The proceedings version of this paper appears in *Asiacrypt 2012* [4]. This is the full paper.

Adaptively Secure Garbling with Applications to One-Time Programs and Secure Outsourcing

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 2, 2012

Abstract. Standard constructions of garbled circuits provide only *static* security, meaning the input x is not allowed to depend on the garbled circuit F. But some applications—notably *one-time programs* (Goldwasser, Kalai, and Rothblum 2008) and *secure outsourcing* (Gennaro, Gentry, Parno 2010)—need *adaptive* security, where x may depend on F. We identify gaps in proofs from these papers with regard to adaptive security and suggest the need of a better abstraction boundary. To this end we investigate the adaptive security of *garbling schemes*, an abstraction of Yao's garbled-circuit technique that we recently introduced (Bellare, Hoang, Rogaway 2012). Building on that framework, we give definitions encompassing *privacy*, *authenticity*, and *obliviousness*, with either *coarse-grained* or *fine-grained* adaptivity. We show how adaptively secure garbling schemes support simple solutions for one-time programs and secure outsourcing, with privacy being the goal in the first case and obliviousness and authenticity the goal in the second. We give transforms that promote static-secure garbling schemes to adaptive-secure ones. Our work advances the thesis that conceptualizing garbling schemes as a first-class cryptographic primitive can simplify, unify, or improve treatments for higher-level protocols.

Keywords: adaptive adversaries, adaptive security, garbled circuits, garbling schemes, one-time programs, secure outsourcing, verifiable computing, Yao's protocol.

Table of Contents

1 Introduction

Overview. Yao's garbled-circuit technique [13, 15, 22, 24, 25] has been extremely influential, engendering an enormous number of applications. Yet, at least in its conventional form, the technique provides only *static* security.[3] Some applications, notably one-time programs [17] and secure outsourcing [12], require *adaptive* security.[4] In such cases Yao's technique can be enhanced in *ad hoc* ways, the resulting protocol then incorporated into the higher-level application.

This paper provides a different approach. We create an abstraction for the goal of *adaptively secure garbling.* Via a single abstraction, we support a variety of applications in a simple and modular way. Let's look at two of the applications that motivate our work.

Two applications. *One-time programs* are due to Goldwasser, Kalai, and Rothblum (GKR) [17]. The authors aim to compile a program into one that can be executed just once, on an input of the user's choice. Unachievable in any "standard" model of computation, GKR assume a model providing what they call *one-time memory.* Their solution makes crucial use of Yao's garbled-circuit technique. Recognizing that this does not support adaptive queries, GKR embellish the method by a technique involving output-masking and n-out-of-n secret sharing.

In a different direction, *secure outsourcing* was formalized and investigated by Gennaro, Gentry, and Parno (GGP) [12]. Here a *client* transforms a function f into a function F that is handed to a *worker.* When, later, the client would like to evaluate f at x (and various such inputs may arise), he should be able to quickly map x to a garbled input X and give this to the worker, who will compute and return $Y = F(X)$. The client must be able to quickly reconstruct from this $y = f(x)$. He should be sure that the correct value was computed—the computation is *verifiable*—while the server shouldn't learn anything significant about x, including $f(x)$.[5] GGP again make use of circuit garbling, and they again realize that they need something from it—its authenticity—that is a *novum* for this domain.

Issues. Assuming the existence of a one-way function, GKR [17] claim that their construction turns a (statically-secure) garbled circuit into a secure one-time program. We point to a gap in their proof, namely, the absence of a reduction showing that their simulator works based on the one-way function assumption. By presenting an example of a statically-secure garbled circuit that, under their transform, yields a program that is not one-time, we also show that the gap cannot be filled without changing either the construction or the assumption. The problem is that the GKR transform fails to ensure adaptive security of garbled circuits under the stated assumption.

Lindell and Pinkas (LP) [21] prove static security of a version of Yao's protocol assuming a semantically secure encryption scheme satisfying some extra properties (an elusive and efficiently verifiable range). GGP [12] build a one-time outsourcing scheme from the LP protocol, claiming to prove its security based on the same assumption as used in LP. Again, there is a gap in this proof arising from an implicit assumption of adaptive security of the LP construction.

[3] Of course conventional garbled circuits certainly *can* be used to build schemes, say for multiparty computation, meeting adaptive security notion; we are only asserting that the standard and well-known methods fail, by themselves, to provably achieve adaptive definitions of garbling-scheme security.

[4] In speaking of adversaries or security, *non-adaptive* and *dynamic* are common synonyms for what we are here calling *static* and *adaptive*.

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

We do not believe these are major problems for either work. In both cases, alternative ways to establish the the authors' main results already existed. Goyal, Ishai, Sahai, Venkatesan and Wadia [18] present an unconditional one-time compiler (no complexity-theoretic assumption is used at all), while Chung, Kalai and Vadhan [10] present secure outsourcing schemes based solely on FHE (garbled circuits are not employed). Our interpretation of the stated gaps is that they are symptoms of something else—a missing abstraction boundary. As recently argued by Bellare, Hoang and Rogaway (BHR) [5], it is useful and simplifying to see garbling not just as a technique, but as a first-class primitive. To do so, our earlier work defines syntax and security notions for *garbling schemes*, provides proven-correct solutions, then solves some example higher-level problems by employing a garbling scheme that satisfies the appropriate definition. But the security notions of BHR do not go far enough to handle what GKR or GGP need, since BHR deals only with static notions of security. The applications we point to motivate the study of adaptive security for garbling schemes, while the gaps indicate that the issues may be more subtle than recognized.

Of course we communicated our findings to the GKR and GGP authors. GKR responded after a few weeks with an updated manuscript [16]. It modifies the claim from their original paper [17] to now claim that their transform works under the stronger assumption of a sub-exponentially hard one-way function. (This allows "complexity-leveraging," where a static adversary can guess the input that will be used by an adaptive adversary with a probability that, although exponentially-small, is enough under the stronger assumption.) GGP responded to acknowledge the gap and suggest that they would address it by assuming the LP construction, or some related realization of Yao's idea, already provides adaptive security.

Definitions. We now discuss our contributions in more depth. We start from the abstraction of a garbling scheme—the raw syntax—introduced by BHR [5]. That work gave multiple definitions sitting on top of this syntax, but all were for static adversaries, in the sense that the function f to garble and its input x are selected at the same time. We extend the definitions to adaptive ones, considering two flavors of adaptive security. With *coarse-grained* adaptive security the input x can depend on the garbled function F but x itself is atomic, provided all at once. With *fine-grained* adaptive security not only may x depend on the garbled function F, but individual bits of x can depend on the "tokens" the adversary has so-far learned.[6] We will see that coarse-grained adaptive security is what's needed for GGP's approach to secure outsourcing, while fine-grained adaptive security is what's needed for GKR's approach to one-time programs.

Orthogonal to adaptive security's granularity are the security aims themselves. Following BHR, we consider three different notions: privacy, obliviousness, and authenticity. This gives rise to nine different security notions: {prv, obv, aut} × {static, coarse, fine}. We compactly denote these prv, prv1, prv2, obv, obv1, obv2, aut, aut1, aut2. Informally, when a function f gets transformed into a garbled function F, an encoding function e, and a decoding function d, privacy ensures that F, d, and $X = e(x)$ don't reveal anything beyond $y = f(x)$ that shouldn't be revealed; obliviousness ensures that F and X don't reveal even y; and authenticity ensures that F and X don't enable the computation of a valid $Y \neq F(X)$. Privacy is the classical requirement, while obliviousness and authenticity are motivated by the application to secure outsourcing.

[6] Fine-grained adaptive security requires the garbling scheme be *projective*: the garbled version of each $x = x_1 \cdots x_n \in \{0,1\}^n$ must be $(X_1^{x_1}, \ldots, X_n^{x_n})$ for some vector of $2n$ strings $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$. Typical garbling schemes have this structure.

Our primary definitions for adaptive secrecy (prv1, prv2, obv1, obv2) are simulation-based. In Appendix B we give indistinguishability-based counterparts as well. For static security this was already done by BHR, but it was not clear how to lift those definitions to the adaptive setting.

Relations. We explore the provable-security relationships among our definitions. As expected, the simulation-based definitions imply indistinguishability-based ones (namely, prv1 $\Rightarrow$ prv1.ind, prv2 $\Rightarrow$ prv2.ind, obv1 $\Rightarrow$ obv1.ind, and obv2 $\Rightarrow$ obv2.ind). But none of the converse statements hold. BHR had earlier shown that, for the static setting, the converse statements *do* hold as long as the associated side-information function[7] is efficiently invertible.[8] In contrast, we show that, for adaptive privacy, this condition still won't guarantee equivalence of simulation-based and indistinguishability-based notions. (For obliviousness, it is true that obv1.ind $\Rightarrow$ obv1 and obv2.ind $\Rightarrow$ obv2 if Φ is efficiently invertible.) The results are our main reason to focus on simulation-based definitions for adaptive privacy. Appendix C paints a complete picture of the relations among our basic definitions. Apart from the trivial relations (prv2 $\Rightarrow$ prv1 $\Rightarrow$ prv, obv2 $\Rightarrow$ obv1 $\Rightarrow$ obv, and aut2 $\Rightarrow$ aut1 $\Rightarrow$ aut) nothing implies anything else.

Achieving adaptive security. Basic garbling-scheme constructions [5, 13, 15, 22] either do not achieve adaptive security or present difficulties in proving adaptive security that we do not know how to overcome. One could give new constructions and directly prove them xxx1 or xxx2 secure, for xxx $\in \{\mathrm{prv}, \mathrm{obv}, \mathrm{aut}\}$. An alternative is to provide generic ways to transform statically secure garbling schemes to adaptively secure ones. Combined with results in BHR [5], this would yield adaptively-secure garbling schemes.

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

To give transforms that work we make two steps, first passing from static security to coarse-grained adaptive security, and thence to fine-grained adaptive security. We design these transformations first for privacy (prv-to-prv1, prv1-to-prv2) and then for simultaneously achieving all three goals (all-to-all1 and all1-to-all2). Our prv-to-prv1 transform uses a one-time-padding technique from [18], while our prv1-to-prv2 transform uses the secret-sharing component of OMSS.

