\ b \leftarrow A(P(x)) :\ b = h(x)] - 1$$

is negligible for every PPT adversary $\mathcal{A}$. Starting from any one-way permutation, one can construct another one-way permutation with a hard-core predicate, by the Goldreich-Levin construction [40].

In Proposition 5.4.3, for any string $f \in \{0,1\}^*$, we let $f.n = f.m = |f|$, and let $\mathsf{ev}^*(f,x) = f$, for any $x \in \{0,1\}^{|f|}$. Define $\Phi^*(f) = |f|$ for all $f \in \{0,1\}^*$. We note that (Φ^*,ev^*) is efficiently invertible.

Proposition 5.4.3. $\mathsf{GS}(\mathrm{prv2.ind},\Phi^*) \cap \mathsf{GS}(\mathsf{ev}^*) \not\subseteq \mathsf{GS}(\mathrm{prv1},\Phi^*)$, assuming the existence of a one-way, length-preserving permutation $P : \{0,1\}^* \to \{0,1\}^*$.

Proof. We build a projective scheme $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}^*)$ so that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv2.ind},\Phi^*) \cap \mathsf{GS}(\mathsf{ev}^*)$ but $\mathcal{G} \notin \mathsf{GS}(\mathrm{prv1},\Phi^*)$. Let $h : \{0,1\}^* \to \{0,1\}$ be a hard-core predicate of P. On input $(1^k, f)$, algorithm Gb samples $r_1,\ldots,r_n \twoheadleftarrow \{0,1\}^k$ and creates

$$F = (P(r_1), P(r_2), \ldots, P(r_n), S \oplus f),$$

where $n = |f|$ and $S = h(r_1)\|\cdots\|h(r_n)$. It then picks $S_1,\ldots,S_n \twoheadleftarrow \{0,1\}^{kn}$, lets $e =$

$(S_1, S_1, \ldots, S_n, S_n)$ and $d = (r_1 \| \cdots \| r_n) \oplus S_1 \oplus \cdots \oplus S_n$, and returns (F, e, d). Algorithm En is defined, as the scheme $\mathcal{G}$ is projective. Let Ev be the identity. Finally, on input d and $Y = (F, X)$, algorithm De parses $X = (S_1, \ldots, S_n)$ and $F = (s_1, \ldots, s_n, U)$. It then computes $r_1 \| \cdots \| r_n = d \oplus S_1 \oplus \cdots \oplus S_n$, where $k = |s_1|$ and each string r_i has length k. If $P(r_i) = s_i$ for all $i \leq n$ then it returns $f = U \oplus S$, where $S = h(r_1) \| \cdots \| h(r_n)$, otherwise it returns $\perp$.

Consider an adversary $\mathcal{A}$ attacking the prv2.ind security of $\mathcal{G}$. Without loss of generality, assume that $\mathcal{A}$ queries (f_0, f_1) such that $\Phi^*(f_0) = \Phi^*(f_1)$. If $f_0 = f_1$ then the advantage of $\mathcal{A}$ will be 0, no matter how it queries oracle Input, so assume that $f_0 \neq f_1$. If $\mathcal{A}$ fully specifies its input strings x_0 and x_1 then $\mathsf{ev}^*(f_0, x_0) \neq \mathsf{ev}^*(f_1, x_1)$, and $\mathcal{A}$'s advantage is again 0. Otherwise, if $\mathcal{A}$ does not fully specify its input strings, then the strings S_i it obtains from oracle Input are independent random strings, so $\mathcal{A}$ gains nothing from querying Input. Then, $\mathcal{A}$'s advantage is negligible, since h is a hard-core predicate of P.

Next, consider the following adversary $\mathcal{B}(1^k)$ attacking the prv1 security of $\mathcal{G}$. It chooses $f \twoheadleftarrow \{0,1\}$ and queries f to its Garble oracle to get answer (F, d). It then queries 0 to oracle Input to receive answer X. It returns 1 only if $\mathsf{De}(d, \mathsf{Ev}(F, X)) = f$ and $F = (s, U)$, with $|s| = k$ and $|U| = 1$. If the challenge bit is 1 then $\mathcal{B}$'s guess is always correct. Suppose that the challenge bit is 0. Fix a computationally unbounded simulator $\mathcal{S}$. The simulator does not know if f is 0 or 1 until the last query, and thus f is independent of F and d. Let $F = (s, U)$. Since P is permutation, $r = P^{-1}(s)$ is uniquely defined, and thus $h(r)$ is also independent of f. Then $f \neq \mathsf{De}(d, \mathsf{Ev}(F, X)) = U \oplus h(r)$ with probability $1/2$, no matter how the simulator chooses X. Thus, $\mathcal{B}$'s guess is correct with probability at least $1/2$. Hence $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv1}, \Phi^*, \mathcal{S}}(\mathcal{B}, k) \geq 1/2$. □

Relations among obliviousness notions. Next we consider relations for obliviousness notions. The following says that obv1 security always implies obv1.ind security, and conversely if Φ is efficiently invertible.

Proposition 5.4.4. For any PT Φ: (1) $\mathsf{GS}(\mathrm{obv1}, \Phi) \subseteq \mathsf{GS}(\mathrm{obv1.ind}, \Phi)$, and (2) If Φ is

efficiently invertible then $\mathsf{GS}(\text{obv1.ind}, \Phi) \subseteq \mathsf{GS}(\text{obv1}, \Phi)$.

Proof. For part (1), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{obv1}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{obv1.ind}, \Phi)$. Let $\mathcal{A}$ be an adversary attacking the obv1.ind security of $\mathcal{G}$ over Φ. We construct a PT obv1-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to GARBLE, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own GARBLE oracle to get back F and returns this to $\mathcal{A}$. For the next query (x_0, x_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$ or $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$. Else it queries x_a to its own INPUT oracle to get X and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then for any $\mathcal{S}$ we have

$$\begin{aligned} \Pr\left[\,\text{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 1\,\right] &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\text{obv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \\ \Pr\left[\,\text{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 0\,\right] &= \frac{1}{2} \end{aligned}$$

where b denotes the challenge bit in game $\text{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}$. Subtracting, we see that

$$\mathbf{Adv}^{\text{obv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \leq 2 \cdot \mathbf{Adv}^{\text{obv1},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) \ .$$

By assumption there is a PT simulator $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well.

For part (2), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{obv1.ind}, \Phi)$ and let M be a Φ-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{obv1}, \Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the obv1-security of $\mathcal{G}$ over Φ. We define a simulator $\mathcal{S}$ that on input $(1^k, \phi, 0)$, lets $f \leftarrow M(\phi)$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and returns F. For the next query, the simulator chooses $x \twoheadleftarrow \{0,1\}^{f.n}$ and then answers $X = \mathsf{En}(e, x)$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1 to GARBLE, adversary $\mathcal{A}$ lets $f_0 \leftarrow M(\Phi(f_1))$ and then queries f_0, f_1 to its own GARBLE oracle to get back F, which it returns to $\mathcal{B}$. When receiving x_1 on the

next query, if $x_1 \notin \{0,1\}^{f_1.n}$ then $\mathcal{A}$ returns $\perp$ to $\mathcal{B}$. Else it chooses $x_0 \twoheadleftarrow \{0,1\}^{f_0.n}$, queries (x_0, x_1) to its INPUT oracle, and returns the answer X to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. Then

$$\begin{aligned} \Pr\left[\,\mathrm{Obv1Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b = 1\,\right] &= \Pr\left[\,\mathrm{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c = 1\,\right] \\ \Pr\left[\,\neg\mathrm{Obv1Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b = 0\,\right] &= \Pr\left[\,\neg\mathrm{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c = 0\,\right] \end{aligned}$$

where b and c denote the challenge bit in game $\mathrm{ObvInd}_{\mathcal{G},\Phi}$ and $\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}$ respectively. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv1},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k) \leq \mathbf{Adv}^{\mathrm{obv1.ind},\Phi}_{\mathcal{G}}(\mathcal{A},k)\ .$$

But the RHS is negligible by assumption, hence the LHS is as well. □

The following says that obv2 security always implies obv2.ind security, and conversely if Φ is efficiently invertible.

Proposition 5.4.5. For any PT Φ: (1) $\mathsf{GS}(\mathrm{obv2},\Phi) \subseteq \mathsf{GS}(\mathrm{obv2.ind},\Phi)$, and (2) If Φ is efficiently invertible then $\mathsf{GS}(\mathrm{obv2.ind},\Phi) \subseteq \mathsf{GS}(\mathrm{obv2},\Phi)$.

Proof. For part (1), let $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}) \in \mathsf{GS}(\mathrm{obv2},\Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv2.ind},\Phi)$. Let $\mathcal{A}$ be an adversary attacking the obv2.ind security of $\mathcal{G}$ over Φ. We construct a PT obv2-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to GARBLE, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$, and also returns $\perp$ to $\mathcal{A}$'s subsequent INPUT queries, if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own GARBLE oracle to get back F and returns this to $\mathcal{A}$. Then, for each query (i, c_0, c_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i, c_a) to its own INPUT oracle and returns the resulting token X_i to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then

for any $\mathcal{S}$ we have

$$\begin{aligned} \Pr\left[\,\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 1\,\right] &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{obv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \\ \Pr\left[\,\neg\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b = 0\,\right] &= \frac{1}{2} \end{aligned}$$

where b denotes the challenge bit in game $\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}$. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \le 2 \cdot \mathbf{Adv}^{\mathrm{obv2},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k)\ .$$

By assumption there is a PT simulator $\mathcal{S}$ such that the RHS is negligible. Hence the LHS is negligible as well.

