The proceedings version of this paper appears in *CCS 2012* [12]. This is the full paper, an earlier version of which was circulated under the title "Garbling Schemes".

Foundations of Garbled Circuits

Mihir Bellare[1] Viet Tung Hoang[2] Phillip Rogaway[2]

[1] Dept. of Computer Science and Engineering, University of California, San Diego, USA
[2] Dept. of Computer Science, University of California, Davis, USA

October 1, 2012

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). Starting from a PRF, we provide an efficient garbling scheme achieving privacy and we analyze its concrete security. We next consider *obliviousness* and *authenticity*, properties needed for private and verifiable outsourcing of computation. We extend our scheme to achieve these ends. We provide highly efficient blockcipher-based instantiations of both schemes. Our treatment of garbling schemes presages more efficient garbling, more rigorous analyses, and more modularly designed higher-level protocols.

Keywords: Garbled circuits, garbling schemes, provable security, secure function evaluation, Yao's protocol.

Table of Contents

1 Introduction

Overview. This paper 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.

History. The idea of a garbled circuit is due to A. Yao, who described the technique in oral presentations [21, p. 27] about SFE [60, 61]. The first written account of the method is by Goldreich, Micali, and Wigderson [22]. The protocol they describe, crediting Yao [60], 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 specifically, there's a four-row table for each gate of the circuit, each row employing public-key encryption[3] to encrypt a pair of random strings whose xor is the token for the outgoing wire.

The term *garbled circuit*[4] 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 [46].

Yao's idea has been enormously impactful, engendering numerous applications, implementations, and refinements.[5] Still, there has been little definitional attention paid to garbled circuits themselves. A 2004/2009 paper by Lindell and Pinkas [39, 41] provides the first 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,[6] is a secure two-party SFE. Implemented SFE methods do not coincide with what's in Lindell and Pinkas [41], and absence of a good abstraction boundary makes daunting the task of providing a full proof for what's actually in optimized SFE implementations.

Scattered throughout the enormous literature dealing with garbled circuits, several papers do work to abstract out what these provide. A first set of such work begins with Feige, Kilian, and Naor [18] and is followed by [9, 16, 31, 34]. Each paper aims to modularly use garbled circuits in some intending application. To that end, they single out, definitionally, precisely what they need, usually arriving at something close to what we will later call "prv.sim security over Φ_{circ}." None of the papers pick up definitions from any other, nor does any prove that any particular construction satisfies the notion given. The conceptualization of garbling as involving a component that creates garbled circuits and another that evaluates them is found in all of these works, and in Schneider's [55, 56]. A second line of definitions begins with Ishai and Kushilevitz [28] and continues with [2, 4, 6, 7, 29, 30, 53]. 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 to investigate obliviousness, authenticity, and precise measures of efficiency. Finally, in concurrent work, Kamara and Wei offer definitions to support their idea of garbling *structured* circuits [32]. See Appendix A for further discussion of selected related work.

[3] 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.

[4] Synonyms in the literature include *encrypted circuit* and *scrambled circuit*.

[5] There are more than 2,700 Google-scholar-known citations to [60, 61].

[6] This approach for making the rows of the garbled gate is first mentioned by Goldreich [21].

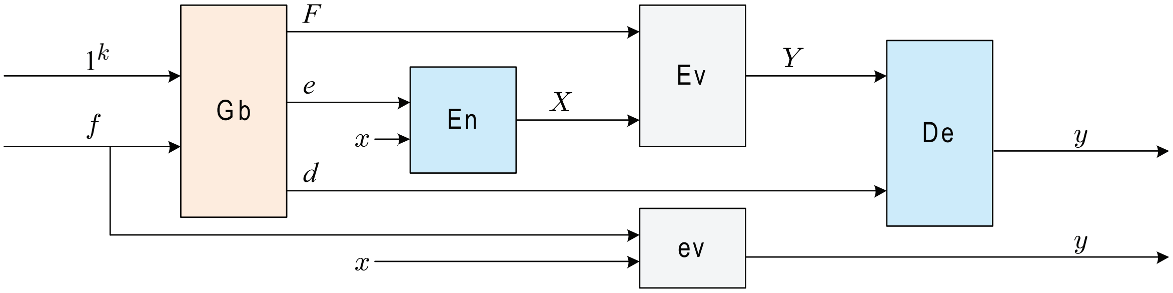


Fig. 1. Components of a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Function Gb maps f and k to (F, e, d), strings encoding the garbled function, the encoding function, and the decoding function. Possession of e and x lets one compute the garbled 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 $\mathsf{ev}(f, x)$.

Contributions. 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.

Roughly said, a garbling algorithm Gb is a randomized algorithm that transforms a function $f : \{0,1\}^n \to \{0,1\}^m$ 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* $y = d(Y)$, 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. 1.

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

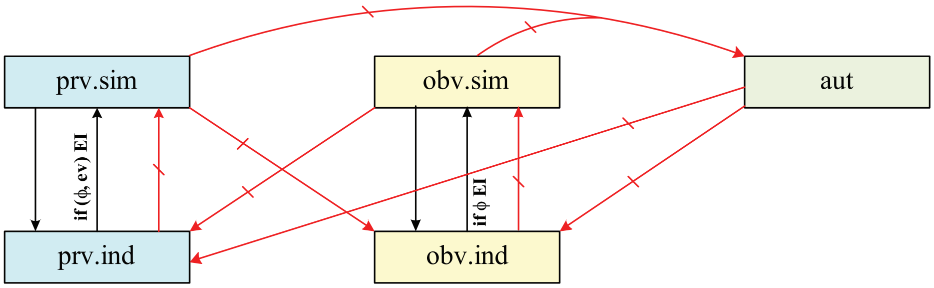


Fig. 2. Relations among security notions. A solid arrow is an implication; an if-labeled arrow, a conditional implication; a hatched arrow, a separation. Implications and separations are found in Section 4.

This is the fastest approach now known for garbling circuits. We do not prove or implement such a scheme secure in the current paper; see the discussion below.

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 [20]; 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. 2.

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. Once viewed as a "theoretical" approach for multiparty computation, a long line of work, beginning with Fairplay [43], 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 [26, 27, 37]. We aim to support such work, and applications further afield. 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. As an analog, authenticated encryption took off after it was reconceptualized as a primitive, not a method formed of encryption schemes and MACs.

Garble1 and Garble2 are close to numerous other protocols (especially [46]) 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 [41]. 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 [41], 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.

This paper is the first of several we envision. In it we aim to instill fresh, practice-oriented foundations in an area where, historically, omitted definitions and proofs have been the norm. The current work maintains a circumscribed focus: to investigate the definitions central to the reconceptualization of garbling schemes as a *sui generis* cryptographic goal. Upcoming work will explore several directions:

- We can construct *extremely efficient* garbling schemes, like the one-call, fixed-key, AES128-based scheme we mentioned. This can be done in a way that does not preclude the free-xor and row-elimination techniques that have proven so effective [26, 35, 50]. Proofs remain complex, even in the random-permutation model. Implementations are underway, these achieving about 15 nsec/gate.
- We can generalize security to the *adaptive* (=*dynamic*) setting. This is needed for one-time programs [23] and secure outsourcing [20]. For one flavor of adaptivity, prv1/obv1/aut1, the input x may depend on the garbled function F. For finer-grained notions, prv2/obv2/aut2, each bit of x can depend on previously acquired X_i-values. Transformations turn prv/obv/aut schemes into prv1/obv1/aut1 ones and these into prv2/obv2/aut2 ones.
- Building on the oft-described metaphor of lockboxes and keys (eg, [41, pp. 163–164]), we can formulate garbling-scheme security using a formal treatment of dual-key enciphering. We choose to do this by absorbing the functionality of the ideal primitive into the code-based definition. Privacy, obliviousness, and authenticity become yes/no matters—no probabilities.[7]

For all of these directions, the framework developed here serves as the needed starting point.

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. 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].

2 Preliminaries

This section provides basic notation, definitions and conventions. A reader might skip this on first reading and refer back as necessary.

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 $\text{Func}(a, b)$ 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 $\varepsilon : \mathbb{N} \to \mathbb{R}^+$ 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 [14] so we recall here the language and specify the particular conventions we use. A code-based game—see Fig. 5 for an example—consists of an

[7] In fact, the only hint of an intended model for Yao's work on two-party SFE is an idealized one supporting perfect, deterministic, public-key encryption [61, Section 3.2]. Formal treatments may be possible beyond garbling schemes, to the applications that routinely use them.