Applications. We treat the two applications that motivated this work, one-time programs and secure outsourcing. We show that adaptive garbling schemes yield these applications easily and directly. Specifically, we show that a prv2 projective garbling scheme can be turned into a secure one-time program by simply putting the garbled inputs into the one-time memory. We also show how to easily turn an obv1+aut1 secure garbling scheme into a secure one-time outsourcing scheme. (GGP [12] show how to lift one-time outsourcing schemes to many-time ones using FHE.) The simplicity of these transformations underscores our tenet that abstracting garbling schemes and treating adaptive security for them enables modular and rigorous applications of the garbled-

[7] The side-information function Φ captures that about f one allows to be revealed in its garbled counterpart F. When f encodes a circuit, the side-information might be its size (gate count), input length, and output length.

[8] Side-information function Φ is efficiently invertible if there is a PT algorithm M that, on input $\phi = \Phi(f)$ for some f, outputs f' such that $\Phi(f') = \phi$. If Φ is efficiently invertible then obv and obv.ind are equivalent. Each garbling scheme has a procedure ev that describes how to interpret a string f as a function $\mathsf{ev}(f, \cdot) : \{0,1\}^n \to \{0,1\}^m$. The pair (Φ, ev) is efficiently invertible if there is a PT algorithm M that, on input $\phi = \Phi(f)$ and $y = f(x)$ for some f and some x, outputs an f' and x' such that $\Phi(f') = \phi$ and $f'(x') = y$. If (Φ, ev) is efficiently invertible then prv and prv.ind are equivalent.

Transform	**Model**	**Cost**	**See**
prv-to-prv1	standard model	$\|F\| + \|d\| + \|X\|$	Theorem 2
prv1-to-prv2	standard model	$(n+1)\,\|X\|$	Theorem 3
all-to-all1	standard model	$\|F\| + \|d\| + \|X\| + k$	Theorem 8
all1-to-all2	standard model	$(n+1)\,\|X\|$	Theorem 9
rom-prv-to-prv1	random-oracle model	$\|X\| + k$	Theorem 4
rom-prv1-to-prv2	random-oracle model	$\|X\| + nk$	Theorem 5
rom-all-to-all1	random-oracle model	$\|X\| + 2k$	Theorem 10
rom-all1-to-all2	random-oracle model	$\|X\| + nk$	Theorem 11

Fig. 1. Achieving adaptive security. The name of each transform specifies its relevant property. The word all means that prv, obv, and aut are all upgraded. Column "Cost" specifies the length of the garbled input in the constructed scheme in terms of the lengths of the input scheme's garbled function F, decoding function d, garbled input X, the number of input bits n, and security parameter k.

circuit technique. Basing the applications on garbling schemes also allows instantiations to inherit efficiency features of future schemes.

Applying our prv-to-prv1 and then prv1-to-prv2 transforms to the prv-secure garbling scheme of BHR [5] yields a prv2-secure scheme based on any one-way function. Combining this with the above yields one-time programs based on one-way functions, recovering the claim of GKR [17]. Similarly, applying our all-to-all1 transform to the obv+aut secure scheme of BHR yields an obv1+aut1 secure garbling scheme based on a one-way function, and combining this with the above yields a secure one-time outsourcing scheme based on one-way functions.

Efficiency. Let us say a garbling scheme has *short* garbled inputs if their length depends only on the security parameter k, the length n of f's input, and the length m of f's output. It does not depend on the length of f. The statically-secure schemes of BHR, as with all classical garbled-circuit constructions, have short garbled inputs. But our prv-to-prv1 and all-to-all1 transforms result in long garbled inputs. In the ROM (random-oracle model) we are able to provide schemes producing short garbled inputs, as illustrated in Fig. 1. Constructing an adaptively secure garbling scheme with short garbled inputs under standard assumptions remains open.[9]

Short garbled inputs are particularly important for the application to secure outsourcing, for in their absence the outsourcing scheme may fail to be non-trivial. (Non-trivial means that the client effort is less than the effort needed to directly compute the function [12].) In particular, the one-time outsourcing scheme we noted above, derived by applying all-to-all1 to BHR, fails to be non-trivial. ROM schemes do not fill the gap because of the use of FHE in upgrading one-time schemes to many-time ones [12]. Thus, a secure and non-trivial instantiation of the GGP method is still lacking. (However, as we have noted before, non-trivial secure outsourcing may be achieved by entirely different means [10].)

Further related work. Applebaum, Ishai, and Kushilevitz [1] investigate ideas similar to obliviousness and authenticity. Their approach to obtaining these ends from privacy can be lifted and formalized in our settings; one could specify transforms prv1-to-all1 and prv2-to-all2, effectively handling the constructive story "horizontally" instead of "vertically." The line of work on *random-*

[9] Intuitively, the underlying encryption appears to need some kind of security against selective-opening attacks that reveal decryption keys (SOA-K), and this is hard without long keys [3]. However, there is some hope because full-fledged SOA-K security does not seem to be needed.

ized encodings that the same authors have been at the center of provides an alternative to garbling schemes [19] but lacks the granularity to speak of adaptive security.

Independent and concurrent work by Kamara and Wei (KW) investigates the garbling what they call *structured circuits* [20] and, in the process, give definitions somewhat resembling prv1, obv1, and aut1, although circuit-based, not function-hiding, and not allowing the adversary to specify the initial function. KW likewise draw motivation from GKR and GGP, indicating that, in these two settings, the adversary can choose the inputs to the computation as a function of the garbled circuit, motivating adaptive notions of privacy and unforgeability.

Independent and concurrent work by Applebaum, Ishai, Kushilevitz and Waters [2] provides a definition of adaptive security for randomized encodings of functions that corresponds to our coarse-grained privacy. They provide a ROM construction with short keys achieving this notion. They also provide negative results on the length of keys for adaptive security in the standard model.

2 Framework

We now review the syntactic framework of garbling schemes from our earlier work [5]. See Appendix A for basic notation, including conventions for randomized algorithms, code-based games, and circuits.

Garbling schemes. A *garbling scheme* [5] 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 rest 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. The values $n = f.n$ and $m = f.m$ are efficiently computable from f. 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* $X = \mathsf{En}(e, x)$. String F describes a *garbled function*, $\mathsf{Ev}(F, \cdot)$, that maps a 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* $y = \mathsf{De}(d, Y)$. The correctness requirement is that 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 $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = \mathsf{ev}(f, x)$. We also require that e and d depend only on k, $f.n$, $f.m$, $|f|$ and the random coins r of Gb. This non-degeneracy requirement excludes trivial solutions.

A common design 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. Let $\mathsf{GS}(\text{proj})$ denote the set of all projective garbling schemes, and let $\mathsf{GS}(\mathsf{ev})$ be the set of all garbling schemes whose evaluation function is ev.

Boolean circuits arise often in this work. We recall in Appendix A.3 a formalization for them following [5], including the definition of the canonical circuit-evaluation function. We say that $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ is a circuit-garbling scheme if ev is the canonical circuit-evaluation function.

Side-information functions. A garbled circuit might reveal the size of the circuit that is being garbled, its topology, the original circuit itself, or something else. The information that we allow to be revealed is captured by a *side-information function*, Φ, which deterministically maps f to a string $\phi = \Phi(f)$. We parameterize our advantage notions by Φ. We require that $f.n$, $f.m$ and $|f|$ be easily

```
proc GARBLE($f, x$)                                    $\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}$
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($f$)                                       $\mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$
if $b = 1$ then
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
else
    $(F, d) \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$
return $(F, d)$

proc INPUT($x$)
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $b = 1$ then $X \leftarrow \mathsf{En}(e, x)$
else
    $y \leftarrow \mathsf{ev}(f, x)$, $X \leftarrow \mathcal{S}(y, 1)$
return $X$
```

```
proc GARBLE($f$)                                       $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$
$n \leftarrow f.n$;  $Q \leftarrow \emptyset$;  $\tau \leftarrow \varepsilon$
if $b = 1$ then
    $(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$
else
    $(F, d) \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$
return $(F, d)$

proc INPUT($i, c$)
if $i \notin \{1, \ldots, n\} \setminus Q$ then return $\bot$
$x_i \leftarrow c$;  $Q \leftarrow Q \cup \{i\}$
if $|Q| = n$ then
    $x \leftarrow x_1 \cdots x_n$;  $y \leftarrow \mathsf{ev}(f, x)$;  $\tau \leftarrow y$
if $b = 1$ then $X_i \leftarrow X_i^{x_i}$
else $X_i \leftarrow \mathcal{S}(\tau, i, |Q|)$
return $X_i$
```

Fig. 2. Three kinds of privacy: prv, prv1, prv2. Games to define the static, coarse-grained, and fine-grained privacy of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Each game starts with INITIALIZE() that samples a bit $b \twoheadleftarrow \{0,1\}$, and its FINALIZE($b'$) returns the predicate $(b = b')$. Notation $s \twoheadleftarrow S$ denotes uniform sampling from a finite set.

determined from $\phi = \Phi(f)$. Side-information function Φ_{size} maps a circuit $f = (n, m, q, A, B, G)$ to (n, m, q), while Φ_{topo} maps f to $f^- = \mathrm{Topo}(f) = (n, m, q, A, B)$ and Φ_{circ} is the identity, $\Phi_{\text{circ}}(f) = f$.

SIZES. A cost metric of interest is the length of the garbled input. We say that garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ has *short garbled inputs* if there is a polynomial s such that $|\mathsf{En}(e, x)| \le s(k, f.n, f.m)$ for all $k \in \mathbb{N}$, $f \in \{0,1\}^*$, $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$, and $x \in \{0,1\}^{f.n}$. Let T be a transform that maps a garbling scheme $\mathcal{G}$ to a garbling scheme $\mathsf{T}[\mathcal{G}]$. We say that T *preserves* short garbled inputs if $\mathsf{T}[\mathcal{G}]$ has short garbled inputs when $\mathcal{G}$ does.

Typical Yao-style constructions, including Garble1 and Garble2 [5], have short garbled inputs. But they are only statically-secure. Keeping garbled inputs short seems challenging for adaptive security in the standard model.

3 Adaptive Privacy and One-Time Programs

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

3.1 Definitions for adaptive privacy

On the left-hand panel of Fig. 2 we review the defining game for the privacy notion from BHR [5]. The adversary is *static*, in the sense it must commit to its initial function f and its input x at the same time. Thus the latter is independent of the garbled function F (and the decoding function d) derived from f. It is natural to consider stronger privacy notions, ones where the adversary obtains F and *then* selects x. Two formulations for this are specified in Fig. 2. We call these *adaptive* security. The notion in the middle panel, denoted by prv1, this paper, is *coarse-grained* adaptive security.

The notion in the right panel, denoted by prv2, is *fine-grained* adaptive security. This notion is only applicable for projective garbling schemes.