For part (2), let $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}) \in \mathsf{GS}(\mathrm{obv2.ind},\Phi)$ and let M be a Φ-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv2},\Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the obv2 security of $\mathcal{G}$ over Φ. We define a simulator $\mathcal{S}$ that, on input $(1^k,\phi,0)$, lets $f \leftarrow M(\phi)$ then $(F,e,d) \leftarrow \mathsf{Gb}(1^k,f)$, where M is a Φ-inverter, and returns F. For each subsequent query (i,j), the simulator lets $e = (X^0_1,X^1_1,\ldots,X^0_n,X^1_n)$, chooses $x_i \twoheadleftarrow \{0,1\}$, and answers $X^{x_i}_i$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1 to Garble, adversary $\mathcal{A}$ lets $f_0 \leftarrow M(\Phi(f_1))$ and then queries f_0,f_1 to its own Garble oracle to get back F, which it returns to $\mathcal{B}$. Then, for each query (i,c_1) of $\mathcal{B}$, the adversary $\mathcal{A}$ chooses $c_0 \twoheadleftarrow \{0,1\}$ and queries (i,c_0,c_1) to its Input oracle, and returns the resulting token X_i to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. Then

$$\begin{aligned} \Pr\left[\,\mathrm{Obv2Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b = 1\,\right] &= \Pr\left[\,\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c = 1\,\right] \\ \Pr\left[\,\neg\mathrm{Obv2Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b = 0\,\right] &= \Pr\left[\,\neg\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c = 0\,\right] \end{aligned}$$

where b and c denote the challenge bit in game $\mathrm{Obv2Ind}_{\mathcal{G},\Phi}$ and $\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}$ respectively. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv2},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k) \le \mathbf{Adv}^{\mathrm{obv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)\ .$$

But the RHS is negligible by assumption, hence the LHS is as well. □

EQUIVALENCE IN IDEALIZED MODELS. In idealized models, define obv1.prom as obv1 security in which the simulator too has oracle access to the ideal primitives, and obv1.nprom as obv1 security in which the simulator doesn't have oracle access to the ideal primitives, and will itself reply to the oracle queries made by the adversary. Define obv2.prom and obv2.nprom likewise. Then, if Φ is efficiently invertible then (1) obv1.prom and obv1.nprom are equivalent, and (2) obv2.prom and obv2.nprom are equivalent.

For part (1), it suffices to show that obv1.prom security implies obv1.nprom security, since the latter obviously implies the former. By part (1) of Proposition 5.4.4, obv1.prom security implies obv1 security. The proof still holds, even if the simulator $\mathcal{S}$ uses the programmability power to collude with the obv1.prom adversary $\mathcal{B}$ to fool the obv1.ind adversary $\mathcal{A}$, because what $(\mathcal{S}, \mathcal{B})$ receives is independent of $\mathcal{A}$'s challenge bit. Because Φ is efficiently invertible, by part (2) of Proposition 5.4.4, obv1.ind security then implies obv1.nprom security.

For part (2), it suffices to show that obv2.prom security implies obv2.nprom security, since the latter obviously implies the former. By part (1) of Proposition 5.4.5, obv2.prom security implies obv2 security. The proof still holds, even if the simulator $\mathcal{S}$ uses the programmability power to collude with the obv2.prom adversary $\mathcal{B}$ to fool the obv2.ind adversary $\mathcal{A}$, because what $(\mathcal{S}, \mathcal{B})$ receives is independent of $\mathcal{A}$'s challenge bit. Because Φ is efficiently invertible, by part (2) of Proposition 5.4.5, obv2.ind security then implies obv2.nprom security.

5.5 Separations

For each xxx $\in \{\text{prv}, \text{obv}, \text{aut}\}$, it is obvious that xxx2 security implies xxx1 security and that xxx1 security implies xxx security. We want to prove that the converse directions are not true, even for projective schemes. Moreover, there are separations among our notions of privacy, obliviousness, and authenticity. The relations among notions are illustrated in Fig. 5.4.2. Recall that $\mathsf{ev}_{\mathsf{circ}}$ names the canonical circuit-evaluation procedure.

We will use the scheme described by the top box of Fig. 5.5.1 to separate xxx and xxx1 notions. The idea is as follows. We append 0^k to the garbled input (which is harmless), except for the case that x is a "poisoned" point s. There, we instead append a k-bit random string t (which is unlikely to be 0^k). We choose s at random so that a static adversary is unlikely to query $x = s$, but then include s to the garbled function, making it is trivial for an adaptive adversary to choose $x = s$. To make sure that the probability to query $x = s$ (that is 2^{-n}) is negligible in terms of k, we only perform this trick if $n \geq k$. To deal with authenticity, we append the same string t above to d. Procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u); it returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. This creates a loophole for an adversary to win, if it can query (Y, t) for some string Y. However, an aut adversary is unlikely to know t, because t is disclosed only if it queries the poisoned point $x = s$.

To separate xxx1 and xxx2 notions, we will use the scheme described by the bottom box of Fig. 5.5.1. The idea is as follows. We choose a random $(n-1)$-bit string $V = v_1 \cdots v_{n-1}$, and want to "poison" points $x \in \{V \parallel 0, V \parallel 1\}$. In order to do this, we choose $(n-1)$-bit strings V_i^0, V_i^1 for every $i \leq n$, and append V_i^b to token X_i^b. We let $V_n^0 = V$, making it trivial for a projective adaptive adversary to choose a poisoned point, by querying $(n, 0)$ to its INPUT oracle. Since V is random, an xxx1 adversary on the other hand may know V only *after* it already specifies its x, which is too late to query $x \in \{V\|0, V\|1\}$. Of course it can try to guess V, but then its chance of success is only 2^{1-n}. To make sure that the probability above is negligible in terms of k, we only perform this trick if $n > k$. The other strings V_i^b are independently random (so it's harmless to append to the tokens), except that the checksum of $V_1^{v_1}, \ldots, V_{n-1}^{v_{n-1}}$ (the shares corresponding to a poisoned point) is a random string t whose last bit is 0. To deal with authenticity, we append the same string t above to d. Procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u); it returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. This creates a loophole for an adversary to win, if it can query (Y, t) for some string Y. However, an aut1 adversary is unlikely to know t, because t is

disclosed only if it queries a poisoned point $x \in \{V\|0, V\|1\}$.

SEPARATIONS AMONG PRIVACY NOTIONS. The following says that prv security does not imply prv1 security, even for circuit-garbling schemes.

Proposition 5.5.1. $\mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 5.5.1. We claim that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n \geq k$; otherwise $\mathcal{G}'$ will inherit the prv security from $\mathcal{G}$, as it only appends the garbled function and decoding function with independent random strings, and the garbled input with 0^k. Unless $\mathcal{A}$ can query $x = s$, where s is the random string appended to the garbled function, the same argument applies and $\mathcal{G}'$ will again inherit the prv security from $\mathcal{G}$. However, since $s \twoheadleftarrow \{0,1\}^n$, the chance that $x = s$ is $2^{-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the prv1 security of $\mathcal{G}'$. Choose two circuits f_0, f_1 of the same topology such that $f_0.n = k$ and $f_0(x) = f_1(\overline{x})$ for every $x \in \{0,1\}^k$. Pick $b \twoheadleftarrow \{0,1\}$ and query $f = f_b$ to the oracle GARBLE. When receiving answer (F, s), query $x = x_b$ to INPUT to receive (X, u), where $x_0 = \overline{s}$ and $x_1 = s$. Return 1 only if the last bit of u coincides with b. If the challenge bit is 1 then the adversary answers 0 only if $b = 1$ and the last bit of t is 0, which happens with probability 1/4. Otherwise, since the simulator's inputs are independent of b, the chance that the last bit of u is b is exactly 1/2. Hence the adversary wins with advantage 1/4. □

Similarly, the following proposition says that, even for projective circuit-garbling schemes, prv1 security doesn't imply prv2 security.

Proposition 5.5.2. $\mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

proc $\mathsf{Gb}'(1^k, f)$ $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ $t \twoheadleftarrow \{0,1\}^k, \quad s \twoheadleftarrow \{0,1\}^{f.n}$ **return** $\big((F, s), (e, s, t), (d, t)\big)$	**proc** $\mathsf{Ev}'(F', X')$ $(F, s) \leftarrow F', \ \ (X, u) \leftarrow X'$ **return** $\big(\mathsf{Ev}(F, X), \varepsilon\big)$	**proc** $\mathsf{En}'(e', x)$ $(e, s, t) \leftarrow e', \ \ n \leftarrow \lvert s\rvert$ $k \leftarrow \lvert t\rvert, \ \ X \leftarrow \mathsf{En}(e, x)$ **if** $x = s$ **and** $n \geq k$ **then** **return** (X, t) **return** $(X, 0^k)$
proc $\mathsf{Gb}'(1^k, f)$ $n \leftarrow f.n, \ \ (F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$ **for** $i \in \{1, \ldots, n\}$ **do** $V_i^0, \ V_i^1 \twoheadleftarrow \{0,1\}^{n-1}$ $v_1 \cdots v_{n-1} \leftarrow V_n^0, \quad t \twoheadleftarrow \{0,1\}^{n-2}0$ **if** $n > k$ **then** $V_1^{v_1} \leftarrow t \oplus V_2^{v_2} \oplus \cdots \oplus V_{n-1}^{v_{n-1}}$ **else** $t \twoheadleftarrow \{0,1\}^{k-1}$ $e' \leftarrow \big((X_1^0, V_1^0), (X_1^1, V_1^1), \ldots, (X_n^0, V_n^0), (X_n^1, V_n^1)\big)$ **return** $(F, e', (d, t))$		**proc** $\mathsf{En}'(e', x)$ $(T_0^1, T_1^1, \ldots, T_n^0, T_n^1) \leftarrow e'$ $x_1 \cdots x_n \leftarrow x$ **return** $(T_1^{x_1}, \ldots, T_n^{x_n})$ **proc** $\mathsf{Ev}'(F, X')$ $\big((X_1, V_1), \ldots, (X_n, V_n)\big) \leftarrow X'$ $(X_1, \ldots, X_n) \leftarrow X$ **return** $\big(\mathsf{Ev}(F, X), \varepsilon\big)$

Figure 5.5.1: In both schemes (top and bottom), procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u). It returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. **Separation between** xxx **and** xxx1 **notions (top):** Garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ that separates prv and prv1 (and, later, separates obv from obv1 , and aut from aut1). It is built from a circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$. **Separation between** xxx1 **and** xxx2 **notions (bottom):** Garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ that separates prv1 from prv2 (and later separates obv1 from obv2, and aut1 from aut2 too). It is built from a projective circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$.