Protocol	**Application**	**Needs**	**Over**	**Also needs**
Y86 [21]	2-party SFE (semi-honest)	prv	Φ_{circ}	oblivious transfer (OT)
AF90 [1]	PFE (semi-honest)	prv	Φ_{size}	OT
FKN94 [18]	server-aided SFE (semi-honest)	prv	Φ_{circ}	none
NPS99 [46]	privacy-preserving auctions	prv	Φ_{circ}	proxy OT
KO04 [34]	2-party SFE (malicious)	prv	Φ_{circ}	OT, ZK proofs, commitment, trapdoor perm
FAZ05 [19]	private credit checking	prv	Φ_{size}	sym encryption
FM06 [44]	2-party SFE (malicious)	prv	Φ_{circ}	OT, commitment
AL07 [8]	2-party SFE (covert)	prv	Φ_{circ}	OT, commitment
LP07 [40]	2-party SFE (malicious)	prv	Φ_{circ}	OT, commitment
GKR08 [23]	one-time programs	prv2	Φ_{size}	none (model provides one-time memory)
GMS08 [24]	2-party SFE (covert)	prv	Φ_{circ}	OT, commitment, PRG, CR hash
BFK+09 [10]	private medical diagnostics	obv	Φ_{circ}	OT, sym encryption, homomorphic enc
PSS09 [47]	private credit checking	prv	Φ_{topo}	sym encryption
BHHI10 [9]	KDM encryption	prv	Φ_{size}	KDM encryption (wrt to linear functions)
GGP10 [20]	secure outsourcing	aut1 + obv1	Φ_{circ}	fully homomorphic encryption (FHE)
HS10 [25]	2-party guaranteed SFE	prv	Φ_{circ}	OT, auth encryption, asym enc, signature
KM10 [33]	secure text processing	prv	Φ_{topo}	OT, oblivious PRF
SS10 [53]	worry-free encryption	prv	Φ_{size}	asym encryption, signature
A11 [2]	KDM encryption	prv	Φ_{size}	KDM encryption (wrt to projections)
KMR11 [31]	server-aided SFE (malicious)	aut + obv	Φ_{circ}	coin-tossing protocol, commitment
LP11 [42]	2-party SFE (malicious)	prv	Φ_{circ}	OT, commitment

Fig. 3. Recasting protocols in more generic terms. All of the above protocols appear to be alternatively describable from a garbling scheme meeting our definitions. All but [20] need the scheme to be projective.

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 $\text{Gm}^{\mathcal{A}}(k)$, 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 "$\text{Gm}^{\mathcal{A}}(k) \Rightarrow c$" denote the event that this game output takes value c and let "$\text{Gm}^{\mathcal{A}}(k)$" be shorthand for "$\text{Gm}^{\mathcal{A}}(k) \Rightarrow \mathsf{true}$." Boolean flags are assumed initialized to false and $\mathsf{BAD}(\text{Gm}^{\mathcal{A}}(k))$ is the event that the execution of game Gm with adversary $\mathcal{A}$ 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

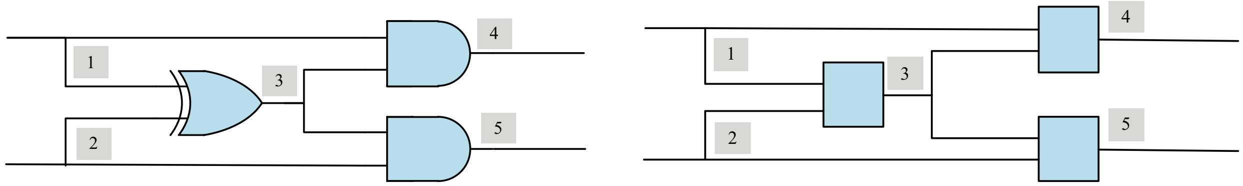


Fig. 4. **Left: A conventional circuit** $f = (n, m, q, A, B, G)$. It has $n = 2$ inputs, $m = 2$ outputs, and $q = 3$ gates. Gates are numbered 3, 4, 5, according to their outgoing wires. The diagram encodes $A(3) = 1$, $B(3) = 2$, $A(4) = 1$, $B(4) = 3$, $A(5) = 3$, and $B(5) = 2$. The gate symbols 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.

is or is not a valid circuit.[8] The boundary cases matter; we have repeatedly found that degenerate or under-analyzed circuit types materially impact if a garbling scheme is correct.[9] Beyond this, a lack of agreement on what a circuit *is* makes even informal discourse problematic.[10] Finally, we have found that it is simply 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. 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.

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 $A\colon \text{Gates} \to \text{Wires}\backslash\text{OutputWires}$ is a function to identify each gate's *first* incoming wire and $B : \text{Gates} \to \text{Wires}\backslash\text{OutputWires}$ 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. 4 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 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 $x = x_1 x_2 \cdots x_n$:

[8] 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 $G(x, y) = \overline{x} \vee y$?

[9] For example, the scheme of Naor, Pinkas, and Sumner [46] 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 [57] because of a dependency in gate-labels associated to fan-out ≥ 2 gates.

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

```
01 proc $\mathsf{ev_{circ}}(f, x)$
02 $(n, m, q, A, B, G) \leftarrow f$
03 for $g \leftarrow n + 1$ to $n + q$ do $a \leftarrow A(g)$, $b \leftarrow B(g)$, $x_g \leftarrow G_g(x_a, x_b)$
04 return $x_{n+q-m+1} \cdots x_{n+q}$
```

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 $\mathsf{ev_{circ}}$ 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. 4. 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.

3 Garbling Schemes and Their Security

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

3.1 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.[11] 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 $f.m = \mathsf{m}(f)$.[12] 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 *garbled input* $X = \mathsf{En}(e, x)$.[13] 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, $\mathsf{ev} = \mathsf{ev_{circ}}$ for the canonical circuit-evaluation function that we defined in Section 2.3.

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

[12] 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.

[13] 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.

```
proc GARBLE($f_0, f_1, x_0, x_1$)                    Game $\mathrm{PrvInd}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
if $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ then return $\bot$
if $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$ then return $\bot$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$;   $X \leftarrow \mathsf{En}(e, x_b)$
return $(F, X, d)$
```

```
proc GARBLE($f, x$)                                  Game $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $b = 1$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$; $X \leftarrow \mathsf{En}(e, x)$
         else $y \leftarrow \mathsf{ev}(f, x)$; $(F, X, d) \leftarrow \mathcal{S}(1^k, y, \Phi(f))$
return $(F, X, d)$
```

```
proc GARBLE($f_0, f_1, x_0, x_1$)                    Game $\mathrm{ObvInd}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
if $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ then return $\bot$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$;   $X \leftarrow \mathsf{En}(e, x_b)$
return $(F, X)$
```

```
proc GARBLE($f, x$)                                  Game $\mathrm{ObvSim}_{\mathcal{G},\Phi,\mathcal{S}}$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $b = 1$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$; $X \leftarrow \mathsf{En}(e, x)$
         else $(F, X) \leftarrow \mathcal{S}(1^k, \Phi(f))$
return $(F, X)$
```

```
proc GARBLE($f, x$)                                  proc FINALIZE($Y$)                 Game $\mathrm{Aut}_{\mathcal{G}}$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$        if $x \notin \{0,1\}^{f.n}$ then return 0
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$;   $X \leftarrow \mathsf{En}(e, x)$    return $(\mathsf{De}(d, Y) \neq \bot$ and $Y \neq \mathsf{Ev}(F, X))$
return $(F, X)$
```

Fig. 5. 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 $(b = b')$.

3.2 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. However, 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.

3.3 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, $\Phi_{\mathrm{size}}(f) = (n, m, q)$ 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: $\Phi_{\mathrm{topo}}(f) = (n, m, q, A, B)$, with notation and conventions as above. Side-information function Φ_{circ} reveals the entire circuit: $\Phi_{\mathrm{circ}}(f) = f$.

3.4 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 $\mathrm{PrvInd}_{\mathcal{G},\Phi}$ (top-left of Fig. 5) and a simulation-based notion of privacy via game $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$ (top-right of Fig. 5, 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 Section 2.

Refer first to game $\mathrm{PrvInd}_{\mathcal{G},\Phi}$. 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

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

the probability, normalized to $[0, 1]$, that the adversary correctly predicts b. Protocol $\mathcal{G}$ is prv.ind secure over Φ if for every PT adversary $\mathcal{A}$ the function $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.ind},\,\Phi}(\mathcal{A}, \cdot)$ 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 $\mathrm{PrvSim}_{\mathcal{G},\Phi,\mathcal{S}}$ 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. 5, 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

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

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 $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.sim},\,\Phi,\mathcal{S}}(\mathcal{B}, k)$ 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 4.

3.5 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. 5. 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

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

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

3.6 Authenticity

So far we have dealt exclusively with secrecy notions. One can formalize an authenticity property as well [20], which we do via game $\mathrm{Aut}_{\mathcal{G}}$ of Fig. 5. 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 $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A},k) = \Pr[\mathrm{Aut}_{\mathcal{G}}^{\mathcal{A}}(k)]$. Protocol $\mathcal{G}$ is aut-secure if for all PT adversaries $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A},\cdot)$ is negligible.

3.7 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.

3.8 Remarks

We end this section with discussion of our definitions.

Universal circuits. Fix one of our privacy or obliviousness notions and a projective garbling scheme $\mathcal{G}$ secure for Φ_{topo}. Then, using universal circuits, it is easy to construct a garbling scheme $\mathcal{G}_{\$}$ secure in the same sense but now with respect to Φ_{size}. This has long been understood in folklore, and is easily formalized using the language we have introduced. See Appendix B for details on universal circuits and the overhead they entail. Because of the ready translation from security with respect to Φ_{topo} to security with respect to Φ_{size}, the schemes we present in the remainder of this paper, Garble1 and Garble2, will have side-information Φ_{topo}.

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. 5, 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 [41], 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. It is important not to 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.[14]

Idealized models. As in many cryptographic domains, it seems possible to obtain better efficiency working in idealized models [13]. All of our security definitions easily lift to ideal-model settings. In the *random-oracle* model (ROM) [13], 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. 6 (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 a formal symbol indicating to the

[14] An asymptotic version of the counterexample is in Proposition 8.

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

```
proc Hash($\ell, w$)
if $\mathsf{H}[\ell, w] = \bot$ then $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$
```

Fig. 6. 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.

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.

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. 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.