In detail, let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme and let Φ be a side-information function. We define three simulation-based notions of privacy via the games $\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}$, $\mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$, and $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$ of Fig. 2. Here $\mathcal{S}$, the *simulator*, is an always-terminating algorithm that maintains state across invocations. An adversary $\mathcal{A}$ interacting with any of these games must make exactly one GARBLE query. For game Prv1 it is followed by a single INPUT query. For game Prv2 it is followed by multiple INPUT queries. There, the garbling scheme must be projective. The advantage the adversary gets is defined by

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

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

Let us now explain the three games, beginning with static privacy. Here we let the adversary select f and x and we do one of two things: garble f to make (F,e,d) and encode x to make X, giving the adversary (F,X,d); or, alternatively, we ask the simulator produce a "fake" (F,X,d) based only on the security parameter k, the partial information $\Phi(f)$ about f, and the output $y = \mathsf{ev}(f,x)$. The adversary will have to guess if the garbling was real or fake.

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

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

3.2 The OMSS transform

In the process of constructing one-time programs from garbled circuits, GKR [17] recognize the need for adaptive privacy of the garbled circuits. Their construction incorporates a technique to provide it. This technique is easily abstracted to provide, in our terminology, a transform that aims to convert a projective, prv garbling scheme into a projective, prv2 garbling scheme. Instead of garbling f we pick $r \twoheadleftarrow \{0,1\}^m$ and garble the circuit g defined by $g(x) = f(x) \oplus r$ for every $x \in \{0,1\}^n$ where $n = f.n$ and $m = f.m$. Then we secret share r as $r = r_1 \oplus \cdots \oplus r_n$ and include r_i in the i-th token, so that evaluation reconstructs r and it can be xored back at decoding time to

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

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

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

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

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

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

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

Fig. 3. OMSS definition (top). Scheme $\mathsf{OMSS}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_2, \mathsf{Ev}_2, \mathsf{ev})$ where $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. **OMSS counterexample (bottom).** The garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ obtained from $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ is prv secure when $\mathcal{G}'$ is, but $\mathsf{OMSS}[\mathcal{G}]$ is not prv2 secure. We assume the decoding rule of $\mathcal{G}'$ is vacuous, a feature inherited by $\mathcal{G}$. We are letting $\overline{v}$ denote the bitwise complement of a string v.

recover $\mathsf{ev}(f, x)$ as $\mathsf{ev}(g, x) \oplus r$. Intuitively, this should work because the simulator can garble a dummy constant function with random output s and does not have to commit to r until it gets the target output value y of f and needs to provide the last token, at which point it can pick $r = s \oplus y$ so that the final output is y as desired [17]. Just the same, we show by counterexample that the OMSS does not work, in general, to convert a prv secure scheme to a prv2 secure one: we present a prv secure $\mathcal{G}$ such that $\mathsf{OMSS}[\mathcal{G}]$ is not prv2 secure.[10]

Now proceeding formally, we associate to circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\text{proj})$ the circuit-garbling scheme $\mathsf{OMSS}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_2, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}(\text{proj})$ defined at the top of Fig. 3. For simplicity we are assuming that the decoding rule d in $\mathcal{G}$ is always vacuous, meaning $d = \varepsilon$. (We do not need non-trivial d to achieve privacy [5], and this lets us stay closer to GKR [17], whose garbled circuits have no analogue of our decoding rule.) In the code, $g(\cdot) \leftarrow f(\cdot) \oplus r$ means that we construct from f, r a circuit g such that $\mathsf{ev}(g, x) = \mathsf{ev}(f, x) \oplus r$ for all $x \in \{0,1\}^{f.n}$. (Note we can do this in such a way that $\Phi_{\text{topo}}(g) = \Phi_{\text{topo}}(f)$.)

The claim under consideration is that if $\mathcal{G}$ is prv secure relative to $\Phi = \Phi_{\text{topo}}$ then $\mathcal{G}_2$ is prv2 secure relative to $\Phi = \Phi_{\text{topo}}$. To prove this, we would need to let $\mathcal{A}_2$ be an arbitrary PT adversary and build a PT simulator $\mathcal{S}_2$ such that $\mathbf{Adv}_{\mathcal{G}_2}^{\text{prv2}, \Phi, \mathcal{S}_2}(\mathcal{A}_2, \cdot)$ is negligible. GKR suggest a plausible strategy for the simulator that, in particular, explains the intuition for the transform. We present here our understanding of this strategy adapted to our setting. In its first phase the simulator $\mathcal{S}_2$

[10] In Section 3.7 we extend this to show that the OMSS-based one-time compiler of GKR [17] is not secure. The underlying technical issues, are, however in our view easier understood in terms of garbling, divorced from the application to one-time programs.

has input $1^k, \phi, 0$ where $\phi = \Phi(f)$, with f being the query made by the adversary to Garble. Simulator $\mathcal{S}_2$ picks $s \twoheadleftarrow \{0,1\}^n$ and lets f_s be the circuit that has output s on all inputs and $\Phi_{\mathrm{topo}}(f_s) = \phi$. It also picks random m-bit strings $s_1, \ldots, s_n$ and a random input $w \twoheadleftarrow \{0,1\}^n$. It lets $(G, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), \varepsilon) \twoheadleftarrow \mathsf{Gb}(1^k, f_s)$ and returns G to the adversary. In the second phase, when given input τ, i, j, for $j \leq n-1$, the simulator lets $T_i \leftarrow (X_i^{w_i}, s_i)$ and returns T_i to the adversary as the token for bit i of the input. In the case that $j = n$, the simulator obtains (from τ as per our game) the output $y = \mathsf{ev}(f, x)$ of the function on input x, the latter defined by the adversary's queries to Input. It now resets $s_i = y \oplus s \oplus s_i \oplus s_1 \oplus \cdots \oplus s_n$ and returns $(X_i^{w_i}, s_i)$, so that evaluation of the garbled function indeed results in output y.

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

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

To carry this out, we start with an arbitrary projective circuit-garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev})$ assumed to be prv secure relative to $\Phi = \Phi_{\mathrm{topo}}$. We then transform it into the projective circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ shown at the bottom of Fig. 3. The idea is as follows. We choose m-bit random shares V_i^0, V_i^1 for every $i \leq n$, and distribute them to the tokens. Next, choose a "poisoned" point $v = v_1 \cdots v_n$ at random, and append it to the garbled function, making it trivial for an adaptive adversary to query $x = v$. Since v is random, a static adversary can guess v with probability only 2^{-n}. To make sure this probability is negligible in terms of k, we only do the following trick if $n \geq k$. Let V be the encryption of $\mathsf{ev}(g, \overline{v})$ by using the one-time pad constructed from the shares corresponding to v, namely, the pad is the checksum of $V_1^{v_1}, \ldots, V_n^{v_n}$. Append V to the garble function as well. So if the adversary queries $x = v$ then it will learn $\mathsf{ev}(g, \overline{v})$ in addition to $\mathsf{ev}(g, v)$; while if $x \neq v$ then the shares the adversary receives won't allow it to decrypt V. The following proposition says that $\mathcal{G}$ continues to be prv secure but an attack shows that $\mathsf{OMSS}[\mathcal{G}]$ is not prv2 secure. (The proof shows it is in fact not even prv1 secure.)

Proposition 1 Let ev be the canonical circuit-evaluation function. Assume $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}, \mathsf{Ev}', \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj})$ and let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{proj})$ be the garbling scheme shown at the bottom of Fig. 3. Then (1) $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj})$, but (2) $\mathsf{OMSS}[\mathcal{G}] \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$.

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

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

proc $\mathsf{En}_1(e_1, x)$
$(e, d', F') \leftarrow e_1$,   $X \leftarrow \mathsf{En}(e, x)$
return $(X, d', F')$

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

proc $\mathsf{De}_1(d_1, Y_1)$
$(Y, d') \leftarrow Y_1$,  $d \leftarrow d_1 \oplus d'$
return $\mathsf{De}(d, Y)$
```

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

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

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

Fig. 4. Transform prv-to-prv1 (top): Scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$ obtained by applying the prv-to-prv1 transform to $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi)$. **Transform prv1-to-prv2 (bottom):** Projective garbling scheme $\mathcal{G}_2 = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv2}, \Phi)$ obtained by applying the prv1-to-prv2 transform to projective garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$.

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

Let $\mathcal{S}_2$ be *any* PT simulator and consider game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$. We claim that $\mathcal{A}_2(1^k)$ returns 1 with probability 1 if the challenge bit b in the game is 1. This is because in this case we have $y = \mathsf{ev}(f_{R_0,R_1}, v)$ and $y' = r \oplus \mathsf{ev}(f_{R_0,R_1}, \overline{v})$ so by definition of f_{R_0,R_1} we have $y \oplus y' \oplus r = R_0 \oplus R_1$. Next we claim that $\mathcal{A}_2(1^k)$ returns 1 with probability at most 1/2 if the challenge bit b is 0. (We emphasize that this claim is made regardless of the strategy of the simulator, showing that no simulator could possibly do well.) In the first phase, the simulator $\mathcal{S}_2$ is given $1^k, \Phi_{\mathrm{topo}}(f), 0$ as input and can obtain no information on R_0 or R_1 beyond their length because the topology of f_{R_0,R_1} is by construction independent of R_0, R_1. In the second phase, the only useful information that the sender gets is $y = \mathsf{ev}(f_{R_0,R_1}, v)$. It thus learns R_{v_1} but it has no information about R_{1-v_1} and thus the probability that the $y' \oplus r$ computed by the adversary equals $y \oplus R_0 \oplus R_1$ is at most 1/2. □

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

3.3 Achieving prv1 security

We now describe a transform $\mathsf{prv\text{-}to\text{-}prv1}$ that successfully turns a prv secure circuit garbling scheme into a prv1 secure one. Combined with established results [5], this yields prv1 secure schemes based on standard assumptions. The idea (cf. [18]) is to use one-time pads to mask F and d, and then append the pads to X. This will ensure that the adversary learns nothing about F and d until it fully specifies function f and x. Given a (not necessarily projective) garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, the $\mathsf{prv\text{-}to\text{-}prv1}$ transform returns the garbling scheme $\mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}] = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ at the top of Fig. 4. We claim:

Theorem 2. For any Φ, if $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi)$ then $\mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}] \in \mathsf{GS}(\mathrm{prv1}, \Phi)$.

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

Transform $\mathsf{prv\text{-}to\text{-}prv1}$ does not require the starting scheme $\mathcal{G}$ to be projective. However, it is important that if $\mathcal{G}$ is projective, so is $\mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}]$. Seeing this requires a slight re-interpretation of certain quantities in the algorithms at the top of Fig. 4. Specifically, e will now have the form $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ and Gb_1 will let $e_1 = ((X_1^0, d', F'), (X_1^1, d', F'), X_2^0, X_2^1, \ldots, X_n^0, X_n^1)$. Also X in En_1 will have the form $(X_1, \ldots, X_n)$ and En_1 will return $((X_1, d', F'), X_2, \ldots, X_n)$.

A potentially simpler transform of a prv secure garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ into a prv1 secure garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ is as follows. Algorithm $\mathsf{Gb}_1(1^k, f)$ lets $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f)$ and returns $(\varepsilon, (e, F, d), \varepsilon)$. Let $\mathsf{En}_1((e, F, d), x) = (\mathsf{En}(e, x), F, d)$. Let $\mathsf{Ev}_1(\varepsilon, (X, F, d)) = (\mathsf{Ev}(F, X), d)$. Let $\mathsf{De}_1(\varepsilon, (Y, d)) = \mathsf{De}(d, Y)$. This works, but the scheme does not meet the non-degeneracy requirement we have imposed in Section 2. The $\mathsf{prv\text{-}to\text{-}prv1}$ transform can be seen as a way to effectively implement this trivial transform while avoiding degeneracy.

3.4 Achieving prv2 security

Next we show how to transform a prv1 scheme into a prv2 one. Formally, given a projective garbling scheme $\mathcal{G} = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$, the $\mathsf{prv1\text{-}to\text{-}prv2}$ transform returns the projective garbling scheme $\mathsf{prv1\text{-}to\text{-}prv2}[\mathcal{G}] = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev})$ shown at the bottom of Fig. 4. The idea is to mask the garbled input and then use the second part of GKR's idea as represented by OMSS, namely secret-share the mask, putting a piece in each token, so that unless one has all tokens, one learns nothing about the garbled input. The formal proof of the following is in Appendix D.2.

Theorem 3. For any Φ, if $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then $\mathsf{prv1\text{-}to\text{-}prv2}[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$.

The proof sketch is as follows. We first build, from a given prv2 adversary $\mathcal{A}_2$, a prv1 adversary $\mathcal{A}_1$, and then, from the simulator $\mathcal{S}_1$ for the latter, a simulator $\mathcal{S}_2$ for $\mathcal{A}_2$. The prv2 simulator $\mathcal{S}_2$ can