Proof. By assumption, $\mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 5.5.1. We claim that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n > k$; otherwise $\mathcal{G}'$ will inherit the prv1 security from $\mathcal{G}$, as it only appends tokens and decoding function with independent random strings. Let $V = V_n^0$ be the random string appended into token X_n^0. Unless the adversary queries $x \in \{V\|0, V\|1\}$, the same argument applies and $\mathcal{G}'$ will again inherit the prv1 security from $\mathcal{G}$. However, as $V \twoheadleftarrow \{0,1\}^{n-1}$, the chance that $x \in \{V\|0, V\|1\}$ is $2^{1-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the prv2 security of $\mathcal{G}'$. Choose two circuits f_0, f_1 of the same topology such that $f_0.n = k + 1$ and $f_0(x) = f_1(x \oplus 1^k 0)$ for every $x \in \{0,1\}^{k+1}$. Pick $b \twoheadleftarrow \{0,1\}$ and query $f = f_b$ to the oracle Garble.

Then query $(k+1, 0)$ to INPUT to get answer (X_{k+1}, V_{k+1}). Let $V_{k+1} = v_1 \cdots v_k$. If $b = 1$ then query $(1, v_1), \ldots, (k, v_k)$ to INPUT. Else query $(1, \overline{v}_1), \ldots, (k, \overline{v}_k)$. Let the answers be $(X_1, V_1), \ldots, (X_k, V_k)$, and let $t = V_1 \oplus \cdots \oplus V_k$. Answer 1 only if the last bit of t is $\overline{b}$. If the challenge bit is 1 then the chance that $\mathcal{A}$ answers 1 is 3/4. If the challenge bit is 0, since the simulator's inputs are independent of b, the chance that the adversary answers 1 is 1/2. Hence $\mathcal{A}$ wins with advantage 1/4. □

SEPARATIONS AMONG OBLIVIOUSNESS NOTIONS. The following says that obv security does not imply obv1 security, even for circuit-garbling schemes.

Proposition 5.5.3. $\mathsf{GS}(\mathrm{obv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{obv1}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{obv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 5.5.1. Following exactly the same security proof and attack in the proof of Proposition 5.5.1, we have $\mathcal{G}' \in \mathsf{GS}(\mathrm{obv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{obv1}, \Phi_{\mathrm{topo}})$. □

Similarly, the following proposition says that, for projective circuit-garbling schemes, obv1 security doesn't imply obv2 security.

Proposition 5.5.4. $\mathsf{GS}(\mathrm{obv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{obv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 5.5.1. Following exactly the same security proof and attack in the proof of Proposition 5.5.2, we have $\mathcal{G}' \in \mathsf{GS}(\mathrm{obv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}})$. □

SEPARATIONS AMONG AUTHENTICITY NOTIONS. The following says that aut security does not imply aut1 security, even for circuit-garbling schemes.

Proposition 5.5.5. $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{aut1})$, assuming that the LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 5.5.1. We claim that $\mathcal{G}' \in \mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut1})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Let t be the random string that is appended to the decoding function. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n \geq k$; otherwise $\mathcal{G}'$ will inherit the aut security from $\mathcal{G}$, as it only appends the garbled function with an independent random string, and the garbled input with 0^k, and the chance that adversary can output $Y' = (Y, t)$ is at most 2^{-k}. Unless $\mathcal{A}$ can query $x = s$, where s is the random string appended to the garbled function, the same argument applies and $\mathcal{G}'$ will again inherit the aut security from $\mathcal{G}$. However, since $s \twoheadleftarrow \{0,1\}^n$, the chance that $x = s$ is $2^{-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the aut1 security of $\mathcal{G}'$. Query an arbitrary circuit f, such that $f.n = k$, to GARBLE to receive (F, s). Then, query $x = s$ to INPUT to receive (X, t). Then, output $(1, t)$ and win with advantage 1. □

Similarly, the following proposition says that, for projective circuit-garbling schemes, aut1 security does not imply aut2 security.

Proposition 5.5.6. $\mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{aut2})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 5.5.1. We claim that $\mathcal{G}' \in \mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut2})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Let t be the random string that is appended to the decoding function. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n > k$; otherwise $\mathcal{G}'$ will inherit the aut1 security from $\mathcal{G}$, as it only

appends each token with an independent random string, and the chance that the adversary can output $Y' = (Y, t)$ is 2^{1-k}. Let $V = V_n^0$ be the random string appended into token X_n^0. Unless the adversary queries $x \in \{V\|0, V\|1\}$, the same argument applies and $\mathcal{G}'$ will again inherit the aut1 security from $\mathcal{G}$. However, as $V \twoheadleftarrow \{0,1\}^{n-1}$, the chance that $x \in \{V\|0, V\|1\}$ is $2^{1-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the aut2 security of $\mathcal{G}'$. Query an arbitrary circuit f, such that $f.n = k+1$, to GARBLE. Then, query $(k+1, 0)$ to INPUT to receive (X_{k+1}, V_{k+1}). Let $V_{k+1} = v_1 \cdots v_k$. Then, query $(1, v_1), \ldots, (k, v_k)$ to INPUT to receive $(X_1, V_1), \ldots, (X_k, V_k)$ respectively. Let $t = V_1 \oplus \cdots \oplus V_k$. Then, output $(1, t)$ and win with advantage 1. □

SEPARATIONS AMONG PRIVACY, OBLIVIOUSNESS, AND AUTHENTICITY. The following says that privacy does not imply obliviousness, even when we take the strongest form of privacy (projective adaptive) and the weakest form of obliviousness (static).

Proposition 5.5.7. $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{obv}, \Phi)$ for all Φ, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. We construct a scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{obv}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ create $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $((F, d), e, d)$. Let $\mathsf{Ev}'((F, d), X) = \mathsf{Ev}(F, X)$. Including d in the description of the garbled function does not harm prv2 security because an adversary is always given the descriptions of the garbled function and the decoding function simultaneously, so $\mathcal{G}'$ inherits the prv2 security of $\mathcal{G}$. On the other hand, scheme $\mathcal{G}'$ fails to achieve obv. Let $f_0 = f_1 = \mathrm{OR}$, $x_0 = 00$ and $x_1 = 11$. An adversary simply picks $b \twoheadleftarrow \{0,1\}$ and queries (f_b, x_b). On receiving reply $((F, d), X, d)$, it outputs 1 if $\mathsf{De}(d, \mathsf{Ev}(F, X)) = b$ and outputs 0 otherwise. If the challenge bit is 1 then the adversary always answer 1. Otherwise, since the simulator's input is independent of b, the chance that the adversary answers 1

is 1/2. Hence the adversary wins with advantage 1/2. □

The following says that obliviousness does not imply privacy, even when we take the strongest form of obliviousness (projective adaptive) and the weakest form of privacy (static).

Proposition 5.5.8. $\mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. We construct a scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}', \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ create $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $(F, e, (d, e))$. Let $\mathsf{De}'((d, e), Y) = \mathsf{De}(d, Y)$. Including e in the description of the decoding function does not harm obv2 security because an adversary is never given the description of the decoding function, so $\mathcal{G}'$ inherits the obv2 security of $\mathcal{G}$. On the other hand, $\mathcal{G}'$ fails to achieve prv. Let $f_0 = \mathrm{AND}$, $f_1 = \mathrm{OR}$, and $x_0 = x_1 = 11$. An adversary simply chooses $b \twoheadleftarrow \{0, 1\}$ and queries (f_b, x_b). On receiving reply $(F, X, (d, e))$, it outputs 0 if $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, 01))) = 0$ and outputs 1 otherwise. If the challenge bit is 1 then the adversary always answer 1. Otherwise, since the simulator's input is independent of b, the chance that the adversary answers 1 is 1/2. Hence the adversary wins with advantage 1/2. □

The following says that privacy and obliviousness, even in conjunction and in their strongest forms (projective adaptive), do not imply authenticity, even in its weakest form (static).