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 all side-information functions $\Phi_{\rm circ}, \Phi_{\rm topo}$, and $\Phi_{\rm size}$ are efficiently invertible, and that, $(\Phi_{\rm size}, \mathsf{ev}_{\mathsf{circ}})$ and $(\Phi_{\rm topo}, \mathsf{ev}_{\mathsf{circ}})$ are efficiently invertible.

Proposition 1 For $\Phi \in \{\Phi_{\rm size}, \Phi_{\rm topo}, \Phi_{\rm circ}\}$, there is a linear-time inverter. For $\Phi \in \{\Phi_{\rm size}, \Phi_{\rm topo}\}$ there is a linear-time $(\Phi, \mathsf{ev}_{\mathsf{circ}})$-inverter.

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

Proof (Proposition 1). We first specify a linear-time $(\Phi_{\rm topo}, \mathsf{ev}_{\mathsf{circ}})$-inverter $M_{\rm topo}$. 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 $g \le n+q-m$ 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. 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. Now a linear-time Φ_{topo}-inverter, on input $f^- = (n, m, q, A, B)$, can let $y \leftarrow 0^m$ and return $M_{\text{topo}}(f^-, y)$. Similarly, a linear-time Φ_{size}-inverter, on input (n, m, q), can let $y \leftarrow 0^m$ and return $M_{\text{size}}((n, m, q), y)$. Finally, a linear-time Φ_{circ}-inverter is trivial, returning f on input f. □

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 2 [prv.ind ≈ 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 8 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 2 are tight. This is evidenced by Eq. (1) and Eq. (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 (Proposition 2). 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 $\mathbf{Adv}_{\mathcal{G}}^{\text{prv.ind}, \Phi}(\mathcal{A}, k) = 0$ and it will be trivial to construct $\mathcal{B}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\text{prv.sim}, \Phi, \mathcal{S}}(\mathcal{B}, k) = 0$ 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. Let $\mathcal{S}$ be *any* algorithm playing the role of the simulator. Then

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

$$\Pr\left[\neg\text{PrvSim}_{\mathcal{G}, \Phi, \mathcal{S}}^{\mathcal{B}}(k) \mid b = 0\right] = \frac{1}{2}$$

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

$$\mathbf{Adv}_{\mathcal{G}}^{\text{prv.ind}, \Phi}(\mathcal{A}, k) \le 2 \cdot \mathbf{Adv}_{\mathcal{G}}^{\text{prv.sim}, \Phi, \mathcal{S}}(\mathcal{B}, k) \ . \qquad (1)$$

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 $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{prv.ind}, \Phi)$ and let M be a (Φ, ev)-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\text{prv.sim}, \Phi)$. 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 $\textsc{Garble}$, 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 $\textsc{Garble}$ oracle to get back (F, X, d), which it returns to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. Then

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

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

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

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

A corollary of Propositions 1 and 2 is that prv.sim and prv.ind are equivalent for circuit-garbling schemes over side-information functions Φ_{topo} and Φ_{size}, which we summarize as:

Corollary 1. For $\Phi \in \{\Phi_{\text{topo}}, \Phi_{\text{size}}\}$, $\mathsf{GS}(\text{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) = \mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$.

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 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 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 2, prv.ind security then implies prv.nprom security.

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

Proposition 9 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.

Proof (Proposition 3). The proof is analogous to that of Proposition 2 but 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. Let $\mathcal{S}$ be *any* algorithm playing the role of the simulator. Then

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

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

$$\mathbf{Adv}^{\text{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \le 2 \cdot \mathbf{Adv}^{\text{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k)\ . \tag{3}$$

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 $\mathcal{A}$. Then

$$\Pr\left[\,\mathrm{ObvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k) \;\middle|\; b=1\,\right] = \Pr\left[\,\mathrm{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \;\middle|\; c=1\,\right]$$
$$\Pr\left[\,\neg\mathrm{ObvInd}^{\mathcal{A}}_{\mathcal{G},\Phi}(k) \;\middle|\; b=0\,\right] = \Pr\left[\,\neg\mathrm{ObvSim}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \;\middle|\; c=0\,\right]$$

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

$$\mathbf{Adv}^{\text{obv.sim},\,\Phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) \le \mathbf{Adv}^{\text{obv.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k)\ . \tag{4}$$

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

Again a corollary of Propositions 1 and 3 is that obv.sim and obv.ind are equivalent for circuit-garbling schemes over side-information functions Φ_{circ}, Φ_{topo} and Φ_{size}:

Corollary 2. $\mathsf{GS}(\text{obv.ind}, \Phi) = \mathsf{GS}(\text{obv.sim}, \Phi)$, for any $\Phi \in \{\Phi_{\text{topo}}, \Phi_{\text{size}}, \Phi_{\text{circ}}\}$.

4.4 Separations

We justify the non-implications for the security notions compactly summarized in Fig. 2. We state these as non-containments $\mathsf{A} \not\subseteq \mathsf{B}$ between sets of garbling schemes. We always assume $\mathsf{A} \ne \emptyset$, 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 4 For all Φ and for $\mathsf{ev} = \mathsf{ev}_{\mathsf{circ}}$: $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \not\subseteq \mathsf{GS}(\text{obv.ind}, \Phi)$.

Proof (Proposition 4). By assumption $\mathsf{GS}(\text{prv.sim}, \Phi) \cap \mathsf{GS}(\mathsf{ev}) \ne \emptyset$ 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 5 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 (Proposition 5). 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 6 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 (Proposition 6). 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 7 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 (Proposition 7). 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 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 $\mathbf{Adv}_P^{\mathrm{ow}}(\mathcal{I}, \cdot)$ is negligible, where for each $k \in \mathbb{N}$ we have let

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

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

We note that the (Φ, ev) in Proposition 8 is not efficiently invertible due to the one-wayness of P, so this separation is consistent with Proposition 2.

Proof (Proposition 8). We build $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ so that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv.ind}, \Phi) \cap \mathsf{GS}(\mathsf{ev})$ but $\mathcal{G} \notin \mathsf{GS}(\mathrm{prv.sim}, \Phi)$. 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 $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.ind}, \Phi}(\mathcal{A}) = 0$ 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 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 9 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. 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 (Proposition 9). 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 makes Garble query (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 $\Phi_{\text{circ}}(f_0) = \Phi_{\text{circ}}(f_1)$. 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 topological 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. □

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., [41, pp. 163–164]), but Lindell and Pinkas also give a notion of *double-encryption security* for two-key probabilistic encryption schemes [41, 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 $(A, B, T, X) \mapsto \mathbb{E}^T_{A,B}(X)$ 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.

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. 7 and illustrated in Fig. 8. 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 $\mathsf{a}, \mathsf{b} \in \{0,1\}$, we mean that $g \bmod 2^{\tau(k)-2}$ is encoded as a $(\tau(k)-2)$-bit string and $\mathsf{a} \,\|\, \mathsf{b}$ 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^b_i

```
100 proc $\mathsf{Gb}(1^k, f)$
101 $(n, m, q, A', B', G) \leftarrow f$
102 for $i \in \{1, \ldots, n+q-m\}$ do $t \twoheadleftarrow \{0,1\}$,  $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}t$,  $X_i^1 \twoheadleftarrow \{0,1\}^{k-1}\bar{t}$
103 for $i \in \{n+q-m+1, \ldots, n+q\}$ do $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}0$,  $X_i^1 \twoheadleftarrow \{0,1\}^{k-1}1$
104 for $(g, i, j) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ do
105     $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$
106     $A \leftarrow X_a^i$, $\mathsf{a} \leftarrow \mathrm{lsb}(A)$,  $B \leftarrow X_b^j$, $\mathsf{b} \leftarrow \mathrm{lsb}(B)$,  $T \leftarrow g \,\|\, \mathsf{a} \,\|\, \mathsf{b}$,  $P[g, \mathsf{a}, \mathsf{b}] \leftarrow \mathbb{E}_{A,B}^{T}\big(X_g^{G_g(i,j)}\big)$
107 $F \leftarrow (n, m, q, A', B', P)$
108 $e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
109 $d \leftarrow \varepsilon$
110 return $(F, e, d)$

120 proc $\mathsf{En}(e, x)$
121 $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
122 $x_1 \cdots x_n \leftarrow x$,  $X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$
123 return $X$

130 proc $\mathsf{De}(d, Y)$
131 $(Y_1, \ldots, Y_m) \leftarrow Y$
132 for $i \in \{1, \ldots, m\}$ do $y_i \leftarrow \mathrm{lsb}(Y_i)$
133 return $y \leftarrow y_1 \cdots y_m$

140 proc $\mathsf{ev}(f, x)$
141 $(n, m, q, A, B, G) \leftarrow f$,  $x_1 \cdots x_n \leftarrow x$
142 for $g \leftarrow n+1$ to $n+q$ do
143     $a \leftarrow A(g)$,  $b \leftarrow B(g)$
144     $x_g \leftarrow G_g(x_a, x_b)$
145 return $x_{n+q-m+1} \cdots x_{n+q}$

150 proc $\mathsf{Ev}(F, X)$
151 $(n, m, q, A', B', P) \leftarrow F$,  $(X_1, \ldots, X_n) \leftarrow X$
152 for $g \leftarrow n+1$ to $n+q$ do
153     $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$
154     $A \leftarrow X_a$,  $\mathsf{a} \leftarrow \mathrm{lsb}(A)$,  $B \leftarrow X_b$,  $\mathsf{b} \leftarrow \mathrm{lsb}(B)$
155     $T \leftarrow g \,\|\, \mathsf{a} \,\|\, \mathsf{b}$,  $X_g \leftarrow \mathbb{D}_{A,B}^{T}\big(P[g, \mathsf{a}, \mathsf{b}]\big)$
156 return $(X_{n+q-m+1}, \ldots, X_{n+q})$
```

Fig. 7. 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.