```
proc $\mathsf{Gb}_1(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,   $R \twoheadleftarrow \{0,1\}^k$
$F_1 \leftarrow F \oplus \textsc{Hash}(|F|, 0 \parallel R)$,   $d_1 \leftarrow d \oplus \textsc{Hash}(|d|, 1 \parallel R)$
return $(F_1, (e, R), d_1)$
```

```
proc $\mathsf{En}_1(e_1, x)$
$(e, R) \leftarrow e_1$
return $(\mathsf{En}(e, x), R)$
```

```
proc $\mathsf{Ev}_1(F_1, X_1)$
$(X, R) \leftarrow X_1$,   $F \leftarrow F_1 \oplus \textsc{Hash}(|F_1|, 0 \parallel R)$,   $Y \leftarrow \mathsf{Ev}(F, X)$
return $(Y, R)$
```

```
proc $\mathsf{De}_1(d_1, Y_1)$
$(Y, R) \leftarrow Y_1$,  $d \leftarrow d_1 \oplus \textsc{Hash}(|d_1|, 1 \parallel R)$
return $\mathsf{De}(d, Y)$
```

```
proc $\mathsf{Gb}_2(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}_1(1^k, f)$
for $i \in \{1, \ldots, n\}$ do $S_i \twoheadleftarrow \{0,1\}^k$
$(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$,   $S \leftarrow S_1 \oplus \cdots \oplus S_n$
for $i \in \{1, \ldots, n\}$ do
   $T_i^0 \leftarrow (X_i^0 \oplus \textsc{Hash}(|X_i^0|, 1 \parallel i \parallel S), S_i)$
   $T_i^1 \leftarrow (X_i^1 \oplus \textsc{Hash}(|X_i^1|, 1 \parallel i \parallel S), S_i)$
return $(F, (T_1^0, T_1^1, \ldots, T_n^0, T_n^1), d)$
```

```
proc $\mathsf{Ev}_2(F, T)$
$((U_1, S_1), \ldots, (U_n, S_n)) \leftarrow T$
$S \leftarrow S_1 \oplus \cdots \oplus S_n$
for $i \in \{1, \ldots, n\}$ do
      $X_i \leftarrow U_i \oplus \textsc{Hash}(|U_i|, 1 \parallel i \parallel S)$
return $\mathsf{Ev}_1(F, (X_1, \ldots, X_n))$
```

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

Fig. 5. Transform rom-prv-to-prv1 (top): Garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv1}, \Phi)$ obtained by applying the ROM rom-prv-to-prv1 transform to garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi)$. **Transform rom-prv1-to-prv2 (bottom):** Projective garbling scheme $\mathcal{G}_2 = (\mathsf{Gb}_2, \mathsf{En}_2, \mathsf{De}_1, \mathsf{Ev}_2, \mathsf{ev}) \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv2}, \Phi)$ obtained by applying the ROM rom-prv1-to-prv2 transform to projective garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$. The advantage of these transforms over the ones of Fig. 4 is that they preserve short garbled inputs.

return random tokens for the first $n-1$ bits of the input. Just before it must provide a token for the very last input bit, it gets the final output y. Now, it can run the prv1 simulator on y to get the real tokens and create the last piece of the secret mask and thence its last token so that the shares unmask the real tokens.

3.5 Efficient ROM transforms

The prv-to-prv1 transform does not preserve short garbled inputs, meaning even if $\mathcal{G}$ has short garbled inputs, prv-to-prv1$[\mathcal{G}]$ may not. The prv1-to-prv2 transform preserves short garbled inputs, but we usually want to apply the two transforms in sequence. We do not know how to fill this gap in the standard model under standard assumptions. We will now provide a simple way to do it in the ROM (random-oracle model).

To extend our definitions of garbling-scheme privacy to ROM [6], we follow BHR's treatment [5]. An ROM garbling scheme is a garbling scheme whose first four algorithms have access to an oracle Hash called the random oracle (RO). The model is obtained by adding the following procedure Hash to the games of Fig. 2.

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

The $\textsc{Hash}$ procedure can be called by a garbling scheme's algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, or by the adversary. If the challenge bit b is 0 then the simulator will itself answer queries made to $\textsc{Hash}$ by the adversary. In the code, `ro` is a formal symbol indicating to the simulator that it is being asked to answer a query to $\textsc{Hash}$. For xxx $\in \{\mathrm{prv}, \mathrm{prv1}, \mathrm{prv2}\}$ we let $\mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx}, \Phi)$ be the set of all garbling schemes that are xxx secure over Φ in the ROM.

The rom-prv-to-prv1 transform at the top of Fig. 5 generates the mask of the prv-to-prv1 transform by applying the RO to a random k-bit seed R, and includes R in the encoding function and garbled input and output in place of the full mask, thereby saving space. As a consequence, it preserves short garbled inputs. We claim:

Theorem 4. For any Φ, if $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi)$ then rom-prv-to-prv1$[\mathcal{G}] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv1}, \Phi)$.

The proof is in Appendix D.3. The idea is standard. The simulator can pick F_1, d_1 at random just as in the proof of Theorem 2. Then, once it has F, d, it will pick R at random and program the RO so that $F_1 = F \oplus \textsc{Hash}(|F|, 0\|R)$ and $d_1 = d \oplus \textsc{Hash}(|d|, 1\|R)$. Security relies on the fact that the probability that the adversary queries (ℓ, w) to $\textsc{Hash}$, with R being the suffix of w, prior to receiving R in the garbled input, is negligible.

As with prv-to-prv1, we note that the starting scheme is not assumed projective, but a suitable re-interpretation of the notation is enough to ensure that if the starting scheme is projective, so is the constructed one.

Our prv1-to-prv2 already preserves short garbled inputs, but the size of a token in the constructed scheme is n times the size of a token in the original scheme. The rom-prv-to-prv1 transform at the bottom of Fig. 5 does a little better, increasing the size of each token by an additive nk bits regardless of the length of the tokens of the starting scheme. The idea is again to generate the masks of the prv1-to-prv2 transform by applying the RO to a seed and then secret-sharing the latter instead of the entire mask. The proof of the following, in Appendix D.4, is again standard:

Theorem 5. For any Φ, if $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then rom-prv1-to-prv2$[\mathcal{G}_1] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$.

As the statements of Theorems 4 and 5 indicate, we are assuming in both cases that the starting scheme is a standard-model one. This is for simplicity. One can apply the transform to a ROM scheme. (And, in the case of rom-prv1-to-prv2, are likely to, since the starting scheme is likely an output of rom-prv-to-prv1.) This can be handled by suitable "domain separation" of all ROs involved.

For conceptual simplicity we have presented two separate transforms but we note that one can gain efficiency by going directly from prv to prv2. We would not pick S as in rom-prv1-to-prv2 but instead apply the secret-sharing directly to the R chosen by rom-prv-to-prv1.

3.6 "Standard" schemes are not prv2 secure

It is easy to see that prv security does not in general imply prv1 or prv2 security, meaning that there exist prv secure schemes that are not prv1 (and thus not prv2) secure (cf. Proposition 18). A more interesting question concerns the adaptive security of "standard" constructions of garbled circuits, meaning garbling schemes in the Yao style such as the Garble1 and Garble2 schemes [5] or the scheme of Lindell and Pinkas [21]. These are prv secure. But are they prv1 or prv2 secure? Here we show that they are not prv2 secure. This is for a fundamental reason, namely that they permit what we call *partial evaluation*: if certain output bits depend only on certain input bits, having the tokens for these input bits (and having the decoding rule, but not the tokens, for other input bits) allows one to compute the corresponding output bits. We will show that any scheme with this property is prv2 insecure. But the partial-evaluation property is possessed by all schemes that use the token-based, gate-encryption paradigm of Yao, in particular the ones mentioned above, and thus our results will imply that these schemes are not prv2 secure. We now proceed to formalize and prove this claim, defining what it means for a garbling scheme to permit partial evaluation and then showing that any scheme with this property fails to be prv2 secure.

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective circuit-garbling scheme, so that ev is the canonical circuit-evaluation algorithm, taking as input a circuit $f = (n, m, q, A, B, G)$ and $x \in \{0,1\}^{f.n}$ to return $\mathsf{ev}(f, x) \in \{0,1\}^{f.m}$. We extend ev to a *partial circuit evaluation algorithm* $\overline{\mathsf{ev}}$ that takes f and $x \in \{0,1,\perp\}^{f.n}$ and returns $\overline{\mathsf{ev}}(f, x) \in \{0,1,\perp\}^{f.m}$ as follows:

```
proc $\overline{\mathsf{ev}}(f, x)$
$(n, m, q, A, B, G) \leftarrow f$
for $g \leftarrow n+1$ to $n+q$ do
    $a \leftarrow A(g)$, $b \leftarrow B(g)$
    if ($x_a = \perp$ or $x_b = \perp$) then $x_g \leftarrow \perp$
    else $x_g \leftarrow G_g(x_a, x_b)$
return $x_{n+q-m+1} \cdots x_{n+q}$
```

Note that $\overline{\mathsf{ev}}(f, x) = \mathsf{ev}(f, x)$ if $x \in \{0,1\}^{f.n}$. Partial evaluation captures an inherent property of circuit evaluation, namely the ability to compute a part of the output given only the inputs on which it depends. For example if the first bit of $\mathsf{ev}(f, x)$ depends only on the first two bits of x, then this first output bit can be computed as the first bit of $\overline{\mathsf{ev}}(f, x_1 x_2 \perp \cdots \perp)$.

We say that $\mathcal{G}$ permits partial evaluation of the garbled function if the above property is inherited by the garbled-evaluation process. Thus if, as in the above example, the first bit of $\mathsf{ev}(f, x)$ depends only on the first two bits of x, then this first output bit can be computed given the garbled function F, the tokens $X_1^{x_1}, X_2^{x_2}$ and the decoding rule d, meaning tokens corresponding to the other bits of the input are not necessary. Formally we say that $\overline{\mathsf{Ev}}$ is a *partial garbled-evaluation algorithm* for $\mathcal{G}$ if for any $f \in \{0,1\}^*$, any $(F, (X_1^0, X_1^1, \ldots, X_{f.n}^0, X_{f.n}^1), d) \in [\mathsf{Gb}(1^k, f)]$, and any $x \in \{0,1,\perp\}^{f.n}$, if we let $X_i^{\perp} = \perp$ for $1 \le i \le f.n$, then

$$\mathsf{De}\big(d, \overline{\mathsf{Ev}}(F, (X_1^{x_1}, \ldots, X_n^{x_n}))\big) = \overline{\mathsf{ev}}(f, x) \ .$$

In other words, tokens may now take value $\perp$, and evaluation of the garbled circuit is still possible, the result being the corresponding partial evaluation of the circuit. We say that $\mathcal{G}$ *permits partial evaluation* if it has a PT partial garbled-evaluation algorithm. The following says this condition implies that $\mathcal{G}$ is not prv2 secure:

Proposition 6 Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a projective circuit-garbling scheme that permits partial evaluation. Then $\mathcal{G} \notin \mathsf{GS}(\mathrm{prv2}, \Phi)$ for all Φ.

The result is quite strong with regard to side-information, saying the scheme is insecure for *all* side-information functions. As we indicated above, standard garbling schemes based on the Yao paradigm of encrypted gate entries and token propagation do permit partial evaluation, so this result rules out their prv2 security.

Proof (Proposition 6). For $k \in \mathbb{N}$ let $ID_k \colon \{0,1\}^{k+1} \to \{0,1\}^{k+1}$ denote the identity function and let id_k denote a circuit such that $id_k.n = k+1$, $\mathsf{ev}(id_k, \cdot) = ID_k(\cdot)$, and $\overline{\mathsf{ev}}(id_k, x_1 \cdots x_k \bot) = x_1 \cdots x_k \bot$ for every $x_1, \ldots, x_k \in \{0,1\}$. Let $\overline{\mathsf{Ev}}$ be the partial garbled-evaluation algorithm associated to $\mathcal{G}$. Consider the following adversary:

```
adversary $\mathcal{A}_2(1^k)$
$(F, d) \leftarrow \textsc{Garble}(id_k)$
$x \twoheadleftarrow \{0,1\}^{k+1},\ \ x_1 \cdots x_{k+1} \leftarrow x$
for $i \in \{1, \ldots, k\}$ do $X_i \leftarrow \textsc{Input}(i, x_i)$
$z \leftarrow \mathsf{De}(d, \overline{\mathsf{Ev}}(F, (X_1, \ldots, X_k, \bot)))$
if $z = x_1 \cdots x_k \bot$ then return 1 else return 0
```

Let $\mathcal{S}_2$ be any (even computationally unbounded) simulator. Then for every $k \in \mathbb{N}$, letting b be the challenge bit in game $\mathrm{Prv2}_{\mathcal{G}_2, \Phi, \mathcal{S}_2}$, we have

$$\Pr\left[\, \mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k) \;\middle|\; b = 1 \,\right] = 1 \quad \text{and} \quad \Pr\left[\, \neg\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k) \;\middle|\; b = 0 \,\right] \le 2^{-k} \ .$$

The first equation uses the assumption that $\overline{\mathsf{Ev}}$ is a partial garbled-evaluation algorithm. The second equation is true because $\mathcal{S}$ has no information about the input x until the very last token is requested, and the adversary stops just short of that. Subtracting we have

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv2},\,\Phi,\,\mathcal{S}_2}(\mathcal{A}_2, k) \ge 1 - 2^{-k} \ ,$$

which proves the theorem. □

3.7 One-time programs

Security definition for a one-time compiler. The notion of a *one-time program* was put forward by Goldwasser, Kalai, and Rothblum (GKR [17]). The intent is that possession of a one-time program P for a function f should enable one to evaluate f at any single value x; but, beyond that, the one-time program should be useless. Unachievable in any standard model of computation (where possession of P would enable its repeated evaluation at multiple point), GKR suggest achieving one-time programs in a model of computation that provides *one-time memory*—tamper-resistant hardware whose read-once i-th location returns, on query $(i, b) \in \mathbb{N} \times \{0,1\}$, the string T_i^b, immediately thereafter expunging T_i^{1-b}. A *one-time compiler* probabilistically transforms the description of a function f into a one-time program P and its associated one-time memory T.

For a formal treatment, we begin by specifying two stateful oracles; see Fig. 6. The first, OTP_f, formalizes the desired behavior of a one-time program for f. Here f will now be regarded as a string, not a function, but this string represents a circuit computing a function $\mathsf{ev}(f) \colon \{0,1\}^{f.n} \to \{0,1\}^{f.m}$; we write ev for the canonical circuit-evaluation function [5]. The agent calling out to OTP_f provides x and, on the first query, it gets $\mathsf{ev}(f, x)$. Subsequent queries return nothing. On the right-hand side of Fig. 6 we similarly define an oracle OTM_T, this to model possession of a one-time-memory system. Given a list of ℓ pairs of strings (establish some convention so that every string T is regarded as denoting a list of ℓ pairs of strings, for some $\ell \in \mathbb{N}$), the oracle returns at most one string from each pair satisfying each request.

Elaborating on GKR, we now define a *one-time compiler* as a pair of probabilistic algorithms $\Pi = (\mathsf{Co}, \mathsf{Ex})$ (for *compile* and *execute*). Algorithm Co, on input 1^k and a string f, produces a pair $(P, T) \leftarrow \mathsf{Co}(1^k, f)$ where P (the one-time program) is a string and T (the one-time-memory) encodes a list of 2ℓ strings, for some ℓ. Algorithm Ex, on input of strings P and x, and given access

```
proc $\mathrm{OTP}_f(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $\mathit{called}$ then return $\bot$
$\mathit{called} \leftarrow \mathsf{true}$
return $\mathsf{ev}(f, x)$
```