Proposition 5.5.9. $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{aut})$, for all Φ, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. We construct $\mathcal{G}' = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{aut})$. The construction is as follows. Let $\mathsf{Ev}'(F, X) = \mathsf{Ev}(F, X) \| 0$, and let $\mathsf{De}'(d, Y \| b)$ be $\mathsf{De}(d, Y)$ if $b = 0$, and be 1 otherwise,

where $b \in \{0,1\}$. Appending a constant bit to the garbled output does not harm prv2 security or obv2 security. On the other hand, $\mathcal{G}'$ fails to achieve aut. An adversary simply makes query $(\mathrm{OR}, 00)$ and outputs $1\|1$ to have advantage 1. □

The following says that authenticity, even in its strongest forms (projective adaptive), implies neither privacy nor obliviousness, even in their weakest form (static).

Proposition 5.5.10. $\mathsf{GS}(\mathrm{aut2}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cup \mathsf{GS}(\mathrm{obv}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof. By assumption, $\mathsf{GS}(\mathrm{aut2}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. We construct $\mathcal{G} = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{aut2}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cup \mathsf{GS}(\mathrm{obv}, \Phi_{\mathrm{topo}})$. The construction is as follows. On input $(1^k, f)$, algorithm Gb' creates $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and then outputs $((F, f), e, d)$. On input $((F, f), X)$, algorithm Ev' returns $\mathsf{Ev}(F, X)$. Appending f to F does no harm to authenticity of $\mathcal{G}'$, as the adversary always knows f in its attack. On the other hand, the garbled function leaks f, so both privacy and obliviousness fail over Φ_{topo}. □

5.6 Postponed proofs

The variables specified in simulator code in these proofs are global ones, part of the state that it maintains and updates across its different invocations.

5.6.1 Proof of Theorem 5.2.2

Given any PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$ we build a PT adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$. The assumption of prv security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible. Now we build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$,

$$\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k)\ . \tag{5.6.1}$$

adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim}}(1^k)$
return b'

proc GarbleSim(f)
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\Phi(f))}$
return (F_1, d_1)

proc InputSim(x)
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
$(F, X, d) \leftarrow \text{Garble}(f, x)$
$Z_{\mathtt{F}} \leftarrow F_1 \oplus F$, $Z_{\mathtt{d}} \leftarrow d_1 \oplus d$
return $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$

simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\phi)}$
return (F_1, d_1)

simulator $\mathcal{S}_1(y, 1)$
$(F, X, d) \leftarrow \mathcal{S}(1^k, y, \phi)$
$Z_{\mathtt{F}} \leftarrow F_1 \oplus F$, $Z_{\mathtt{d}} \leftarrow d_1 \oplus d$
return $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$

---

adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\text{GarbleSim},\text{InputSim}}(1^k)$
return b'

proc GarbleSim(f)
$n \leftarrow f.n$, $j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$(F, d) \leftarrow \text{Garble}(f)$
return (F, d)

proc InputSim(i, c)
$x_i \leftarrow c$, $j \leftarrow j + 1$
if $j < n$ **then** $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
  $x \leftarrow x_1 \cdots x_n$, $(X_1, \ldots, X_n) \leftarrow \text{Input}(x)$
  $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
  $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return T_i

simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
$(F, d) \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return (F, d)

simulator $\mathcal{S}_2(\tau_2, i, j)$
if $j < n$ **then** $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
  $y \leftarrow \tau_2$, $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(y, 1)$
  $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
  $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return T_i

Figure 5.6.1: **Top: constructed adversary and simulator for proof of Theorem 5.2.2.** For the first query, return random F_1 and d_1. For the second query, given $y = \mathsf{ev}(f, x)$, produce the real triple (F, X, d), and return X with the one-time pads $F_1 \oplus F$ and $d_1 \oplus d$. **Bottom: constructed adversary and simulator for proof of Theorem 5.2.3.** Except for the last query, return random tokens. For the last query, given $y = \mathsf{ev}(f, x)$, produce the real tokens and create the last piece of secret masks so that the shares unmask the real tokens.

This yields the theorem.

The constructions have to deal with some pesky issues related to the fact that the simulator needs to know the lengths of the pads, so let us settle these first. We know that algorithm Gb runs in polynomial time. This means there are polynomials L'_1, L'_2 such that if $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$ then $|F|$ is at most $L'_1(k, |f|)$ and $|d|$ is at most $L'_2(k, |f|)$. By suitable padding, we assume wlog these lengths are exactly, rather than at most, $L'_1(k, |f|)$ and

$L'_2(k, |f|)$, respectively. (Formally, $\mathcal{G}$ would have to be first transformed to ensure this condition via the suitable padding.) Recall that the side-information $\Phi(f)$ always reveals $f.n$, $f.m$ and $|f|$. This means there are PT functions L_1, L_2 such that $L_1(1^k, \Phi(f)) = L'_1(k, |f|)$ and $L_2(1^k, \Phi(f)) = L'_2(k, |f|)$.

Proceeding now to the constructions, we define $\mathcal{A}$ and $\mathcal{S}_1$ as in the top of Fig. 5.6.1. There, adversary $\mathcal{A}$ runs $\mathcal{A}_1$, simulating the latter's GARBLE and INPUT oracles via procedures GARBLESIM and INPUTSIM, respectively. The last of these invokes the GARBLE oracle from $\mathcal{A}$'s own $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$ game. (The f in the GARBLE call is the one that was earlier queried to GARBLESIM.) The two phases of the simulator are specified separately. Letting b, b_1 be the challenge bits in games $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$ and $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$, respectively, we observe that

$$\begin{aligned}
\Pr\left[\,\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid b = 1\,\right] &= \Pr\left[\,\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b_1 = 1\,\right] \\
\Pr\left[\,\neg\mathrm{PrvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid b = 0\,\right] &= \Pr\left[\,\neg\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b_1 = 0\,\right] .
\end{aligned}$$

Subtracting yields Eq. (5.6.1).

5.6.2 Proof of Theorem 5.2.3

Given any PT adversary $\mathcal{A}_2$ against the prv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$. Now the assumption of prv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, \cdot)$ is negligible. Now we build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2, k) \le \mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, k) \; . \tag{5.6.2}$$

This yields the theorem.

We may assume wlog (again, formally, by first transforming the algorithms of $\mathcal{G}_1$ via suitable padding if necessary) that there is a PT function L' such that if $(F, e, d) \in [\mathsf{Gb}_1(1^k, f)]$ and $e = (X^0_1, X^1_1, \ldots, X^0_{f.n}, X^1_{f.n})$ and $x \in \{0,1\}^{f.n}$ and $X = (X_1, \ldots, X_{f.n}) = \mathsf{En}(e, x)$ and $(\ell, \ell_1, \ldots, \ell_{f.n}) \leftarrow L'(1^k, |f|, f.n)$ then $|X| = \ell$ and $|X^0_i| = |X^1_i| = \ell_i$ for all $1 \le i \le f.n$. Now,

since $\Phi(f)$ is assumed to always reveal $f.n$, $f.m$ and $|f|$, there is a PT function L such that $L(1^k, \Phi(f)) = L'(1^k, |f|, f.n)$.

Proceeding now to the constructions, we define $\mathcal{A}_1$ and $\mathcal{S}_2$ as in the bottom of Fig. 5.6.1. There, adversary $\mathcal{A}_1$ runs $\mathcal{A}_2$, simulating the latter's GARBLE and INPUT oracles via procedures GARBLESIM and INPUTSIM, respectively. These invoke GARBLE and INPUT oracles from $\mathcal{A}_1$'s own $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ game. The first phase of the simulator is specified first, and in the second piece of code, $j \in \{1, \ldots, n\}$. The simulator gets n from ϕ. Letting b_1, b_2 be the challenge bits in games $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ and $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$, respectively, we observe that

$$\begin{aligned}
\Pr\left[\,\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b_1 = 1\,\right] &= \Pr\left[\,\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}(k) \mid b_2 = 1\,\right] \\
\Pr\left[\,\neg\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b_1 = 0\,\right] &= \Pr\left[\,\neg\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}(k) \mid b_2 = 0\,\right] .
\end{aligned}$$

Subtracting yields Eq. (5.6.2).

5.6.3 Proof of Theorem 5.2.4

Given any PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$ we build a PT adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$. The assumption of prv security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible. Now we build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$,

$$\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\mathrm{prv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) + Q(k)/2^k$$

where Q is a polynomial such that $Q(k)$ upper bounds the total number of queries to HASH (made either directly by $\mathcal{A}_1$ or by scheme algorithms) in the execution of game $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ with $\mathcal{A}_1$ on input 1^k. This yields the theorem.