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 tables, one for each gate g. Table $P[g, \cdot, \cdot]$ has four rows, entry a, b the row to use when the left incoming token is of type a and the right incoming token is of type 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.

5.2 Security notion for dual-key ciphers

We already defined the syntax of a DKC, a permutation $\mathbb{E}_{A,B}^T \colon \{0,1\}^k \to \{0,1\}^k$ 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. 9. The game starts by choosing a bit $b \twoheadleftarrow \{0,1\}$ and a key $K \twoheadleftarrow \{0,1\}^k$. It chooses

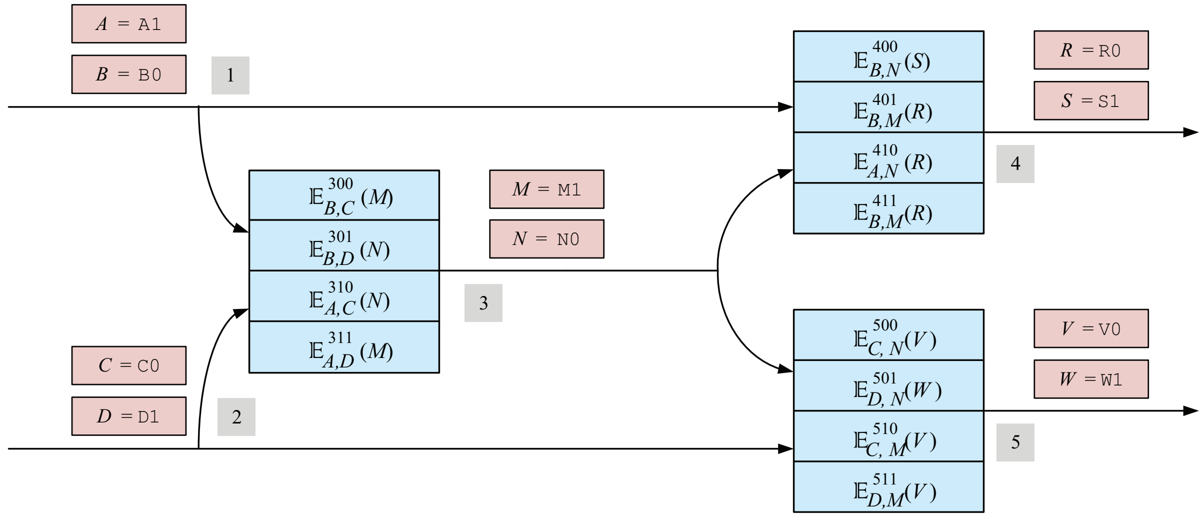


Fig. 8. Garbled circuit corresponding to the conventional circuit of Fig. 4. For each wire i, the token with semantics 0 (that is, X_i^0) is written on top; the token with semantics 1 (that is, X_i^1) is written on bottom. Possession of token A and C, for example, lets one decrypt the third row of the 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.

```
proc INITIALIZE()
$b \twoheadleftarrow \{0,1\}$,   $K \twoheadleftarrow \{0,1\}^k$,   $R_1, R_2, \ldots \twoheadleftarrow \{0,1\}^k$
for $i \in \{1, 2 \ldots\}$ do
    $K_{2i} \twoheadleftarrow \{0,1\}^{k-1}\,0$
    $K_{2i-1} \twoheadleftarrow \{0,1\}^{k-1}\,1$
return $\mathrm{lsb}(K)$

proc ENCRYPT($i, j, pos, T$)                              Game DKC
if $used[T]$ or $i \geq j$ then return $\perp$
$used[T] \leftarrow$ true
if $pos = 1$ then $(A, B) \leftarrow (K, K_i)$ else $(A, B) \leftarrow (K_i, K)$
if $b = 1$ then $X \leftarrow K_j$ else $X \leftarrow R_j$
return $(K_i, K_j, \mathbb{E}^T_{A,B}(X))$
```

Fig. 9. 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 $(b = b')$.

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 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.

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 5.4.

Theorem 10. Let $\mathbb{E}$ be 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 [41] 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 [41] 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 [51]) is well known.

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 $\text{PrvInd}_{\text{Garble1}[\mathbb{E}],\Phi_{\text{topo}}}$".

Overview. Without loss of generality, assume that $\mathcal{A}$ outputs (f_0, f_1, x_0, x_1) that satisfies $\Phi_{\text{topo}}(f_0) = \Phi_{\text{topo}}(f_1) = (n, m, q, A', B')$, $x_0, x_1 \in \{0,1\}^n$, and $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. We reformulate the game $\text{PrvInd}_{\text{Garble1}[\mathbb{E}],\Phi_{\text{topo}}}$ as game Real, and specify another game Fake whose output is independent of its challenge bit; thus $\Pr[\text{Fake}^{\mathcal{A}}(k)] = 1/2$. We also describe hybrid games $\text{Hy}_0, \ldots, \text{Hy}_{n+q-m}$ 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 $\ell \le q + n - m$. 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'),

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

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

Subtracting, we bound

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

Game Real. Consider game Real in Fig. 10. 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 ev(f, x, i)
(n, m, q, A, B, G) ← f
for g ← n + 1 to n + q do a ← A(g), b ← B(g), x_g ← 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 $\mathrm{Garb}(Y, \alpha, \beta)$ 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 $\mathrm{Garb\$}(\mathit{rnd}, \alpha, \beta)$ 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. 10. 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 $f^- = (n, m, q, A', B')$ 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 below. 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.

```
00 proc $\textsc{Garble}(f_0, f_1, x_0, x_1)$                                Game Real / Game Fake
01 $(n, m, q, A', B', G) \leftarrow f_c$
02 for $i \in \{1, \dots, n+q\}$ do
03     $v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$
04     if $i \le n+q-m$ then $t_i \twoheadleftarrow \{0,1\}$ else $t_i \leftarrow v_i$
05     $X_i^{v_i} \twoheadleftarrow \{0,1\}^{k-1} t_i$,  $X_i^{\overline{v}_i} \twoheadleftarrow \{0,1\}^{k-1} \overline{t}_i$
06 for $g \in \{n+1, \dots, n+q\}$ do
07     $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$,  $\textsc{Garb}(X_g^{v_g}, 0, 0)$
08     $\textsc{Garb\$}(\mathsf{false}, 1, 0)$,  $\textsc{Garb\$}(\mathsf{false}, 0, 1)$,  $\textsc{Garb\$}(\mathsf{false}, 1, 1)$     ⟵ Use in game Real
09     $\textsc{Garb\$}(\mathsf{true}, 1, 0)$,  $\textsc{Garb\$}(\mathsf{true}, 0, 1)$,  $\textsc{Garb\$}(\mathsf{true}, 1, 1)$     ⟵ Use in game Fake
10 $F \leftarrow (n, m, q, A', B', P)$
11 return $(F, (X_1^{v_1}, \dots, X_n^{v_n}), \varepsilon)$
```

```
00 proc $\textsc{Garble}(f_0, f_1, x_0, x_1)$                                Game $\mathrm{Hy}_\ell$
01 $(n, m, q, A', B', G) \leftarrow f_c$
10 for $i \in \{1, \dots, n+q\}$ do
11     $v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$
12     if $i \le n+q-m$ then $t_i \twoheadleftarrow \{0,1\}$ else $t_i \leftarrow v_i$
13     $X_i^{v_i} \twoheadleftarrow \{0,1\}^{k-1} t_i$,  $X_i^{\overline{v}_i} \twoheadleftarrow \{0,1\}^{k-1} \overline{t}_i$
20 for $g \in \{n+1, \dots, n+q\}$ do
21     $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$,  $\textsc{Garb}(X_g^{v_g}, 0, 0)$
22     $\textsc{Garb\$}(a \le \ell, 1, 0)$,  $\textsc{Garb\$}(b \le \ell, 0, 1)$,  $Y \leftarrow \textsc{Garb\$}(a \le \ell, 1, 1)$
23     if $a \le \ell < b$ and $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$ then $\textsc{Garb}(Y, 1, 0)$
24 $F \leftarrow (n, m, q, A', B', P)$
25 return $(F, (X_1^{v_1}, \dots, X_n^{v_n}), \varepsilon)$
```

```
30 private proc $\textsc{Garb}(Y, \alpha, \beta)$
31 $T \leftarrow g \,\|\, (t_a \oplus \alpha) \,\|\, (t_b \oplus \beta)$
32 $A \leftarrow X_a^{v_a \oplus \alpha}$,  $B \leftarrow X_b^{v_b \oplus \beta}$
33 $P[g, t_a \oplus \alpha, t_b \oplus \beta] \leftarrow \mathbb{E}_{A,B}^{T}(Y)$
```

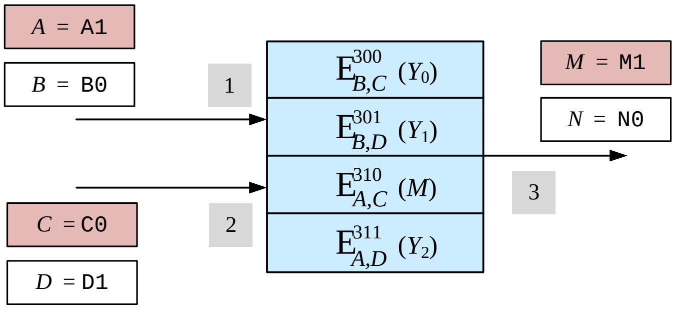
```
40 private proc $\textsc{Garb\$}(\mathit{rnd}, \alpha, \beta)$
41 if $\mathit{rnd}$ then $Y \twoheadleftarrow \{0,1\}^k$ else $Y \leftarrow X_g^{G_g(v_a \oplus \alpha, v_b \oplus \beta)}$
42 $\textsc{Garb}(Y, \alpha, \beta)$
43 return $Y$
```

Fig. 10. Games Real**,** Fake**, and** Hy_ℓ **(for** $0 \le \ell \le n+q-m$**) used in the proof of Theorem 10.** Each game has a procedure $\textsc{Initialize}()$ that samples a challenge bit $c \twoheadleftarrow \{0,1\}$. All variables are global. Each game has local procedures $\textsc{Garb}$ and $\textsc{Garb\$}$ to which the adversary $\mathcal{A}$ has no access. The procedure $\textsc{Finalize}(c')$ 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.