```
proc $\mathrm{OTM}_T(i, b)$
$(T_1^0, T_1^1, \ldots, T_\ell^0, T_\ell^1) \leftarrow T$
if $i \notin [1..\ell]$ or $\mathit{used}_i$ or $b \notin \{0,1\}$ then return $\bot$
$\mathit{used}_i \leftarrow \mathsf{true}$
return $T_i^b$
```

Fig. 6. Oracles model one-time programs and one-time memory. Oracle OTP depends on a string f representing a boolean circuit. Oracle OTM depends on a list of strings T.

to an oracle $\mathcal{O}$, returns a string $y \leftarrow \mathsf{Ex}^{\mathcal{O}}(P, x)$. We require the following *correctness* condition of $\Pi = (\mathsf{Co}, \mathsf{Ex})$: if $(P, T) \leftarrow \mathsf{Co}(1^k, f)$ and $x \in \{0,1\}^{f.n}$ then $\mathsf{Ex}^{\mathrm{OTM}_T(\cdot,\cdot)}(P, x) = \mathsf{ev}(f, x)$.

The security of $\Pi = (\mathsf{Co}, \mathsf{Ex})$ will be relative to a side-information function Φ; the value $\phi = \Phi(f)$ captures the information about f that P is allowed to reveal.[11] So fix a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$, an adversary $\mathcal{A}$, a security parameter k, and a string f. (1) Consider the distribution $\mathsf{Real}_{\Pi,\mathcal{A},f}(k)$ determined by the following experiment: first, sample $(P, T) \leftarrow \mathsf{Co}(1^k, f)$; then, run $\mathcal{A}^{\mathrm{OTM}_T(\cdot)}(1^k, P)$ and output whatever $\mathcal{A}$ outputs. (2) Alternatively, fix a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$, a side-information function Φ, a simulator $\mathcal{S}$, a security parameter k, and a string f. Consider the distribution $\mathsf{Fake}_{\Pi,\Phi,\mathcal{S},f}(k)$ determined by the following experiment: run $S^{\mathrm{OTP}_f(\cdot)}(1^k, \Phi(f))$ and output whatever S outputs. For $\mathcal{D}$ an algorithm and Π, Φ, $\mathcal{A}$, $\mathcal{S}$, and k as above, let

$$\begin{aligned}\mathbf{Adv}^{\mathrm{otc}}_{\Pi,\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k) = {} & \Pr[(f,\sigma) \leftarrow \mathcal{D}(1^k); v \leftarrow \mathsf{Real}_{\Pi,\mathcal{A},f}(k) :\ \mathcal{D}(\sigma, v) \Rightarrow 1] - \\ & \Pr[(f,\sigma) \leftarrow \mathcal{D}(1^k); v \leftarrow \mathsf{Fake}_{\Pi,\Phi,\mathcal{S},f}(k) :\ \mathcal{D}(\sigma, v) \Rightarrow 1]\end{aligned}$$

One-time compiler Π is said to be (OTC-) *secure* with respect to side-information function Φ if for any PPT adversary $\mathcal{A}$ there is a PPT simulator $\mathcal{S}$ such that for all PPT distinguishers $\mathcal{D}$, function $\mathbf{Adv}^{\mathrm{otc}}_{\Pi,\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k)$ is negligible.

Discussion. Let us briefly talk through the definition. The distinguisher $\mathcal{D}$ selects f and is presented with a string drawn from one of two worlds. In the first world, the distinguisher is given the output (equivalently, the view) of an adversary $\mathcal{A}$ who has the garbled program P for f and its associated one-time memory. Using the execution procedure Ex the adversary could compute $\mathsf{ev}(f, x)$, if it so wishes, but it is not compelled to do so. In the second world, the distinguisher is given output produced by a simulator $\mathcal{S}$. That simulator has no one-time memory; it has only the side-information $\Phi(f)$ about f and an ideal one-time program for f. In a protocol we deem secure, no matter what the adversary does, there will be a simulator such that the two views described will be computationally close.

To arrive at an achievable notion of security, one *must* allow that information beyond the function's value at x to be leaked; minimally, information on the size of the circuit will be revealed. Indeed the construction of GKR leaks more—it divulges the *topology* of a circuit computing f. We follow BHR's approach for handling side-information, one where Φ acts as a "knob" controlling just what may be learned of f.

Constructing an OTC from a garbling scheme. A projective circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ can be turned into a one-time compiler $\Pi = (\mathsf{Co}, \mathsf{Ex})$ in a natural way: let

[11] For example, we might have $\Phi(f) = \Phi_{\mathrm{size}}(f) = (f.n, f.m, f.q)$, the number of inputs, outputs, and gates; or $\Phi(f) = \Phi_{\mathrm{topo}}(f) = f^-$, the topology of f; or $\Phi(f) = (f.n, f.m, u(f.q))$ for some monotonic u like $u(q) = 10^6\lceil 10^{-6} q \rceil$.

$\mathrm{OTC}[\mathcal{G}] = (\mathsf{Co}, \mathsf{Ex})$ be defined as follows. (1) $\mathsf{Co}(1^k, f)$: let $(F, e, d) \leftarrow \mathsf{Gb}(f)$ and return (P, T) where $P = (F, d)$ and $T = e$. (2) $\mathsf{Ex}^{\mathcal{O}}(P, x)$: Let $(F, d) \leftarrow P$, let $x_1 \cdots x_n \leftarrow x$, query oracle $\mathcal{O}$ on $(1, x_1), \ldots, (n, x_n)$ to obtain $X_1, \ldots, X_n$, respectively, and return $\mathsf{De}(d, \mathsf{Ev}(F, X))$ with $X = (X_1, \ldots, X_n)$. The proof of the following is in Appendix D.9. The straightforwardness of the construction and its trivial proof are, we believe, points in our favor, evidence of our claim that the garbling-scheme abstraction and appropriate security notions for it engender applications in direct, simple and less error-prone ways.

Theorem 7. If $\mathcal{G}$ is a prv2-secure projective garbling scheme over side-information function Φ then $\mathrm{OTC}[\mathcal{G}]$ is OTC-secure with respect to side-information Φ.

A concrete one-time compiler may be obtained from any prv-secure (projective) garbling scheme by (1) using our prv-to-prv1 transform to go from the prv garbling scheme to a prv1 one (2) using our prv1-to-prv2 transform to go from the prv1 scheme to a prv2 one, and (3) applying Theorem 7. BHR [5] provide prv-secure garbling schemes based on PRFs and thence on one-way functions, yielding a one-way function based one-time compiler to recover the original claim of GKR [17].

Analysis of $\mathrm{OTC}[\mathsf{OMSS}[\mathcal{G}]]$. The claim of GKR [17], in our language, is that if $\mathcal{G}$ is a prv-secure (projective) garbling scheme then $\mathrm{OTC}[\mathsf{OMSS}[\mathcal{G}]]$ is otc-secure. Proposition 1, showing that $\mathsf{OMSS}[\mathcal{G}]$ need not be prv2-secure, does not refute this claim, for the prv2 security of $\mathsf{OMSS}[\mathcal{G}]$, while sufficient to establish the claim, may not be necessary. Here we accordingly refute the claim by extending the counter-example of Proposition 1 to give a projective, prv-secure garbling scheme $\mathcal{G}$ for which $\mathrm{OTC}[\mathsf{OMSS}[\mathcal{G}]]$ is shown by attack to not be otc-secure. (That is, we show that this transform will yield programs that are not one-time.)

This example does not contradict the updated claim of [16], made in response to our work, of a OTC based on exponentially-hard one-way functions. The latter would correspond, in our language, to the claim that $\mathrm{OTC}[\mathsf{OMSS}[\mathcal{G}]]$ is a secure OTC if $\mathcal{G}$ has exponential prv-security.

Proceeding to the counter-example, recall that in the proof of Proposition 1, we gave a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ such that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$ but $\mathcal{G}_2 = \mathsf{OMSS}[\mathcal{G}] \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$. Now we show that $\mathrm{OTC}[\mathcal{G}_2]$ is otc-insecure, by demonstrating an attack. Distinguisher $\mathcal{D}(1^k)$ picks $R_0, R_1 \twoheadleftarrow \{0,1\}$, and lets f_{R_0,R_1} denote a circuit such that $f_{R_0,R_1}.n = k$, $f_{R_0,R_1}.m = 1$ and $\mathsf{ev}(f_{R_0,R_1}, x) = R_{x_1}$ where x_1 is the first bit of x. (We construct the circuit in such a way that the topology is independent of R_0, R_1 and depends only on k.) It queries f_{R_0,R_1}, and then outputs 1 only if the oracle's answer is $R_0 \oplus R_1$.

The adversary $\mathcal{A}(1^k)$ is given (G, ε), and parses $(G', v, V) \leftarrow G$ and $v_1 \cdots v_n \leftarrow v$. Next for every $i \leq n$, it queries (i, v_i) to Input to get back T_i and lets $(X_i, r_i) \leftarrow T_i$ and $(Z_i, V_i) \leftarrow X_i$. It lets $y \leftarrow \mathsf{De}_2(\varepsilon, \mathsf{Ev}_2(G, (T_1, \ldots, T_n)))$ and $y' \leftarrow V \oplus V_1 \oplus \cdots \oplus V_n$ and $r \leftarrow r_1 \oplus \cdots \oplus r_n$. It then returns $y \oplus y' \oplus r$. Note that $y = \mathsf{ev}(f_{R_0,R_1}, v)$ and $y' = r \oplus \mathsf{ev}(f_{R_0,R_1}, \overline{v})$ so by definition of f_{R_0,R_1} we have $y \oplus y' \oplus r = R_0 \oplus R_1$. Hence given $\mathsf{Real}_{\Pi,\mathcal{A},f}(k)$, the distinguisher always outputs 1.

Let $\mathcal{S}$ be any (even computationally unbounded) simulator. It is given only $1^k, \Phi_{\mathrm{topo}}(f)$ as input and can obtain no information on R_0 or R_1 beyond their length because the topology of f_{R_0,R_1} is by construction independent of R_0, R_1. The simulator is given oracle access to $\mathrm{OTP}_{f_{R_0,R_1}}$ and can obtain either R_0 or R_1 but not both. Since R_0 is independent of $R_0 \oplus R_1$, and so is R_1, the probability that the simulator can output $R_0 \oplus R_1$ is 1/2. Hence, given $\mathsf{Fake}_{\Pi,\Phi,\mathcal{S},f}(k)$, the distinguisher outputs 1 with probability 1/2.

```
proc Garble($f, x$)                                    $\mathrm{Obv}_{\mathcal{G},\Phi,\mathcal{S}}$
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($f$)                                       $\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}$
if $b = 1$ then
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
else
    $F \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$
return $F$

proc Input($x$)
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $b = 1$ then $X \leftarrow \mathsf{En}(e, x)$
else $X \leftarrow \mathcal{S}(1)$
return $X$
```

```
proc Garble($f$)                                       $\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}$
$n \leftarrow f.n$;  $Q \leftarrow \emptyset$;  $\sigma \leftarrow \varepsilon$
if $b = 1$ then
    $(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$
else
    $F \leftarrow \mathcal{S}(1^k, \Phi(f), 0)$
return $F$

proc Input($i, c$)
if $i \notin \{1, \ldots, n\} \setminus Q$ then return $\bot$
$x_i \leftarrow c$;  $Q \leftarrow Q \cup \{i\}$
if $b = 1$ then $X_i \leftarrow X_i^{x_i}$
else $X_i \leftarrow \mathcal{S}(i, |Q|)$
return $X_i$
```

```
proc Garble($f, x$)                                    $\mathrm{Aut}_{\mathcal{G}}$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
$X \leftarrow \mathsf{En}(e, x)$
return $(F, X)$
```

```
proc Garble($f$)                                       $\mathrm{Aut1}_{\mathcal{G}}$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
return $F$

proc Input($x$)
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$X \leftarrow \mathsf{En}(e, x)$
return $X$
```