Let L_1, L_2 be as in the proof of Theorem 5.2.2, that is, L_1 and L_2 are PT functions that give the length of the pads masking the garbled function F and decoding function d respectively. The constructions of $\mathcal{A}$ and $\mathcal{S}_1$ are then provided at the top of Fig. 5.6.2, and H is a global variable maintained by the simulator, representing the current state of the

adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim},\text{HashSim}}(1^k)$
return b'

proc GarbleSim(f)
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\Phi(f))}$
return (F_1, d_1)

proc InputSim(x)
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
$(F, X, d) \leftarrow \text{Garble}(f, x)$, $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$, $\mathsf{H}[|d|, 1\|R] \leftarrow d_1 \oplus d$
return (X, R)

proc HashSim(ℓ, w)
if $\mathsf{H}[\ell, w] = \perp$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$

simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\phi)}$
return (F_1, d_1)

simulator $\mathcal{S}_1(y, 1)$
$(F, X, d) \leftarrow \mathcal{S}(1^k, y, \phi)$, $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$
$\mathsf{H}[|d|, 1\|R] \leftarrow d_1 \oplus d$
return (X, R)

simulator $\mathcal{S}_1(\ell, w, \texttt{ro})$
if $\mathsf{H}[\ell, w] = \perp$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$

---

adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\text{GarbleSim},\text{InputSim},\text{HashSim}}(1^k)$
return b'

proc GarbleSim(f)
$n \leftarrow f.n$, $j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$(F, d) \leftarrow \text{Garble}(f)$
return (F, d)

proc InputSim(i, c)
$x_i \leftarrow c$, $j \leftarrow j + 1$, $S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ **then**
  $x \leftarrow x_1 \cdots x_n$, $(X_1, \ldots, X_n) \leftarrow \text{Input}(x)$
  $S \leftarrow S_1 \oplus \cdots \oplus S_n$
  for $t \in \{1, \ldots, n\}$ **do** $\mathsf{H}[\ell_t, 1\|t\|S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return T_i

proc HashSim(r, w)
if $\mathsf{H}[r, w] = \perp$ **then** $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$

simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
$(F, d) \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return (F, d)

simulator $\mathcal{S}_2(\tau_2, i, j)$
$S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ **then**
  $y \leftarrow \tau_2$, $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(y, 1)$
  $S \leftarrow S_1 \oplus \cdots \oplus S_n$
  for $t \in \{1, \ldots, n\}$ **do**
    $\mathsf{H}[\ell_t, 1\|t\|S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return T_i

simulator $\mathcal{S}_2(r, w, \texttt{ro})$
if $\mathsf{H}[r, w] = \perp$ **then** $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$

Figure 5.6.2: **Top: constructed adversary and simulator for proof of Theorem 5.2.4.** For the first query, return random F_1 and d_1. For the second query, given $y = \mathsf{ev}(f, x)$, produce the real triple (F, X, d), choose a random seed $R \twoheadleftarrow \{0,1\}^k$, and program the RO so that the pads $F_1 \oplus F$ and $d_1 \oplus d$ are indeed $\text{Hash}(|F|, 0\|R)$ and $\text{Hash}(|d|, 1\|R)$ respectively. **Bottom: constructed adversary and simulator for proof of Theorem 5.2.5.** Except for the last query, return random tokens. For the last query, given $y = \mathsf{ev}(f, x)$, produce the real tokens, choose a random seed $S \twoheadleftarrow \{0,1\}^k$, and program the RO so that the shares unmask the real tokens.

simulated RO. Let game Hy be identical to $\text{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k)$ with challenge bit $b = 0$, but set a flag *bad* if $\mathcal{A}_1$ can query (ℓ, w) to the random oracle such that R is the suffix of w, prior to receiving R from the garbled input, where R is the seed generating the pads. If game Hy

doesn't set *bad* then it is identical to game $\mathrm{Prv}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k)$ with challenge bit $c = 0$. Then

$$\begin{aligned}\Pr[\mathrm{Prv}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid c = 0] - \Pr[\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b = 0] &\leq \Pr[\mathrm{Hy}^{\mathcal{A}_1}(k) \text{ sets } \mathit{bad}] \\ &\leq Q(k)/2^k \ .\end{aligned}$$

On the other hand, $\Pr[\mathrm{Prv}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid c = 1] = \Pr[\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b = 1]$. Subtracting, we get the claimed bound.

5.6.4 Proof of Theorem 5.2.5

Given any PT adversary $\mathcal{A}_2$ against the prv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$. Now the assumption of prv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1,\cdot)$ is negligible. Now we build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{prv2},\Phi,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2,k) \leq \mathbf{Adv}^{\mathrm{prv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1,k) + Q(k)/2^k \ .$$

where Q is a polynomial such that $Q(k)$ upper bounds the total number of queries to HASH (made either directly by $\mathcal{A}_2$ or by scheme algorithms) in the execution of game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$ with $\mathcal{A}_2$ on input 1^k. This yields the theorem.

Let L be as in the proof of Theorem 5.2.3, that is, L is a PT function that gives the lengths of the pads masking the tokens. The constructions of $\mathcal{A}_1$ and $\mathcal{S}_2$ are then provided at the bottom of Fig. 5.6.2. Let game Hy be identical to game $\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}(k)$ with challenge bit $b = 0$, but sets *bad* if $\mathcal{A}_2$ can query (r,w) to the random oracle such that S is the suffix of w, prior to receiving the entire garbled input, where S is the seed generating the pads. If Hy doesn't set *bad* then it is identical to game $\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k)$ with challenge bit $c = 0$. Then

$$\begin{aligned}\Pr[\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}}(k) \mid c = 0] - \Pr[\mathrm{Prv2}^{\mathcal{A}_2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}(k) \mid b = 0] &\leq \Pr[\mathrm{Hy}^{\mathcal{A}_2}(k) \text{ sets } \mathit{bad}] \\ &\leq Q(k)/2^k \ .\end{aligned}$$

adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim}}(1^k)$
return b'

proc GarbleSim(f)
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return F_1

proc InputSim(x)
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
$(F, X) \leftarrow \text{Garble}(f, x)$, $Z_{\mathtt{F}} \leftarrow F_1 \oplus F$
return $(X, Z_{\mathtt{F}})$

simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$
return F_1

simulator $\mathcal{S}_1(1)$
$(F, X) \leftarrow \mathcal{S}(1^k, \phi)$, $Z_{\mathtt{F}} \leftarrow F_1 \oplus F$
return $(X, Z_{\mathtt{F}})$

adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\text{GarbleSim},\text{InputSim}}(1^k)$
return b'

proc GarbleSim(f)
$n \leftarrow f.n$, $j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$F \leftarrow \text{Garble}(f)$
return F

proc InputSim(i, c)
$x_i \leftarrow c$, $j \leftarrow j + 1$
if $j < n$ **then** $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
  $x \leftarrow x_1 \cdots x_n$, $(X_1, \ldots, X_n) \leftarrow \text{Input}(x)$
  $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
  $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return T_i

simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
$F \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return F

simulator $\mathcal{S}_2(i, j)$
if $j < n$ **then** $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
  $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(1)$
  $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
  $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return T_i

Figure 5.6.3: **Top: constructed adversary and simulator from part (1) of the proof of Theorem 5.3.1.** For the first query, return random F_1. For the second query, produce the real pair (F, X), and return X with the one-time pad $F_1 \oplus F$. **Bottom: constructed adversary and simulator from part (1) of the proof of Theorem 5.3.2.** Except for the last query, return random tokens. For the last query, produce the real tokens and create the last piece of secret masks so that the shares unmask the real tokens.

On the other hand, $\Pr[\text{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}}(k) \mid c = 1] = \Pr[\text{Prv2}^{\mathcal{A}_2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}(k) \mid b = 1]$. Subtracting, we get the claimed bound.

5.6.5 Proof of Theorem 5.3.1

For part (1), by adapting the proof of Theorem 5.2.2, we can show that if $\mathcal{G}$ is obv secure then scheme $\mathcal{G}' = \mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}]$ is obv1 secure. Concretely, given any PT adversary $\mathcal{A}_1$ against

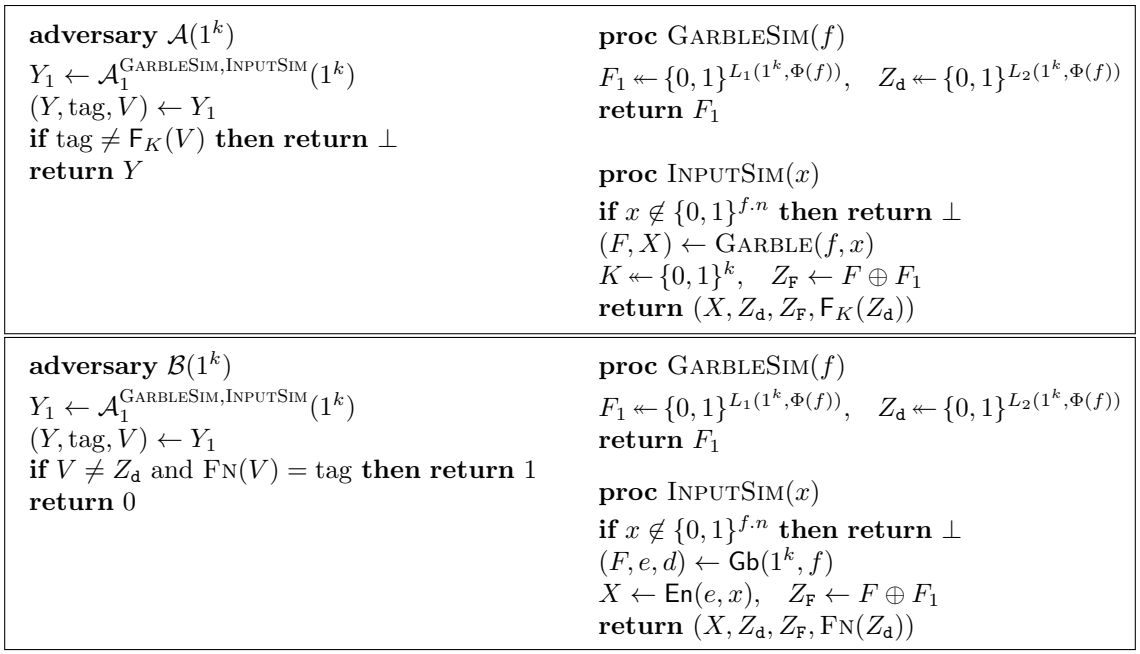

Figure 5.6.4: **Top: constructed adversary $\mathcal{A}$ for part (2) of the proof of Theorem 5.3.1.** Its aut advantage is $\mathbf{Adv}^{\text{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k)$ if $\mathcal{A}_1$ decides to output $V = d'$, as their outputs will be authenticated by the same d. Otherwise, $\mathcal{A}_1$ must forge (V, tag) that bypasses the test $\text{tag} = \mathsf{F}_K(V)$. **Bottom: constructed PRF adversary $\mathcal{B}$ for part (2) of the proof of Theorem 5.3.1.** It feeds $\mathcal{A}_1$ with correct F_1 and X_1. When $\mathcal{A}_1$ outputs $Y_1 = (Y, \text{tag}, V)$, if $V \neq d'$ then $\mathcal{B}$ queries V to its Fn oracle to test if $\text{tag} = \text{Fn}(V)$.

the obv1 security of $\mathcal{G}'$ we build a PT adversary $\mathcal{A}$ against the obv security of $\mathcal{G}$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}^{\text{obv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, \cdot)$ is negligible. We build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\text{obv1},\Phi,\mathcal{S}_1}_{\mathcal{G}'}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\text{obv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) \ .$$

The code of $\mathcal{A}$ and $\mathcal{S}_1$ is shown in the top box of Fig. 5.6.3, with L_1 as in the proof of Theorem 5.2.2, that is, L_1 is a PT function that gives the length of the pad masking the garbled function F. The analysis is analogous to the proof of Theorem 5.2.2.