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

Hybrids. Now consider the hybrid games Hy_ℓ of Fig. 10, 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 from game $\mathrm{Hy}_{\ell-1}$, for every $\ell \in \{1, \dots, n+q-m\}$. 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 $\ell \in \{1, \dots, n+q-m\}$. In game Hy_ℓ, first run the code of game $\mathrm{Hy}_{\ell-1}$, and then do



	Y_0	Y_1	Y_2
Real	N	N	N
Hy$_1$	P	P	N
Fake	P	Q	R

Fig. 11. Garbled circuits of hybrid games, assuming that we garble $(f_c, x_c) = (\text{OR}, 00)$**.** There are three games: Real(Hy$_0$), Hy$_1$, and Fake(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.

the following update Δ_ℓ. For each token, associate it to a fresh uniformly random k-bit string. The two games Hy$_{\ell-1}$ and Hy$_\ell$ 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. 11 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 $P[g, \overline{t}_a, 0]$ and $P[g, \overline{t}_a, 1]$, 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 $P[g, t_a, \overline{t}_b]$ and $P[g, \overline{t}_a, \overline{t}_b]$, 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$_\ell$. Each garbled row will be built from the correct keys and tweak. For rows that 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 $P[g, \overline{t}_a, 0]$ and $P[g, \overline{t}_a, 1]$ 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 $P[g, t_a, \overline{t}_b]$ 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 $rnd = \texttt{true}$, 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. 12. Recall that $\mathcal{D}$ 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, 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 Hy$_{\ell-1}$ if the challenge bit of game DKC is 1, and simulate game Hy$_\ell$ 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 $i \ne \ell$ 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

```
00  proc $\textsc{Garble}(f_0, f_1, x_0, x_1)$        // as defined by adversary $\mathcal{D}$
01  $c \twoheadleftarrow \{0,1\}$,  $(n, m, q, A', B', G) \leftarrow f_c$,  $\ell \twoheadleftarrow \{1, \ldots, q+n\}$
10  for $i \in \{1, \ldots, n+q\}$ do $t_i \leftarrow v_i \leftarrow \mathsf{ev}(f_c, x_c, i)$
11  for $i \leftarrow \{1, \ldots, n+q-m\}$ do $t_i \twoheadleftarrow \{0,1\}$
12  for $g \in \{n+1, \ldots, n+q\}$ do
13    $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$,   $t_\ell \leftarrow \overline{\tau}$ //$\tau = \mathrm{lsb}(K)$
14    if $a = \ell$ then $\textsc{Query}(\mathsf{false}, 1, 0)$,  $\textsc{Query}(\mathsf{false}, 1, 1)$
15    if $b = \ell$ then $\textsc{Query}(\mathsf{false}, 0, 1)$,  $Y_g \leftarrow \textsc{Query}(\mathsf{true}, 1, 1)$
20  for $i \in \{1, \ldots, n+q\}$ do
21    if $X_i^{\overline{v}_i} = \perp$ and $i \neq \ell$ then $X_i^{\overline{v}_i} \twoheadleftarrow \{0,1\}^{k-1}\overline{t}_i$
22    if $X_i^{v_i} = \perp$ then $X_i^{v_i} \twoheadleftarrow \{0,1\}^{k-1} t_i$
30  for $g \in \{n+1, \ldots, n+q\}$ do
31    $a \leftarrow A'(g)$,  $b \leftarrow B'(g)$,  $\textsc{Garb}(X_g^{v_g}, 0, 0)$
32    if $a \neq \ell$ and $b \neq \ell$ then
33      $\textsc{Garb\$}(a \leq \ell, 1, 0)$,  $\textsc{Garb\$}(b \leq \ell, 0, 1)$,  $Y \leftarrow \textsc{Garb\$}(a \leq \ell, 1, 1)$
34      if $a \leq \ell < b$ and $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$ then $\textsc{Garb}(Y, 1, 0)$
35    elsif $a = \ell$ then $\textsc{Garb\$}(\mathsf{false}, 0, 1)$
36    else $\textsc{Garb\$}(\mathsf{true}, 1, 0)$,  if $G_g(\overline{v}_a, 0) = G_g(\overline{v}_a, 1)$ then $\textsc{Garb}(Y_g, 1, 0)$
37  $F \leftarrow (n, m, q, A', B', P)$
38  return $(F, (X_1^{v_1}, \ldots, X_n^{v_n}), \varepsilon)$
```

```
40  private proc $\textsc{Query}(\mathit{rnd}, \alpha, \beta)$
41  $T \leftarrow g \,\|\, (t_a \oplus \alpha) \,\|\, (t_b \oplus \beta)$
42  $\gamma \leftarrow v_g \oplus G_g(v_a \oplus \alpha, v_b \oplus \beta)$
43  if $a = \ell$ then
44    $\mathit{pos} \leftarrow 1$,  $i \leftarrow 2b + (t_b \oplus \beta)$
45  else
46    $\mathit{pos} \leftarrow 0$,  $i \leftarrow 2a + (t_a \oplus \alpha)$
47  if $\mathit{rnd}$ then
48    $j \twoheadleftarrow \{2(g+n+q), 2(g+n+q)+1\}$
49  else
50    $j \leftarrow 2g + (t_g \oplus \gamma)$
51  $(K_i, K_j, Z) \leftarrow \textsc{Encrypt}(i, j, \mathit{pos}, T)$
52  $P[g, t_a \oplus \alpha, t_b \oplus \beta] \leftarrow Z$
53  if $a = \ell$ then
54    $X_b^{v_b \oplus \beta} \leftarrow K_i$
55  else
56    $X_a^{v_a \oplus \alpha} \leftarrow K_i$
57  if $\overline{\mathit{rnd}}$ then $X_g^{v_g \oplus \gamma} \leftarrow K_j$
58  return $K_j$
```

Fig. 12. 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. 10. 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().

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 $\mathrm{Hy}_{\ell-1}$ 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 the high-level description of the "private" procedure $\textsc{Query}(\mathit{rnd}, \alpha, \beta)$. 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 $\mathrm{Hy}_{\ell-1}$, is the correct token ($\mathit{rnd} = \mathsf{false}$) or a random string ($\mathit{rnd} = \mathsf{true}$). 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 fresh random string, so pick $j \twoheadleftarrow \{2(n+q+g), 2(n+q+g)+1\}$.

How should $\mathcal{D}$ call Query? If $\ell = a$ then we want to write to rows $P[g, \overline{t}_a, 0]$ and $P[g, \overline{t}_a, 1]$. Let $\mathit{rnd} = \mathsf{false}$ 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 $P[g, t_a, \overline{t}_b]$ and $P[g, \overline{t}_a, \overline{t}_b]$. For the former, similarly, let $\mathit{rnd} = \mathsf{false}$. 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 $\varDelta_b$ 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 $\mathit{rnd} = \mathsf{true}$. 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 $\nu_i \le \nu$, 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 Q is uniformly distributed over the $(q+n)$-element multiset $\{2\nu_1, \ldots, 2\nu_{n+q-m}, 0, \ldots, 0\}$. Hence $Q \le 2\nu$, and

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

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 $\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prf}}(\mathcal{D}, k) = 2\Pr[\mathrm{PRF}_{\mathsf{F}}^{\mathcal{D}}(k)] - 1$ 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 $\mathtt{Tbl}[T] = \bot$, or returns $\mathtt{Tbl}[T]$ if $\mathtt{Tbl}[T] \ne \bot$. Finalize(b') returns $(b = b')$. We say that F is PRF-secure if $\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prf}}(\mathcal{D}, \cdot)$ is negligible for all polynomial-time adversaries $\mathcal{D}$.