```
proc Garble($f$)                                       $\mathrm{Aut2}_{\mathcal{G}}$
$n \leftarrow f.n$;  $Q \leftarrow \emptyset$;  $\sigma \leftarrow \varepsilon$
$(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$
return $F$

proc Input($i, c$)
if $i \notin \{1, \ldots, n\} \setminus Q$ then return $\bot$
$x_i \leftarrow c$;  $Q \leftarrow Q \cup \{i\}$,  $X_i \leftarrow X_i^{x_i}$
if $|Q| = n$ then $X \leftarrow (X_1, \ldots, X_n)$
return $X_i$
```

Fig. 7. Obliviousness (top). Games for defining the obv, obv1, and obv2 security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. For each game, Initialize() samples $b \twoheadleftarrow \{0,1\}$ and Finalize(b') returns $(b = b')$. **Authenticity (bottom).** Games for defining the aut, aut1, and aut2 security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. Procedure Finalize($Y$) of each game returns 0 if $x \notin \{0,1\}^{f.n}$, otherwise it returns $(\mathsf{De}(d, Y) \neq \bot$ **and** $Y \neq \mathsf{Ev}(F, X))$.

4 Obliviousness, Authenticity and Secure Outsourcing

We define obliviousness and authenticity, both with either the coarse-grained or fine-grained adaptivity. We show how to achieve these goals, in combination with adaptive privacy, via generic transforms and in the standard model. We then give more efficient transforms for the ROM model. Finally we apply this to obtain extremely simple and modular designs, and security proofs, for verifiable outsourcing schemes based on the paradigm of GGP [12].

4.1 Definitions for adaptive obliviousness and authenticity

Obliviousness. Intuitively, a garbling scheme is *oblivious* if garbled function F and garbled input X, these corresponding to f and x, reveal nothing of f or x beyond side-information $\Phi(f)$. In particular, possession of F and X will not allow the calculation of $y = \mathsf{ev}(f, x)$.

The formal definition for *static* obliviousness is from BHR [5]. See the top-left panel of Fig. 7. We add to this two new definitions, to incorporate either coarse-grained or fine-grained adaptive security. See the top-middle and top-right panels of Fig. 7. Fine-grained adaptive security continues to require that $\mathcal{G}$ be projective. The games used for defining obliviousness closely mirror their privacy counterparts. The first important difference is that the adversary does not get the decoding function d. The second important difference is that the simulator must do without $y = \mathsf{ev}(f, x)$. For a garbling scheme $\mathcal{G}$, side-information Φ, simulator $\mathcal{S}$, adversary $\mathcal{A}$, and security parameter $k \in \mathbb{N}$, we let $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\,\Phi,\,\mathcal{S}}(\mathcal{A}, k) = 2\Pr[\mathrm{Obv}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k)] - 1$, $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv1},\,\Phi,\,\mathcal{S}}(\mathcal{A}, k) = 2\Pr[\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k)] - 1$, and finally $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv2},\,\Phi,\,\mathcal{S}}(\mathcal{A}, k) = 2\Pr[\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k)] - 1$. Garbling scheme $\mathcal{G}$ is obv secure with respect to Φ if for every PPT $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\,\Phi,\,\mathcal{S}}(\mathcal{A}, k)$ is negligible. We similarly define obv1 and obv2 security. For $\mathrm{xxx} \in \{\mathrm{obv}, \mathrm{obv1}, \mathrm{obv2}\}$ we let $\mathsf{GS}(\mathrm{xxx}, \Phi)$ denote the set of all garbling schemes that are xxx secure over Φ.

Authenticity. Fig. 7 also formalizes the games underlying three definitions of authenticity, capturing an adversary's inability to create from F and X a garbled output $Y \neq F(X)$ that will be deemed authentic. The static definition of BHR [5] is strengthened either to allow the adversary to specify x subsequent to obtaining F, or, stronger, the bits of x are provided one-by-one, each corresponding token then issued. For the second case, game Aut2, the garbling scheme must once again be projective. For a garbling scheme $\mathcal{G}$, adversary $\mathcal{A}$, and security parameter $k \in \mathbb{N}$, we let $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A}, k) = \Pr[\mathrm{Aut}_{\mathcal{G}}^{\mathcal{A}}(k)]$, $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut1}}(\mathcal{A}, k) = \Pr[\mathrm{Aut1}_{\mathcal{G}}^{\mathcal{A}}(k)]$, and $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut2}}(\mathcal{A}, k) = \Pr[\mathrm{Aut2}_{\mathcal{G}}^{\mathcal{A}}(k)]$. Garbling scheme $\mathcal{G}$ is aut secure if for every PPT $\mathcal{A}$ $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A}, k)$ is negligible. We similarly define aut1 and aut2 security. For $\mathrm{xxx} \in \{\mathrm{aut}, \mathrm{aut1}, \mathrm{aut2}\}$ we let $\mathsf{GS}(\mathrm{xxx})$ denote the set of all garbling schemes that are xxx secure.

4.2 Achieving obv1 and aut1 security

It is tempting to think that the $\mathsf{prv\text{-}to\text{-}prv1}$ operator in Fig. 4 will also promote xxx security, with $\mathrm{xxx} \in \{\mathrm{obv}, \mathrm{aut}\}$, to xxx1 security. However, a second glance reveals that $\mathsf{prv\text{-}to\text{-}prv1}$ does not promote aut to aut1, as the following counter-example illustrates. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme that is aut secure. Consider $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}', \mathsf{Ev}, \mathsf{ev})$ defined as follows. On input $(1^k, f)$, the algorithm Gb' creates $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and then returns $(F, e, 1 \parallel d)$. On input (d', Y), the algorithm De' parses $d' = b \parallel d$, and outputs $\mathsf{De}(d, Y)$ if $b = 1$, and outputs 1 otherwise. The scheme $\mathcal{G}'$ inherits aut security from $\mathcal{G}$. The scheme $\mathcal{G}_1 = \mathsf{prv\text{-}to\text{-}prv1}[\mathcal{G}'] = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev})$ is not even aut secure. An adversary can attack $\mathcal{G}_1$ as follows. First, query an arbitrary circuit f and input $x \in \{0,1\}^{f.n}$ to receive (F_1, X_1). Let $X_1 = (X, d', F')$. Then, output $Y = (1, \overline{d'})$. Let d_1 be the decoding function used to authenticate Y. Then $d_1 \oplus d' = 1 \parallel d$, and $d_1 \oplus \overline{d'} = 0 \parallel \overline{d}$. Hence $\mathsf{De}_1(d_1, Y) = 1$, and the adversary wins with advantage 1.

We now show how to change $\mathsf{prv\text{-}to\text{-}prv1}$ to an operator $\mathsf{all\text{-}to\text{-}all1}$ that promotes any $\mathrm{xxx} \in \{\mathrm{prv}, \mathrm{obv}, \mathrm{aut}\}$ to being xxx1 secure. The insecurity of the $\mathsf{prv\text{-}to\text{-}prv1}$ operator arises because the adversary can forge a *fake* d', where d' is the one-time pad masking the decoding function d. To prevent this, we choose $K \twoheadleftarrow \{0,1\}^k$, and append $\mathsf{F}_K(d')$ to the garbled input X, where $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ is a PRF. The decoding function will be $(d' \oplus d, K)$. See Fig. 8. The proof of the following is in Appendix D.5.

Theorem 8. (1) For any Φ and any $\mathrm{xxx} \in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G} \in \mathsf{GS}(\mathrm{xxx}, \Phi)$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\mathrm{xxx1}, \Phi)$ (2) If $\mathcal{G} \in \mathsf{GS}(\mathrm{aut})$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\mathrm{aut1})$ (3) If $\mathcal{G} \in \mathsf{GS}(\mathrm{proj})$ then $\mathsf{all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}(\mathrm{proj})$.

```
proc $\mathsf{Gb}_1(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,  $F' \twoheadleftarrow \{0,1\}^{|F|}$,  $d' \twoheadleftarrow \{0,1\}^{|d|}$
$F_1 \leftarrow F \oplus F'$,  $K \twoheadleftarrow \{0,1\}^k$,  $d_1 \leftarrow (d \oplus d', K)$
$\mathrm{tag} \leftarrow \mathsf{F}_K(d')$,  $e_1 \leftarrow (e, d', F', \mathrm{tag})$
return $(F_1, e_1, d_1)$

proc $\mathsf{En}_1(e_1, x)$
$(e, d', F', \mathrm{tag}) \leftarrow e_1$
return $(\mathsf{En}(e, x), d', F', \mathrm{tag})$

proc $\mathsf{Ev}_1(F_1, X_1)$
$(X, d', F', \mathrm{tag}) \leftarrow X_1$,  $F \leftarrow F_1 \oplus F'$,  $Y \leftarrow \mathsf{Ev}(F, X)$
return $(Y, d', \mathrm{tag})$

proc $\mathsf{De}_1(d_1, Y_1)$
$(Y, d', \mathrm{tag}) \leftarrow Y_1$,  $(D, K) \leftarrow d_1$,  $d \leftarrow D \oplus d'$
if $\mathrm{tag} \neq \mathsf{F}_K(d')$ then return $\bot$
return $\mathsf{De}(d, Y)$
```

```
proc $\mathsf{Gb}_1(1^k, f)$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,  $R \twoheadleftarrow \{0,1\}^k$, $K \twoheadleftarrow \{0,1\}^k$
$F_1 \leftarrow F \oplus \textsc{Hash}(|F|, 0 \parallel R)$,  $D \leftarrow d \oplus \textsc{Hash}(|d|, 1 \parallel R)$
$\mathrm{tag} \leftarrow \textsc{Hash}(k, K \parallel R)$,  $d_1 \leftarrow (D, K)$
return $(F_1, (e, R, \mathrm{tag}), d_1)$

proc $\mathsf{En}_1(e_1, x)$
$(e, R, \mathrm{tag}) \leftarrow e_1$
return $(\mathsf{En}(e, x), R, \mathrm{tag})$

proc $\mathsf{Ev}_1(F_1, X_1)$
$(X, R, \mathrm{tag}) \leftarrow X_1$,  $F \leftarrow F_1 \oplus \textsc{Hash}(|F_1|, 0 \parallel R)$
$Y \leftarrow \mathsf{Ev}(F, X)$
return $(Y, R, \mathrm{tag})$