Now for each xxx $\in \{\text{prv}, \text{obv}\}$, if $\mathcal{G}$ is xxx secure then $\mathcal{G}'$ is xxx1 secure. In scheme $\mathcal{G}'$, the decoding function is $d \oplus Z_{\mathtt{d}}$, and the garble input is $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}})$, whereas in scheme $\mathcal{G}_1 = \mathsf{all\text{-}to\text{-}all1}[\mathcal{G}]$, the former is $(d \oplus Z_{\mathtt{d}}, K)$, and the latter is $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathsf{F}_K(Z_{\mathtt{d}}))$, with $K \twoheadleftarrow \{0,1\}^k$. Scheme $\mathcal{G}_1$ thus can be re-interpreted as scheme $\mathcal{G}'$, with a different encoding of the garbled input and decoding function. Hence $\mathcal{G}_1$ is also xxx1 secure.

proc GARBLESIM(f)	**proc** INPUTSIM(x) Game G_0
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, $K \twoheadleftarrow \{0,1\}^k$	**if** $x \notin \{0,1\}^{f.n}$ **then return** $\bot$
$Z_{\mathtt{F}} \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$	$X \leftarrow \mathsf{En}(e, x)$
$Z_{\mathtt{d}} \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$	**return** $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathsf{F}_K(Z_{\mathtt{d}}))$
$F_1 \leftarrow F \oplus Z_{\mathtt{F}}, \quad d_1 \leftarrow (d \oplus Z_{\mathtt{d}}, K)$	
return F_1	**proc** FINALIZE(d_1, Y_1)
	if $x \notin \{0,1\}^{f.n}$ **then return** 0
	$(D, K) \leftarrow d_1, \quad (Y, \mathrm{tag}, V) \leftarrow Y_1$
	if $\mathrm{tag} \neq \mathsf{F}_K(V)$ **then return** false
	$d \leftarrow D \oplus V$
	return $(\mathsf{De}(d, Y) \neq \bot \wedge Y \neq \mathsf{Ev}(F, X))$

proc GARBLESIM(f)	**proc** INPUT(x) Games $\mathrm{G}_1/\mathrm{G}_2$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$	**if** $x \notin \{0,1\}^{f.n}$ **then return** $\bot$
$Z_{\mathtt{d}} \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$	$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f), \quad X \leftarrow \mathsf{En}(e, x)$
return F_1	$Z_{\mathtt{F}} \leftarrow F \oplus F_1, \quad K \twoheadleftarrow \{0,1\}^k, \quad d_1 \leftarrow (d \oplus Z_{\mathtt{d}}, K)$
	return $(X, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathsf{F}_K(Z_{\mathtt{d}}))$
	proc FINALIZE(d_1, Y_1)
	if $x \notin \{0,1\}^{f.n}$ **then return** 0
	$(D, K) \leftarrow d_1, \quad (Y, \mathrm{tag}, V) \leftarrow Y_1$
	if $\mathrm{tag} \neq \mathsf{F}_K(V)$ **then return** false
	if $Z_{\mathtt{d}} \neq V$ **then** $bad \leftarrow \mathsf{true}, \ V \leftarrow d'$ $\longleftarrow$ Use in G_2
	$d \leftarrow D \oplus V$
	return $(\mathsf{De}(d, Y) \neq \bot \wedge Y \neq \mathsf{Ev}(F, X))$

Figure 5.6.5: **Games used in part (2) of the proof of Theorem 5.3.1.** In procedure FINALIZE of game G_2, we make sure that V must be $d' = d \oplus D$ before we do the assignment $d \leftarrow D \oplus V$, and thus keep d unchanged.

For part (2), fix an adversary $\mathcal{A}_1$. We claim that there are adversaries $\mathcal{A}$ and $\mathcal{B}$ such that

$$\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq 2^{-k} + \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}, k) \ .$$

Moreover, the running time of $\mathcal{A}$ is at most that of $\mathcal{A}_1$ plus the time to garble $\mathcal{A}_1$'s queries, and so is the running time of $\mathcal{B}$. Let L_1 and L_2 be as in the proof of Theorem 5.2.2, that is, L_1 and L_2 are PT functions that give the length of the pads masking the garbled function F and decoding function d respectively. Consider the games $\mathrm{G}_0 - \mathrm{G}_2$ in Fig. 5.6.5. In each game, $\mathcal{A}_1$ has oracle access to procedure GARBLESIM to get the garbled function and decoding function, and to procedure INPUTSIM to get the garbled input. Game G_0 corresponds to game $\mathrm{Aut1}_{\mathcal{G}_1}$.

We explain the game chain up until the terminal game. $\triangleright$ $\mathrm{G}_0 \rightarrow \mathrm{G}_1$: we use the technique of "swapping dependent and independent variables". Namely, instead of sampling $Z_{\mathtt{F}}$ and

then computing $F_1 \leftarrow F \oplus Z_{\mathtt{F}}$, we sample F_1 and then let $Z_{\mathtt{F}} \leftarrow F \oplus F_1$. Then, we can move the garbling of f and the construction of K, d_1 to procedure INPUT, as the outputs of those commands are not used until then. The transition is conservative. Hence $\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) = \Pr[\mathrm{G}_0^{\mathcal{A}_1}(k)] = \Pr[\mathrm{G}_1^{\mathcal{A}_1}(k)]$. $\triangleright$ $\mathrm{G}_1 \rightarrow \mathrm{G}_2$: in game G_1 we use $\mathcal{A}_1$'s output to recover the decoding function d, but in game G_2 we retrieve the correct d from memory. The two games are identical until G_2 sets *bad*.

Adversary $\mathcal{A}(1^k)$ runs $\mathcal{A}_1(1^k)$. When the latter makes queries, the former replies via the code of the top box of Fig. 5.6.4. Then, $\mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) = \Pr[\mathrm{G}_2^{\mathcal{A}_1}(k)]$, since the decoding function to authenticate $\mathcal{A}$'s output is the *correct* one instead of the one recovered from $\mathcal{A}_1$'s output. Adversary $\mathcal{B}(1^k)$ has an oracle FN that implements either $\mathsf{F}_K(\cdot)$ or a truly random function. It runs $\mathcal{A}_1(1^k)$, and follows the code of the bottom box of Fig. 5.6.4. If the challenge bit b of the PRF game is 0, that is, the oracle FN implements a truly random function, then $\mathcal{B}$ will answer 1 with probability 2^{-k}. On the other hand, if $b = 1$ then $\mathcal{B}$ answers correctly if only if $\mathcal{A}_1$ can forge a pair (V, tag) that bypasses the test $\mathsf{F}_K(V) = \mathrm{tag}$. This, in other words, means that $\mathcal{A}$ can set the flag *bad* in game G_2 to be true. Then,

$$\begin{aligned}\Pr[\mathrm{PRF}^{\mathcal{B}}_{\mathsf{F}}(k) \mid b = 1] &= \Pr[\mathsf{BAD}(\mathrm{G}_2^{\mathcal{A}_1}(k))] \\ &\geq \Pr[\mathrm{G}_2^{\mathcal{A}_1}(k)] - \Pr[\mathrm{G}_1^{\mathcal{A}_1}(k)] = \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) - \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) \ .\end{aligned}$$

Hence $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}, k) \geq \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) - \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) - 2^{-k}$, as claimed.

For part (3), if the encoding function e of $\mathcal{G}$ encodes $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ then the encoding function e_1 of $\mathcal{G}_1$ can be re-interpreted as $T_i^b = X_i^b$ if $i < n$, and $(T_1^0, T_1^1, \ldots, T_n^0, T_n^1)$, where $T_i^b = (X_i^b, Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathsf{F}_K(Z_{\mathtt{d}}))$ if $i = n$, for $b \in \{0, 1\}$. Then, for $x = x_1 \cdots x_n$, the garbled input of $\mathcal{G}_1$ is

$$\big((X_1^{x_1}, \ldots, X_n^{x_n}), Z_{\mathtt{d}}, Z_{\mathtt{F}}, \mathsf{F}_K(Z_{\mathtt{d}})\big) = (T_1^{x_1}, \ldots, T_n^{x_n}),$$

and the scheme $\mathcal{G}_1$ is therefore projective.