Given a PRF F as above, we define the dual-key cipher $\mathbb{E}$ via $\mathbb{E}_{A,B}^T(P) = \mathsf{F}_{A[1:k-1]}(T) \oplus \mathsf{F}_{B[1:k-1]}(T) \oplus P$. This dual-key cipher has tweak length τ and is denoted $\mathbb{E}[\mathsf{F}]$. During evaluation, token types are revealed, but the entire key of F remains secret.

```
proc INITIALIZE()
    $a, b \twoheadleftarrow \{0,1\}$
    return $a$

proc ENCRYPT$(i, j, pos, T)$
    if $used[T]$ or $i \geq j$ then return $\bot$
    $used[T] \leftarrow \mathsf{true}$
    if $K_i = \bot$ then $A \twoheadleftarrow \{0,1\}^{k-1}$,   $K_i \leftarrow A \,\|\, (i \bmod 2)$
    if $K_j = \bot$ then $B \twoheadleftarrow \{0,1\}^{k-1}$,   $K_j \leftarrow B \,\|\, (j \bmod 2)$
    if $R_j = \bot$ then $R_j \twoheadleftarrow \{0,1\}^{k}$
    $X_1 \leftarrow K_j$,   $X_0 \leftarrow R_j$
    return $(K_i, K_j, \mathrm{FN}(T) \oplus \mathsf{F}_A(T) \oplus X_b)$
```

Fig. 13. Proof of Theorem 11. The code is for the adversary $\mathcal{B}$, which has a PRF oracle FN.

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 $\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prf}}(\mathcal{B}, k) = 0.5\,\mathbf{Adv}_{\mathbb{E}[\mathsf{F}]}^{\mathrm{dkc}}(\mathcal{A}, k)$. 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 11. 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. 13. 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 $\mathrm{FN}(T) \oplus \mathsf{F}_A(T) \oplus X_b$ 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

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

where c is the challenge bit of game PRF. To justify the second claim, note that if $c = 0$ then $\mathrm{FN}(T) \oplus \mathsf{F}_A(T) \oplus X_b$ 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 $\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prf}}(\mathcal{B}, k) = \frac{1}{2}\mathbf{Adv}_{\mathbb{E}[\mathsf{F}]}^{\mathrm{dkc}}(\mathcal{A}, k)$. □

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.

5.6 Dual-key ciphers from double encryption

We also prove the dkc-security of the instantiation $\mathbb{E}[E]$ in which $\mathbb{E}_{A,B}^{T}(X) = E_A(E_B(X))$, with E being 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 $\mathbf{Adv}_{\mathbb{E}[E]}^{\mathrm{dkc}}(\mathcal{A}, k) \leq (10Q^2 + 4Q + 8q_E)/2^k$. The above assumes that $Q + q_E \leq 2^{k-3}$.

Theorem 12. Let E be an ideal cipher. Then $\mathbb{E}[E]$ is a secure dual-key cipher.

Proof. Consider games G_0–G_5 in Fig. 14. 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 $q_E + Q \leq 2^{k-3}$.

```
Game $G_0$ $\boxed{G_1}$

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

proc ENC$(A, B)$
return $E_A(B)$

proc DEC$(A, B)$
return $E_A^{-1}(B)$

proc ENCRYPT$(i, j, pos, T)$
if $used[T]$ or $i \ge j$ then return $\bot$
$used[T] \leftarrow$ true, $s \leftarrow i \bmod 2$, $t \leftarrow j \bmod 2$
if $K_i = \bot$ then
    $K_i \twoheadleftarrow \{0,1\}^{k-1}s$
    if $K_i = K$ then $bad \leftarrow$ true, $\boxed{K_i \twoheadleftarrow \{0,1\}^{k-1}s \setminus \text{Keys}}$
    Keys $\leftarrow$ Keys $\cup \{K_i\}$
if $K_j = \bot$ then
    $K_j \twoheadleftarrow \{0,1\}^{k-1}t$
    if $K_j = K$ then $bad \leftarrow$ true, $\boxed{K_j \twoheadleftarrow \{0,1\}^{k-1}t \setminus \text{Keys}}$
    Keys $\leftarrow$ Keys $\cup \{K_j\}$
if $R_j = \bot$ then $R_j \twoheadleftarrow \{0,1\}^k$
$X_1 \leftarrow K_j$, $X_0 \leftarrow R_j$
if $pos = 1$ then $X \leftarrow E_{K_i}(X_b)$ else $X \leftarrow X_b$
$Y \leftarrow E_K(X)$
if $pos = 1$ then return $(K_i, K_j, Y)$
else return $(K_i, K_j, E_{K_i}(Y))$
```

```
Game $G_2$ $\boxed{G_3}$

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

proc ENC$(A, B)$
if $A \ne K$ then return $E_A(B)$
$bad \leftarrow$ true, $\boxed{\textbf{return } \bot}$
if $B \notin \text{Dom}(\pi)$ then
    $Y \leftarrow \{0,1\}^k \setminus \text{Ran}(\pi)$, $\pi[B] \leftarrow Y$
return $\pi[B]$

proc DEC$(A, B)$
if $A \ne K$ then return $E_A^{-1}(B)$
$bad \leftarrow$ true, $\boxed{\textbf{return } \bot}$
if $B \notin \text{Ran}(\pi)$ then
    $X \leftarrow \{0,1\}^k \setminus \text{Dom}(\pi)$, $\pi[X] \leftarrow B$
return $\pi^{-1}[B]$

proc ENCRYPT$(i, j, pos, T)$
if $used[T]$ or $i \ge j$ then return $\bot$
$used[T] \leftarrow$ true, $s \leftarrow i \bmod 2$, $t \leftarrow j \bmod 2$
if $K_i = \bot$ then $K_i \twoheadleftarrow \{0,1\}^{k-1}s \setminus \text{Keys}$, Keys $\leftarrow$ Keys $\cup \{K_i\}$
if $K_j = \bot$ then $K_j \twoheadleftarrow \{0,1\}^{k-1}t \setminus \text{Keys}$, Keys $\leftarrow$ Keys $\cup \{K_j\}$
if $R_j = \bot$ then $R_j \twoheadleftarrow \{0,1\}^k$
$X_1 \leftarrow K_j$, $X_0 \leftarrow R_j$
if $pos = 1$ and $Y \leftarrow H[(i,j)] \ne \bot$ then return $(K_i, K_j, Y)$
if $pos = 0$ and $Y \leftarrow H[j] \ne \bot$ then return $(K_i, K_j, E_{K_i}(Y))$
if $pos = 1$ then $X \leftarrow$ ENC$(K_i, X_b)$ else $X \leftarrow X_b$
if $X \notin \text{Dom}(\pi)$ then $Y \twoheadleftarrow \{0,1\}^k \setminus \text{Ran}(\pi)$, $\pi[X] \leftarrow Y$
$Y \leftarrow \pi[X]$
if $pos = 1$ then $H[(i,j)] \leftarrow Y$, return $(K_i, K_j, Y)$
else $H[j] \leftarrow Y$, return $(K_i, K_j, E_{K_i}(Y))$
```

```
Game $\boxed{G_4}$ $G_5$

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

proc ENC$(A, B)$
if $A \ne K$ then return $E_A(B)$
return $\bot$

proc DEC$(A, B)$
if $A \ne K$ then return $E_A^{-1}(B)$
return $\bot$

proc ENCRYPT$(i, j, pos, T)$
if $used[T]$ or $i \ge j$ then return $\bot$
$used[T] \leftarrow$ true, $s \leftarrow i \bmod 2$, $t \leftarrow j \bmod 2$
if $K_i = \bot$ then $K_i \twoheadleftarrow \{0,1\}^{k-1}s \setminus \text{Keys}$, Keys $\leftarrow$ Keys $\cup \{K_i\}$
if $K_j = \bot$ then $K_j \twoheadleftarrow \{0,1\}^{k-1}t \setminus \text{Keys}$, Keys $\leftarrow$ Keys $\cup \{K_j\}$
if $R_j = \bot$ then $R_j \twoheadleftarrow \{0,1\}^k$
$X_1 \leftarrow K_j$, $X_0 \leftarrow R_j$
if $pos = 1$ and $Y \leftarrow H[(i,j)] \ne \bot$ then return $(K_i, K_j, Y)$
if $pos = 0$ and $Y \leftarrow H[j] \ne \bot$ then return $(K_i, K_j, E_{K_i}(Y))$
if $pos = 1$ then $X \leftarrow E_{K_i}(X_b)$ else $X \leftarrow X_b$
$Y \twoheadleftarrow \{0,1\}^k$
if $X \notin \text{Dom}(\pi)$ then $\pi[X] \leftarrow Y$
else $bad \leftarrow$ true, $\boxed{Y \leftarrow \pi[X]}$
if $pos = 1$ then $H[(i,j)] \leftarrow Y$, return $(K_i, K_j, Y)$
else $H[j] \leftarrow Y$, return $(K_i, K_j, E_{K_i}(Y))$
```

Fig. 14. Games for the proof of Theorem 12. Procedure FINALIZE(b') returns $(b = b')$. Games G_1, G_3, and G_4 include the corresponding boxed statements, but games G_0, G_2, and G_5 do not.

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 sample them so that they are pairwise distinct. The two games are identical until either game sets *bad*. Moreover,

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

$\triangleright G_1 \to G_2$: 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 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.

$\triangleright G_2 \to G_3$: we take away from the adversary the power of querying $\text{Enc}(K, \cdot)$ and $\text{Dec}(K, \cdot)$. 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 the 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 89 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 92 is unreachable and *bad* won't be set. The flag *bad* is triggered only if $X \in \text{Dom}(\pi)$, where $\text{Dom}(\pi)$ is the set of the points that $\pi[\cdot]$ is defined prior to this query. For each $P \in \text{Dom}(\pi)$, 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 $\text{Dom}(\pi)$ 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 92 to set *bad*, and thus $K_j \neq K_{j^*}$ since Bad doesn't happen. 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 92 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 92 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 92 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 thus $\mathbf{Adv}^{\mathrm{dkc}}_{\mathbb{E}[E]}(\mathcal{A}, k) = 2\Pr[G_0^{\mathcal{A}}(k)] - 1 = 2(\Pr[G_0^{\mathcal{A}}(k)] - \Pr[G_5^{\mathcal{A}}(k)]) \leq (10Q^2 + 2Q + 8q_E)/2^k$. □

Unwinding the results. One needs to be careful in combining Theorems 10 and 12 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. Suppose 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 10 and 12, there is a random variable $0 < Q \leq 2\nu$ such that

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

The bound is quite satisfactory. Above, the expectation $\mathbf{E}[Q]$ appears in the first inequality because 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 $\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}, k) = p\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}_1, k) + (1-p)\mathbf{Adv}^{\mathrm{prv.ind},\,\Phi_{\mathrm{topo}}}_{\mathcal{G}}(\mathcal{A}_2, k)$.

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 10. 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, $E = \mathrm{AES128}$, $k = 129$, $|K| = 128$, 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 $E = \mathrm{AES128}$ 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 [48]. 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 [21] (which would have been understood that primitive to be probabilistic, not a blockcipher). The second method is

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

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

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

proc $\mathsf{ev}(f, x)$
$(n, m, q, A, B, G) \leftarrow f$,  $x_1 \cdots x_n \leftarrow x$
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 $x_{n+q-m+1} \cdots x_{n+q}$

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

Fig. 15. 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.

extremely efficient—the most efficient approach now known. Implementation work is currently underway to measure the magnitude of the gain:

- Let $\mathbb{E}_{A,B}^{T}(X) = E_A(E_B(X))$ (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 $\mathbb{E}_{A,B}^{T}(X) = E_{\mathsf{const}}(K) \oplus K \oplus X$ where $K = A \oplus B \oplus T$ and $E = $ 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. For a proof, we would 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 [35, 50].

Basing garbled-circuit evaluation on AES and employing AES-NI in an implementation was also suggested by Kreuter, Shelat, and Shen [37]. They use AES-256, rekeying with gate evaluation.

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. 15.

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 2 it does not matter whether we consider ind or sim, and for simplicity we pick the former.

Theorem 13. Let $\mathbb{E}$ be a secure DKC. 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 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 $Q \le 2\nu$ 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 (Theorem 13). The privacy security can be proved by adapting the proof of Theorem 10 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. 15. 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 10 still applies here.

For obliviousness security, we again adapt the proof of Theorem 10, 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

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

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 $\textsc{Encrypt}$; so it has as many $\textsc{Encrypt}$ 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 $\textsc{Garble}$ with (f', f, x, x). Since $f'(x) = y$ and $\Phi_{\text{topo}}(f') = \Phi_{\text{topo}}(f)$ and the side-information function is Φ_{topo}, the query (f', f, x, x) in game $\text{PrvInd}_{\mathcal{G},\Phi_{\text{topo}}}$ 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

$$\Pr\left[\,\text{PrvInd}_{\mathcal{G},\Phi_{\text{topo}}}^{\mathcal{B}}(k) \;\middle|\; b = 1\,\right] = \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k), \tag{5}$$

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

$$\Pr\left[\neg\mathrm{PrvInd}^{\mathcal{B}}_{\mathcal{G},\Phi_{\mathrm{topo}}}(k) \mid b=0\right] \le 2^{1-k} \ . \tag{6}$$

Subtracting Eq. (5) and Eq. (6) 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 $Y_i \ne Y_i^{y_i}$. This integer i is well-defined if $Y \ne F(X) = (Y_1^{y_1}, \ldots, Y_m^{y_m})$. Consequently, if $\mathsf{De}(d, Y) \ne \bot$ and $Y \ne F(X)$ then Y_i must be $Y_i^{\overline{y_i}}$. This uses the fact that $Y_i^{\overline{y_i}} \ne Y_i^{y_i}$, which is true because the construction makes their type-bits unequal. Thus we have

$$\Pr\left[\neg\mathrm{PrvInd}^{\mathcal{B}}_{\mathcal{G},\Phi_{\mathrm{topo}}}(k) \mid b=0\right] \le \Pr\left[Y_i = Y_i^{\overline{y_i}} \mid Y \ne F(X)\right] \ . \tag{7}$$

Let $A \| \mathsf{a} \leftarrow Y_i^{\overline{y_i}}$, 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. 15, 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. (7) is at most 2^{1-k}. □

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 paper, 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.

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

```
proc GetView($x, f$)                                        Game $\mathrm{PfeSim}_{\mathcal{F},i,\Phi,\mathcal{S}}$
$b \twoheadleftarrow \{0,1\}$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
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 $view \leftarrow \mathcal{S}(1^k, f, |x|)$
```

Fig. 16. 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 $k \in \mathbb{N}$.

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 $(conv, \omega_i)$ 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 (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 $\mathrm{PfeSim}_{\mathcal{F},i,\Phi,\mathcal{S}}$ of Fig. 16, and its advantage is

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

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 $\mathbf{Adv}^{\mathrm{pfe.sim},\Phi,\mathcal{S}}_{\mathcal{F},i}(\mathcal{B}, \cdot)$ 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 $\mathcal{OT} = (\Pi^{\mathrm{ot}}, \mathsf{ev}^{\mathrm{ot}})$ where Π^{ot} is a 2-party protocol and $\mathsf{ev}^{\mathrm{ot}}((X^0_1, X^1_1, \ldots, X^0_n, X^1_n), x) = (X^{x_1}_1, \ldots, X^{x_n}_n)$. Here, a function is described by a vector $(X^0_1, X^1_1, \ldots, X^0_n, X^1_n)$, and its evaluation on an n-bit input x is $(X^{x_1}_1, \ldots, X^{x_n}_n)$. We assume a pfe.sim-secure scheme $\mathcal{OT} = (\Pi^{\mathrm{ot}}, \mathsf{ev}^{\mathrm{ot}})$ relative to the side information function $\Phi_{\mathrm{ot}}((X^0_1, X^1_1, \ldots, X^0_n, X^1_n)) = (|X^0_1|, |X^1_1|, \ldots, |X^0_n|, |X^1_n|)$.

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 $\{\mathsf{ev}(f, \cdot)\colon f \in \{0,1\}^*\}$ 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 14. 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 $\mathcal{OT} = (\Pi^{\rm ot}, \mathsf{ev}^{\rm ot})$ is a OT protocol that is pfe.sim-secure relative to $\Phi_{\rm ot}$. Let $\mathcal{F} = (\Pi, \mathsf{ev})$ be the pfe scheme constructed above. Then $\mathcal{F}$ is pfe.sim-secure relative to Φ.

Proof (Theorem 14). 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

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

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 $\mathit{view}^{\rm ot} \leftarrow \mathsf{View}^1_{\Pi^{\rm ot}}(1^k, x, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1))$. It obtains this by direct execution of 2-party protocol $\Pi^{\rm ot}$ on inputs $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ for party 2 and x for party 1. Parsing $\mathit{view}^{\rm ot}$ as $(\mathit{conv}^{\rm ot}, \omega_1^{\rm ot})$, it appends $\mathit{conv}^{\rm ot}$ to *conv* and then returns $\mathit{view} = (\mathit{conv}, \omega_1^{\rm ot})$ 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 $\mathit{view}^{\rm ot} \leftarrow \text{GetView}(x, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1))$. Parsing $\mathit{view}^{\rm ot}$ as $(\mathit{conv}^{\rm ot}, \omega_1^{\rm ot})$, it appends $\mathit{conv}^{\rm ot}$ to *conv* and then returns $\mathit{view} = (\mathit{conv}, \omega_1^{\rm ot})$ 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 we have just built 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 lets $\mathit{view}^{\rm ot} \leftarrow \mathcal{S}_{\mathcal{OT}}(1^k, x, (X_1, \ldots, X_n), (|X_1|, |X_1|, \ldots, |X_n|, |X_n|))$. Parsing $\mathit{view}^{\rm ot}$ as $(\mathit{conv}^{\rm ot}, \omega_1^{\rm ot})$, it appends $\mathit{conv}^{\rm ot}$ to *conv* and then returns $\mathit{view} = (\mathit{conv}, \omega_1^{\rm ot})$.

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. □

7.2 KDM-secure encryption

We re-establish Applebaum's result [2] that projection-KDM security implies bounded-KDM security. 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.

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. 17. 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 $\mathbf{Adv}_{\Pi,\ell}^{\rm kdm}(\mathcal{A}, k) = 2\Pr[\text{KDM}^{\mathcal{A}}(k)] - 1$. In the asymptotic statements, p, s, and ℓ

```
proc Initialize()                                   proc KDM($j, f$)                                   Game KDM
$K_1, K_2, \dots K_\ell \twoheadleftarrow \mathcal{K}(1^k)$,  $b \twoheadleftarrow \{0,1\}$        $x \leftarrow \mathsf{ev}(f, K_1 \,\|\, \cdots \,\|\, K_\ell)$
                                                    if $b = 1$ then return $\mathcal{E}_{K_j}(x)$ else return $\mathcal{E}_{K_j}(0^{|x|})$
```

Fig. 17. Game for defining the KDM security. Procedure Finalize(b') returns ($b = b'$).

```
00  proc $\mathcal{E}'(K, x)$                                          10  proc $\mathcal{D}'(K, y)$
01  $n \leftarrow \max\{p \cdot \ell, s\}$, $z \leftarrow x0^{n-|x|}$                         11  $(F, d, Y) \leftarrow y$
02  for $i \in \{n+1, \dots, n+q+1\}$ do                        12  $X \leftarrow \mathcal{D}_K(Y)$,  $z \leftarrow \mathsf{De}(d, \mathsf{Ev}(F, X))$
03      $A(i) \leftarrow 1$,  $B(i) \leftarrow i-1$,  $G_i(a,b) \leftarrow \{$return $a\}$       13  return $z[1:s]$
04  for $i \in \{n+q+2, \dots, q+2n\}$ do
05      $A(i) \leftarrow 1$,  $B(i) \leftarrow i-n-q$,  $G_i(a,b) \leftarrow \{$return $b\}$
06  $\mathrm{ID} \leftarrow (n, n, q+n, A, B, G)$
07  $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, \mathrm{ID})$,  $X \leftarrow \mathsf{En}(e, z)$
08  return $(F, d, \mathcal{E}_K(X))$
```

Fig. 18. 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 $\{0,1\}^n$.