5.6.6 Proof of Theorem 5.3.2

Let $\mathcal{G}_2 = \mathsf{all1\text{-}to\text{-}all2}[\mathcal{G}_1]$. For part (1), it suffices to give the proof for the obliviousness case. The proof is similar to that of Theorem 5.2.3. Given any PT adversary $\mathcal{A}_2$ against the obv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}_1$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}^{\mathrm{obv1},\,\Phi,\,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1,\cdot)$ is negligible. We build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{obv2},\,\Phi,\,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2,k) \leq \mathbf{Adv}^{\mathrm{obv1},\,\Phi,\,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1,k)\ . \tag{5.6.3}$$

The code of $\mathcal{A}_1$ and $\mathcal{S}_2$ is given in the bottom box of Fig. 5.6.3, with L as in the proof of Theorem 5.2.3 (that is, L is a PT function that gives the length of the pads masking the tokens).

For part (2), we reuse the procedures GarbleSim and InputSim in part (1). Let $\mathcal{A}_2$ attack the aut2 security of $\mathcal{G}_2$. Adversary $\mathcal{A}_1(1^k)$ runs $\mathcal{A}_2(1^k)$, simulating the latter's Garble and Input oracles via procedures GarbleSim and InputSim respectively. When $\mathcal{A}_2$ outputs Y, adversary $\mathcal{A}_1$ also outputs Y. Let X and T be the garbled input given to $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Then

$$\begin{aligned}\mathbf{Adv}^{\mathrm{aut2}}_{\mathcal{G}_2}(\mathcal{A}_2,k) &= \Pr[Y \neq \mathsf{Ev}_2(F,T) \wedge \mathsf{De}_1(d,Y) \neq \bot] \\ &= \Pr[Y \neq \mathsf{Ev}_1(F,X) \wedge \mathsf{De}_1(d,Y) \neq \bot] = \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1,k);\end{aligned}$$

the second equality holds because $\mathsf{Ev}_2(F,T) = \mathsf{Ev}_1(F,X)$.

5.6.7 Proof of Theorem 5.3.3

For part (1), by adapting the proof of Theorem 5.2.4, we can show that if $\mathcal{G}$ is obv secure then scheme $\mathcal{G}' = \mathsf{rom\text{-}all\text{-}to\text{-}all1}[\mathcal{G}]$ is obv1 secure. Concretely, given any PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}'$ we build a PT adversary $\mathcal{A}$ against the obv security of $\mathcal{G}$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}^{\mathrm{obv},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{A},\cdot)$

adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim},\text{HashSim}}(1^k)$
return b'

proc GarbleSim(f)
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return F_1

proc InputSim(x)
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
$(F, X) \leftarrow \text{Garble}(f, x)$, $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$
return (X, R)

proc HashSim(ℓ, w)
if $\mathsf{H}[\ell, w] = \perp$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$

simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$
return F_1

simulator $\mathcal{S}_1(1)$
$(F, X) \leftarrow \mathcal{S}(1^k, \phi)$, $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$
return (X, R)

simulator $\mathcal{S}_1(\ell, w, \texttt{ro})$
if $\mathsf{H}[\ell, w] = \perp$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$

---

adversary $\mathcal{A}(1^k)$
$Y_1 \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim},\text{HashSim}}(1^k)$
$(Y, R', \text{tag}) \leftarrow Y_1$
if tag $\neq$ HashSim$(k, K \parallel R')$ **then**
  return $\perp$
return Y

proc GarbleSim(f)
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return F_1

proc InputSim(x)
if $x \notin \{0,1\}^{f.n}$ **then return** $\perp$
$(F, X) \leftarrow \text{Garble}(f, x)$, $R \twoheadleftarrow \{0,1\}^k$, $K \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$, tag $\leftarrow$ HashSim$[k, K \parallel R]$
return (X, R, tag)

proc HashSim(ℓ, w)
if $\mathsf{H}[\ell, w] = \perp$ **then** $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$

Figure 5.6.6: **Top: constructed adversary and simulator used in part (1) of the proof of Theorem 5.3.3.** For the first query, return random F_1. For the second query, produce the real pair (F, X), choose a random seed $R \twoheadleftarrow \{0,1\}^k$, and program the RO so that the pad $F_1 \oplus F$ is indeed Hash$(|F|, 0\|R)$. **Bottom: constructed adversary for part (2) of the proof of Theorem 5.3.3.** When $\mathcal{A}_1$ outputs $Y_1 = (Y, R', \text{tag})$, perform the test tag $\neq$ HashSim$(k, K \parallel R')$ as in algorithm De_1 of $\mathcal{G}_1$, and output Y if this test is passed.

is negligible. We build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\text{obv1},\Phi,\mathcal{S}_1}_{\mathcal{G}'}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\text{obv},\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) + Q(k)/2^k, \tag{5.6.4}$$

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_1$ or by the scheme algorithms) in the execution of game $\text{Obv1}_{\mathcal{G}',\Phi,\mathcal{S}_1}$ with $\mathcal{A}_1$ on input 1^k. The code of $\mathcal{A}$ and $\mathcal{S}_1$ is shown in the top box of Fig. 5.6.6, with L_1 as in the proof of Theorem 5.2.2 (that is, L_1 is a PT function that gives the length of the pad masking the garbled function F). The analysis is analogous to the proof of Theorem 5.2.4.

Now for each xxx $\in \{\text{prv}, \text{obv}\}$, if $\mathcal{G}$ is xxx secure then $\mathcal{G}'$ is xxx1 secure. In scheme $\mathcal{G}'$, the decoding function is d_1, and the garble input is (X, R), whereas in scheme $\mathcal{G}_1$ =

rom-all-to-all1$[\mathcal{G}]$, the former is (d_1, K), and the latter is $(X, R, \textsc{Hash}(k, K \parallel R))$, with $K \twoheadleftarrow \{0,1\}^k$. Scheme $\mathcal{G}_1$ thus can be re-interpreted as scheme $\mathcal{G}'$, with a different encoding of the garbled input and decoding function. Hence $\mathcal{G}_1$ is also xxx1 secure.

For part (2), given any PT adversary $\mathcal{A}_1$ against the aut1 security of $\mathcal{G}_1$, we build a PT adversary $\mathcal{A}$ against the aut security of $\mathcal{G}$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) + (2Q(k)+1)/2^k,$$

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_1$ or by the scheme algorithms) in the execution of game $\mathrm{Aut1}_{\mathcal{G}_1}$ with $\mathcal{A}_1$ on input 1^k. The code of the adversary $\mathcal{A}$ is given in the bottom of Fig. 5.6.6. Let Bad be the even that $\mathcal{A}$ can query (ℓ, w) to the random oracle such that either (i) R is a suffix of w, and this query is made prior to receiving R from the garbled input, or (ii) K is a prefix of w. Then $\Pr[\mathrm{Bad}] \leq 2Q(k)/2^k$. Suppose than Bad does not happen. Let $Y_1 = (Y, R', \mathrm{tag})$ be the output of $\mathcal{A}_1$. If $R' \neq R$ then the chance that tag $= \textsc{HashSim}(k, K \parallel R')$ is at most 2^{-k}. If $R = R'$ then $\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) = \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k)$. Hence, totally, $\mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) + \Pr[\mathrm{Bad}] + 2^{-k} \leq \mathbf{Adv}^{\mathrm{aut}}_{\mathcal{G}}(\mathcal{A}, k) + (2Q(k)+1)/2^k$.

5.6.8 Proof of Theorem 5.3.4

Let $\mathcal{G}_2 =$ rom-all1-to-all2$[\mathcal{G}_1]$. For part (1), it suffices to give the proof for the obliviousness case. The proof is similar to that of Theorem 5.2.5. Given any PT adversary $\mathcal{A}_2$ against the obv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}_1$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}^{\mathrm{obv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, \cdot)$ is negligible. We then build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}^{\mathrm{obv2},\Phi,\mathcal{S}_2}_{\mathcal{G}_2}(\mathcal{A}_2, k) \leq \mathbf{Adv}^{\mathrm{obv1},\Phi,\mathcal{S}_1}_{\mathcal{G}_1}(\mathcal{A}_1, k) + Q(k)/2^k,$$

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_2$ or by the scheme algorithms) in the execution of game $\mathrm{Obv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$ with $\mathcal{A}_2$ on input 1^k. The code of $\mathcal{A}_1$ and $\mathcal{S}_2$ is given in the bottom box of Fig. 5.6.6, with L

adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\text{GarbleSim},\text{InputSim},\text{HashSim}}(1^k)$
return b'

proc GarbleSim(f)
$n \leftarrow f.n,\ \ j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$F \leftarrow \text{Garble}(f)$
return F

proc InputSim(i, c)
$x_i \leftarrow c,\ \ j \leftarrow j+1,\ \ S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ **then**
  $x \leftarrow x_1 \cdots x_n,\ \ (X_1, \ldots, X_n) \leftarrow \text{Input}(x)$
  $S \leftarrow S_1 \oplus \cdots \oplus S_n$
  for $t \in \{1, \ldots, n\}$ **do** $\mathsf{H}[\ell_t, 1 \| t \| S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return T_i

proc HashSim(r, w)
if $\mathsf{H}[r, w] = \perp$ **then** $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$

simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
$F \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ **do** $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return F

simulator $\mathcal{S}_2(i, j)$
$S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ **then**
  $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(1)$
  $S \leftarrow S_1 \oplus \cdots \oplus S_n$
  for $t \in \{1, \ldots, n\}$ **do**
    $\mathsf{H}[\ell_t, 1 \| t \| S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return T_i

simulator $\mathcal{S}_2(r, w, \texttt{ro})$
if $\mathsf{H}[r, w] = \perp$ **then** $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$

Figure 5.6.7: **Constructed adversary and simulator used in part (1) of the proof of Theorem 5.3.4**. Except for the last query, return random tokens. For the last query, produce the real tokens, choose a random seed $S \twoheadleftarrow \{0,1\}^k$, and program the RO so that the shares unmask the real tokens,

as in the proof of Theorem 5.2.5 (that is, L is a PT function that gives the length of the pads masking the tokens).

For part (2), we reuse the procedures GarbleSim and InputSim in part (1). Let $\mathcal{A}_2$ attack the aut2 security of $\mathcal{G}_2$. Adversary $\mathcal{A}_1(1^k)$ runs $\mathcal{A}_2(1^k)$, simulating the latter's Garble and Input oracles via procedures GarbleSim and InputSim respectively. When $\mathcal{A}_2$ outputs Y, adversary $\mathcal{A}_1$ also outputs Y. Let Bad be the event that $\mathcal{A}_2$ queries (r, w) to the random oracle such that S is the suffix of w, prior to receiving the entire garbled input, where S is the seed generating the pads. Then $\Pr[\text{Bad}] \leq Q(k)/2^k$, where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_2$ or by the scheme algorithms) in the execution of game $\text{Aut2}_{\mathcal{G}_2}$ with $\mathcal{A}_2$ on input 1^k. Let X

and T be the garbled input given to $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. If Bad does not happen then

$$\begin{aligned}\mathbf{Adv}^{\mathrm{aut2}}_{\mathcal{G}_2}(\mathcal{A}_2, k) &= \Pr[Y \neq \mathsf{Ev}_2(F, T) \wedge \mathsf{De}_1(d, Y) \neq \perp] \\ &= \Pr[Y \neq \mathsf{Ev}_1(F, X) \wedge \mathsf{De}_1(d, Y) \neq \perp] = \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k),\end{aligned}$$

the second equality holds because $\mathsf{Ev}_2(F, T) = \mathsf{Ev}_1(F, X)$. Totally, by union bound, we obtain $\mathbf{Adv}^{\mathrm{aut2}}_{\mathcal{G}_2}(\mathcal{A}_2, k) \leq \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) + \Pr[\mathrm{Bad}] \leq \mathbf{Adv}^{\mathrm{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) + Q(k)/2^k$.

References

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[2] B. Applebaum. Key-dependent message security: Generic amplification and completeness. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 527–546. Springer, May 2011.

[3] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Aug. 2009.

[4] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[5] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[6] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163. Springer, July 2010.

[7] B. Applebaum, Y. Ishai, and E. Kushilevitz. How to garble arithmetic circuits. In R. Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, Oct. 2011.

[8] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In S. P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Feb. 2007.

[9] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai. Bounded key-dependent message security. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 423–444. Springer, May 2010.

[10] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In M. Backes and P. Ning, editors, *ESORICS 2009*, volume 5789 of *LNCS*, pages 424–439. Springer, Sept. 2009.

[11] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[12] M. Bellare, R. Dowsley, B. Waters, and S. Yilek. Standard security does not imply security against selective-opening. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 645–662. Springer, Apr. 2012.

[13] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher, to appear in *IEEE Symposium of Security and Privacy (Oakland' 13)*, 2013.

[14] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. Cryptology ePrint Archive, 2013.

[15] M. Bellare, V. Hoang, and S. Keelveedhi. Instantiating Random Oracles via UCEs. Manuscript, April 2013.

[16] M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Computer and Communications Security (CCS'12)*. ACM, pages 784–796, 2012.

[17] M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, Oct 2012.

[18] M. Bellare, V. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT 20122*, pages 134–153, 2012.

[19] M. Bellare, V. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. Cryptology ePrint Archive, Report 2012/564, Oct 2012.

[20] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.

[21] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006.

[22] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Aug. 2008.

[23] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *27th Intl. Colloquium on Automata, Languages, and Programming — ICALP 2000*, pages 512–523. Springer, 2000.

[24] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[25] D. Canright. A very compact s-box for aes. *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 441–455, 2005.

[26] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, Dec. 2010.

[27] S. Choi, J. Katz, R. Kumaresan, and H. Zhou. On the security of the free-xor technique. Cryptology ePrint Archive, Report 2011/510, Sep 2011.

[28] S. G. Choi, D. Dachman-Soled, T. Malkin, and H. Wee. Improved non-committing encryption with applications to adaptively secure protocols. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 287–302. Springer, Dec. 2009.

[29] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the "free-XOR" technique. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Mar. 2012.

[30] K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Aug. 2010.

[31] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Aug. 2005.

[32] I. Damgård and J. B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Aug. 2000.

[33] Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging Merkle-Damgård for practical applications. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 371–388. Springer, Apr. 2009.

[34] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[35] K. Frikken, M. Atallah, and C. Zhang. Privacy-preserving credit checking. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 147–154. ACM, 2005.

[36] S. Furuhashi. The messagepack format. http://msgpack.org.

[37] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Aug. 2010.

[38] O. Goldreich. Cryptography and cryptographic protocols. Manuscript, June 9 2001.

[39] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[40] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989.

[41] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[42] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In A. V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

[43] S. Goldwasser and S. Micali. Probabilistic encryption. Journal of computer and system sciences, 28(2), pp. 270–299, 1984.

[44] S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. Manuscript, full version of [45], July 2012.

[45] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Aug. 2008.

[46] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Feb. 2010.

[47] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 289–306. Springer, Apr. 2008.

[48] S. Gueron. Advanced encryption standard (AES) instructions set. *Intel Corporation*, 25, 2008.

[49] W. Henecka, A. Sadeghi, T. Schneider, I. Wehrenberg, et al. Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462. ACM, 2010.

[50] A. Herzberg and H. Shulman. Secure guaranteed computation. Cryptology ePrint Archive, Report 2010/449, 2010.

[51] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[52] Y. Huang, C. Shen, D. Evans, J. Katz, and A. Shelat. Efficient secure computation with garbled circuits. In S. Jajodia and C. Mazumdar, editors, *ICISS*, volume 7093 of *Lecture Notes in Computer Science*, pages 28–48. Springer, 2011.

[53] Intel. Intel SSE4 programmer's manual. software.intel.com/file/17971/.

[54] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, Nov. 2000.

[55] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002.

[56] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008.

[57] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 207–221. Springer, Jan. 2010.

[58] JustGarble source. http://cseweb.ucsd.edu/groups/justgarble

[59] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, report 2011/272, October 25 2011.

[60] S. Kamara and L. Wei. Special-purpose garbled circuits. Manuscript, 2012.

[61] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 485–492. ACM Press, Oct. 2010.

[62] J. Katz and L. Malka. Constant-round private function evaluation with linear complexity. In D. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 556–571. Springer, Dec 2011.

[63] J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, Aug. 2004.

[64] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.

[65] V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In G. Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 83–97. Springer, Jan. 2008.

[66] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21th USENIX Security Symposium (USENIX 2012)*, 2012. Full version as Cryptology ePrint Archive, Report 2012/179.

[67] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 410–420. ACM Press, Oct. / Nov. 2006.

[68] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity* (ECCC), TR04-063, 2004.

[69] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, May 2007.

[70] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, Apr. 2009.

[71] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Mar. 2011.

[72] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 20–20. USENIX Association, 2004.

[73] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Feb. 2004.

[74] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[75] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Apr. 2006.

[76] P. Mohassel and S. Sadeghian. How to hide circuits in MPC: An efficient framework for Private Function Evaluation, to appear in *EUROCRYPT 2013*. Springer, May 2013.

[77] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM STOC*, pages 590–599. ACM Press, July 2001.

[78] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[79] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Aug. 2002.

[80] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Mar 2009.

[81] A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 89–106. Springer, June 2009.

[82] K. Pietrzak. A leakage-resilient mode of operation. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer, Apr. 2009.

[83] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):12–19, 2002.

[84] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Dec. 2009.

[85] P. Rogaway. The round complexity of secure protocols. MIT Ph.D. Thesis, 1991.

[86] P. Rogaway and J. P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 433–450. Springer, Aug. 2008.

[87] A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *International Conference on Information Security and Cryptology (ICISC'08)*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

[88] A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 463–472. ACM Press, Oct. 2010.

[89] T. Schneider. Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, 2008.

[90] T. Schneider. *Engineering Secure Two-Party Computation Protocols – Advances in Design, Optimization, and Applications of Efficient Secure Function Evaluation*. PhD thesis, Ruhr-University Bochum, Germany, February 9, 2011. http://thomaschneider.de/papers/S11Thesis.pdf.

[91] S. Tate and K. Xu. On garbled circuits and constant round secure function evaluation. Technical report, Computer Privacy and Security Lab, Department of Computer Science, University of North Texas, 2003.

[92] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 519–528. ACM Press, Oct. 2007.

[93] L. Valiant. Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203. ACM, 1976.

[94] I. Wegener. The complexity of Boolean functions. John Wiley & Son, NY, USA, 1987.

[95] R. Winternitz. A secure one-way hash function built from DES. *Proceedings of the IEEE Symposium on Information Security and Privacy*, pages 88–90. IEEE Press, 1984.

[96] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[97] A. C. Yao. Protocols for secure computations. In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.