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, Ostrovsky (BHHO) [15] to instantiate a projection-KDM secure encryption scheme [15]. See the discussion of Applebaum [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 a scheme $\Pi' = (\mathcal{K}, \mathcal{E}', \mathcal{D}') = \mathrm{S2B}[\Pi, \mathcal{G}, q, \ell]$ that is q-bounded KDM secure, as shown in Fig. 18. The message space of Π' is also $\{0,1\}^s$.

Theorem 15. 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.

[15] In BHHO's scheme, the public key is a list of group elements $(g_1, \dots, g_r)$ and the private key is $(g_1^{s_1}, \dots, g_r^{s_r})$, with $s_1, \dots, s_r \twoheadleftarrow \{0,1\}$. However, if we view $s = s_1 \cdots s_r$ as the private key then BHHO's scheme is projection-KDM secure.

```
proc KDM(j, f)                                            Games G_0
    $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, \mathrm{ID})$,  $X \leftarrow \mathsf{En}\big(e, C(\mathbf{K})\big)$
    return $(F, d, \mathcal{E}_{K_j}(X))$
```

```
proc KDM(j, f)                                            Games G_1
    $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, C)$,  $X \leftarrow \mathsf{En}(e, \mathbf{K})$
    return $(F, d, \mathcal{E}_{K_j}(X))$
```

```
proc KDM(j, f)                                            Games G_2
    $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, C)$,  $X \leftarrow \mathsf{En}\big(e, 0^n\big)$
    return $\big(F, d, \mathcal{E}_{K_j}\big(0^{|X|}\big)\big)$
```

```
proc KDM(j, f)                                            Games G_3
    $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, \mathrm{ID})$,  $X \leftarrow \mathsf{En}(e, C(0^n))$
    return $\big(F, d, \mathcal{E}_{K_j}\big(0^{|X|}\big)\big)$
```

Fig. 19. Code for proof of Theorem 15.

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.

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 $\mathbf{Adv}_{\Pi'}^{\mathrm{kdm}}(\mathcal{A}, k) \le \mathbf{Adv}_{\Pi}^{\mathrm{kdm}}(\mathcal{B}, k) + 2\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv}, \Phi_{\mathrm{size}}}(\mathcal{B}', k)$. 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

$$\mathbf{Adv}_{\Pi}^{\mathrm{kdm}}(\mathcal{B}, k) = \Pr[G_1^{\mathcal{A}}(k) \Rightarrow 1] - \Pr[G_2^{\mathcal{A}}(k) \Rightarrow 1]$$

where games $G_0 - G_3$ are described in Fig. 19. (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

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv}, \Phi_{\mathrm{size}}}(\mathcal{B}', k) = \Pr[G_0^{\mathcal{A}}(k) \Rightarrow 1] - \Pr[G_1^{\mathcal{A}}(k) \Rightarrow 1]$$

Otherwise, if $c = 1$ then

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv}, \Phi_{\mathrm{size}}}(\mathcal{B}', k) = \Pr[G_2^{\mathcal{A}}(k) \Rightarrow 1] - \Pr[G_3^{\mathcal{A}}(k) \Rightarrow 1]$$

Summing up, we have

$$\mathbf{Adv}_{\Pi}^{\mathrm{kdm}}(\mathcal{B}, k) + 2\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv}, \Phi_{\mathrm{size}}}(\mathcal{B}', k) = \Pr[G_0^{\mathcal{A}}(k) \Rightarrow 1] - \Pr[G_3^{\mathcal{A}}(k) \Rightarrow 1] = \mathbf{Adv}_{\Pi'}^{\mathrm{kdm}}(\mathcal{A}, k) \ .$$

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 $\mathbf{Adv}_{\Pi'}^{\mathrm{kdm}}(\mathcal{A}, k) \le \mathbf{Adv}_{\Pi}^{\mathrm{kdm}}(\mathcal{B}, k) + 2Q\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv}, \Phi_{\mathrm{size}}}(\mathcal{B}', k)$. 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

Scheme	# of AES calls	# of bits encrypted by 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	$O(r \log r)$	$k \cdot \max\{p \cdot \ell, s\}$	$O(kr \log r)$

Fig. 20. 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 [59]. 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.

$(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f_0)$ 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 [15] 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.

Acknowledgments

This research was supported under NSF grants CNS 0904380, CNS 1228828, CNS 1228890, CNS-1116800 and CCF 0915675; many thanks to the NSF for their continuing support. The paper was refined and first presented while Rogaway was a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the program *Semantics and Syntax: A Legacy of Alan Turing*, co-organized by Nigel Smart. Many thanks to Tommi Meskanen, Valtteri Niemi, and Noora Nieminen, all of the University of Turku, Finland, who pointed out a definitional issue ("Invalid queries," Section 3.8) that needed fixing. Thanks to Shafi Goldwasser, Kenny Patterson, Samuel Ranellucci, Thomas Ristenpart, and Thomas Schneider for their comments, suggestions, and pleasant interactions.

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. 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. 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. 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. 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, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Computer and Communications Security (CCS'12)*. ACM, 2012.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. O. Goldreich. Cryptography and cryptographic protocols. Manuscript, June 9 2001.

22. 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.
23. 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.
24. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In N. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 289–306. Springer, Apr. 2008.
25. A. Herzberg and H. Shulman. Secure guaranteed computation. Cryptology ePrint Archive, Report 2010/449, 2010.
26. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.
27. 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.
28. 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.
29. Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In P. Widmayer, F. Ruiz, R. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002.
30. 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.
31. S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, report 2011/272, October 25 2011.
32. S. Kamara and L. Wei. Special-purpose garbled circuits. Manuscript, 2012.
33. J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In E. Al-Shaer, A. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 485–492. ACM Press, Oct. 2010.
34. 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.
35. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.
36. 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.
37. 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.
38. L. Kruger, S. Jha, E. 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.
39. 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.
40. 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.
41. 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.
42. 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.
43. 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 287–302. USENIX Association, 2004.
44. 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.
45. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM STOC*, pages 590–599. ACM Press, July 2001.
46. 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.
47. A. Paus, A. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In M. Abdalla, D. Pointcheval, P. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 89–106. Springer, June 2009.
48. K. Pietrzak. A leakage-resilient mode of operation. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer, Apr. 2009.
49. B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):12–19, 2002.
50. B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Dec. 2009.

51. P. Rogaway. The round complexity of secure protocols. MIT Ph.D. Thesis, 1991.
52. A. 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.
53. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In E. Al-Shaer, A. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 463–472. ACM Press, Oct. 2010.
54. T. Schneider. Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, 2008.
55. 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.
56. T. Schneider. *Engineering Secure Two-Party Computation Protocols.* Springer-Verlag Berlin Heidelberg, 2012.
57. 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.
58. J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In P. Ning, S. di Vimercati, and P. Syverson, editors, *ACM CCS 07*, pages 519–528. ACM Press, Oct. 2007.
59. L. Valiant. Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203. ACM, 1976.
60. A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.
61. A. Yao. Protocols for secure computations. In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.

A Related Work

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

RANDOMIZED ENCODINGS. Loosely related to garbling schemes, *randomized encodings* (initially *randomized polynomials*) begin with Ishai and Kushilevitz [28] and continue, with many definitional variants, in work by Applebaum, Ishai, Kushilevitz, and others [2, 4–7, 29, 30, 53]. 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,[16] for example, that every bit of $F(x, r)$ depends on at most one bit of x, a property that has been called *decomposability* [30]. Proven realizations meeting these requirements [4, 5] do not closely resemble conventional realizations of garbled circuits [41, 46].

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 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 [53]. (Despite identical names, this definition is different from that above, and different again from the decomposable randomized encodings of [30], 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

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

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 has effectively assumed a projective scheme, a boolean circuit as input, and prv.sim security over $\Phi_{\rm size}$. 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 dynamic setting, 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 [20], 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 [31]. 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 [49, Section 2.3] already remarks that "In this form the representation reveals nothing but the wiring of the circuit". Later, Paus, Sadeghi, and Schneider [47] 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 $\Phi_{\rm topo}$.

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 [45], DFAs [58], and ordered binary decision diagrams [38]. 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* [32], 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 dynamic versions of their definitions, Ind2 and Sim2, and an unforgeability notion, Unf2. These definitions resemble the weaker form of the dynamic-security definitions (prv1, obv1, and aut1) mentioned in our Introduction and the subject of separate work.

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 [17]. For the protocol to make sense with respect to the definitions given, the latter should be reinterpreted as applying to structured circuits.

B Universal Circuits

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 $\Omega(q 2^q)$ circuits of q gates. Valiant [59] 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 are known [36, 52, 54], but their asymptotic size is larger.