proc $\mathsf{De}_1(d_1, Y_1)$
$(Y, R, \mathrm{tag}) \leftarrow Y_1$,  $(D, K) \leftarrow d_1$
$d \leftarrow D \oplus \textsc{Hash}(|D|, 1 \parallel R)$
if $\textsc{Hash}(|K|, K \parallel R) \neq \mathrm{tag}$ then return $\bot$
return $\mathsf{De}(d, Y)$
```

Fig. 8. Transform all-to-all1 (top): Scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{obv1}, \Phi) \cap \mathsf{GS}(\mathrm{aut1})$ obtained from scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi) \cap \mathsf{GS}(\mathrm{obv}, \Phi) \cap \mathsf{GS}(\mathrm{aut})$. The transform uses a PRF $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$. **Transform rom-all-to-all1 (bottom):** Garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}_1, \mathsf{En}_1, \mathsf{De}_1, \mathsf{Ev}_1, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi) \cap \mathsf{GS}(\mathrm{obv1}, \Phi) \cap \mathsf{GS}(\mathrm{aut1})$ obtained by applying the ROM rom-all-to-all1 transform to garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv}, \Phi) \cap \mathsf{GS}(\mathrm{obv}, \Phi) \cap \mathsf{GS}(\mathrm{aut})$. It makes use of an RO-modeled $\textsc{Hash}$. The advantage of the bottom transform over the top one is that it preserves short garbled inputs.

4.3 Achieving obv2 and aut2 security

The transform to promote coarse-grained to fine-grained security is unchanged; we let all1-to-all2 = prv1-to-prv2 be the transform at the bottom of Fig. 4. We claim it has additional features captured by the following, whose proof is in Appendix D.6.

Theorem 9. (1) For any Φ and any xxx $\in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{xxx1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then all1-to-all2$[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{xxx2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ (2) If $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{aut1}) \cap \mathsf{GS}(\mathrm{proj})$ then all1-to-all2$[\mathcal{G}_1] \in \mathsf{GS}(\mathrm{aut2}) \cap \mathsf{GS}(\mathrm{proj})$.

4.4 Efficient ROM transforms

Again, the all-to-all1 transform does not preserve short garbled inputs. We give the transform rom-all-to-all1 in the ROM to fill the gap. The same attack to break the aut1 security of all-to-all1 can be used to show that rom-prv-to-prv1 is inadequate to handle authenticity as well. The rom-all-to-all1 transform at the bottom of Fig. 8 generates the mask of the all-to-all1 transform by applying the RO to a random k-bit seed R, and includes R in the encoding rule and garbled input and output in place of the full mask, thereby saving space. As a consequence, it preserves short garbled inputs. Instead of using a PRF $\mathsf{F}_K : \{0,1\}^* \to \{0,1\}^k$, we call $\textsc{Hash}(k, K \parallel \cdot)$. For each xxx $\in \{\mathrm{obv}, \mathrm{obv1}, \mathrm{obv2}\}$, let $\mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx}, \Phi)$ be the set of all garbling schemes that are xxx secure over Φ in the ROM. Likewise, for each xxx $\in$ {aut, aut1, aut2}, let $\mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx})$ be the set of all garbling schemes that are xxx secure in the ROM. We claim:

Theorem 10. (1) For any Φ and any xxx $\in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G} \in \mathsf{GS}(\mathrm{xxx}, \Phi)$ then $\mathsf{rom\text{-}all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx1}, \Phi)$. (2) If $\mathcal{G} \in \mathsf{GS}(\mathrm{aut})$ then $\mathsf{rom\text{-}all\text{-}to\text{-}all1}[\mathcal{G}] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{aut})$.

The proof is in Appendix D.7. We note that the starting scheme is not assumed projective, but a suitable re-interpretation of the notation is enough to ensure that if the starting scheme is projective, so is the constructed one.

The ROM transform to promote coarse-grained to to fine-grained security is unchanged; we let $\mathsf{rom\text{-}all1\text{-}to\text{-}all2} = \mathsf{rom\text{-}prv1\text{-}to\text{-}prv2}$ be the transform at the bottom of Fig. 5. We claim the following theorem; the proof is in Appendix D.8

Theorem 11. (1) For any Φ and any xxx $\in \{\mathrm{prv}, \mathrm{obv}\}$: If $\mathcal{G}_1 \in \mathsf{GS}(\mathrm{xxx1}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$ then scheme $\mathsf{rom\text{-}all1\text{-}to\text{-}all2}[\mathcal{G}_1] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{xxx2}, \Phi) \cap \mathsf{GS}(\mathrm{proj})$, and (2) If $\mathcal{G}_1 \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{aut1}) \cap \mathsf{GS}(\mathrm{proj})$ then scheme $\mathsf{rom\text{-}all1\text{-}to\text{-}all2}[\mathcal{G}_1] \in \mathsf{GS}^{\mathrm{rom}}(\mathrm{aut2}) \cap \mathsf{GS}(\mathrm{proj})$.

4.5 Application to secure outsourcing

DEFINITIONS. An outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$ is a tuple of PT algorithms that, intuitively, will be run partly on a *client* and partly on a *server*. Generation algorithm Gen is run by the client on input of the unary encoding 1^k and a string f describing the function $\mathsf{ev}(f, \cdot)\colon \{0,1\}^{f.n} \to \{0,1\}^{f.m}$ to be evaluated (so that ev, like in a garbling scheme, is a deterministic evaluation algorithm) to get back a public key pk that is sent to the server and a secret key sk that is kept by the client. Algorithm Inp is run by the client on input pk, sk and $x \in \{0,1\}^{f.n}$ to return a garbled input X that is sent to the server. Associated state information St is preserved by the client. Algorithm Comp is run by the server on input pk, X to get a garbled output Y that is returned to the client. The latter runs deterministic algorithm Out on pk, sk, Y, St to get back $y \in \{0,1\}^{f.n} \cup \{\bot\}$. Correctness requires that for all $k \in \mathbb{N}$, all $f \in \{0,1\}^*$, and all $x \in \{0,1\}^{f.n}$, if $(pk, sk) \leftarrow \mathsf{Gen}(1^k, f)$, $(X, St) \leftarrow \mathsf{Inp}(pk, sk, x)$, $Y \leftarrow \mathsf{Comp}(pk, X)$, and $y \leftarrow \mathsf{Out}(pk, sk, Y, St)$, then $y = \mathsf{ev}(f, x)$. Our syntax is the same as that of GGP [12] except for distinguishing between functions and their descriptions, as represented the addition of ev to the list.

The games OSVF_{Π} and $\mathrm{OSPR}_{\Pi,\Phi,\mathcal{S}_{\mathrm{os}}}$ of Fig. 9 are used to define *verifiability* and *privacy* of an outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$, where Φ is a side-information function and $\mathcal{S}_{\mathrm{os}}$ is a simulator. In both games, the adversary is allowed only one GETPK query, and this must be its first oracle query. For adversaries $\mathcal{A}_{\mathrm{os}}$ and $\mathcal{B}_{\mathrm{os}}$, we let

$$\mathbf{Adv}_{\Pi}^{\mathrm{osvf}}(\mathcal{A}_{\mathrm{os}}, k) = \Pr[\mathrm{OSVF}_{\Pi}^{\mathcal{A}_{\mathrm{os}}}(k)] \quad \text{and} \quad \mathbf{Adv}_{\Pi}^{\mathrm{ospr}, \Phi, \mathcal{S}_{\mathrm{os}}}(\mathcal{B}_{\mathrm{os}}, k) = 2\Pr[\mathrm{OSPR}_{\Pi,\Phi,\mathcal{S}_{\mathrm{os}}}^{\mathcal{B}_{\mathrm{os}}}(k)] - 1 \ .$$

We say that Π is verifiable if $\mathbf{Adv}_{\Pi}^{\mathrm{osvf}}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible for all PT adversaries $\mathcal{A}_{\mathrm{os}}$. We say that Π is private over Φ if for all PT adversaries $\mathcal{B}_{\mathrm{os}}$ there is a PT simulator $\mathcal{S}_{\mathrm{os}}$ (that maintains state across invocations) such that $\mathbf{Adv}_{\Pi}^{\mathrm{ospr}, \Phi, \mathcal{S}_{\mathrm{os}}}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible. An adversary is said to be one-time if it makes only one INPUT query. We say that Π is one-time verifiable if $\mathbf{Adv}_{\Pi}^{\mathrm{osvf}}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible for all PT one-time adversaries $\mathcal{A}_{\mathrm{os}}$. We say that Π is one-time private over Φ if for all PT one-time adversaries $\mathcal{B}_{\mathrm{os}}$ there is a PT simulator $\mathcal{S}_{\mathrm{os}}$ such that $\mathbf{Adv}_{\Pi}^{\mathrm{ospr}, \Phi, \mathcal{S}_{\mathrm{os}}}(\mathcal{A}_{\mathrm{os}}, \cdot)$ is negligible.

Our verifiability definition coincides with that of GGP [12] but our privacy definition is stronger: it requires not just "input privacy" (concealing each input x) but, also, privacy of the function f (relative to Φ). (As in our garbling definitions this is subject to $\Phi(f)$ being revealed). Also, while

```
proc $\textsc{GetPK}(f)$                                   $\mathrm{OSVF}_{\Pi}$
$(pk, sk) \leftarrow \mathsf{Gen}(1^k, f)$,  $i \leftarrow 0$
return $pk$

proc $\textsc{Input}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$i \leftarrow i+1$,  $x_i \leftarrow x$
$(X_i, St_i) \leftarrow \mathsf{Inp}(pk, sk, x)$
return $X_i$

proc $\textsc{Finalize}(Y, j)$
if $j \notin \{1, \ldots, i\}$ then return false
$y \leftarrow \mathsf{Out}(pk, sk, Y, St_j)$
return $(y \notin \{\mathsf{ev}(f, x_j), \bot\})$
```

```
proc $\textsc{GetPK}(f)$                                   $\mathrm{OSPR}_{\Pi,\Phi,\mathcal{S}_{\mathrm{os}}}$
$c \twoheadleftarrow \{0,1\}$
if $c = 1$ then $(pk, sk) \leftarrow \mathsf{Gen}(1^k, f)$
else $pk \leftarrow \mathcal{S}_{\mathrm{os}}(1^k, \Phi(f), 0)$
return $pk$

proc $\textsc{Input}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
if $c = 1$ then $(X, St) \leftarrow \mathsf{Inp}(pk, sk, x)$
else $X \leftarrow \mathcal{S}_{\mathrm{os}}(1)$
return $X$

proc $\textsc{Finalize}(c')$
return $(c = c')$
```

```
$\mathsf{Gen}(1^k, f)$
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
    return $(F, (e, d))$

$\mathsf{Inp}(F, (e, d), x)$
    $X \leftarrow \mathsf{En}(e, x)$
    return $(X, \varepsilon)$

$\mathsf{Comp}(F, X)$
    $Y \leftarrow \mathsf{Ev}(F, X)$
    return $Y$

$\mathsf{Out}(F, (e, d), Y, St)$
    $y \leftarrow \mathsf{De}(d, Y)$
    return $y$
```

Fig. 9. Games to define the **verifiability** (OSVF) (top left) and **privacy** (OSPR) (top right) of outsourcing scheme $\Pi = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$, and the outsourcing scheme $\Pi[\mathcal{G}] = (\mathsf{Gen}, \mathsf{Inp}, \mathsf{Out}, \mathsf{Comp}, \mathsf{ev})$ (bottom) constructed from garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$.

GGP use an indistinguishability-style formalization, we use a simulation-style one, as this is stronger for some side-information functions.

To be "interesting" the work of the client in an outsourcing scheme should be less than the work required to compute the function directly, for otherwise outsourcing is not buying anything. An outsourcing scheme is said to be non-trivial if this condition is met.

Achieving one-time security. GGP show how to use FHE to turn any one-time verifiable and private outsourcing scheme into a fully verifiable and private one. This allows us to focus on designing the former. We show how a garbling scheme that is both aut1 and obv1 secure immediately implies a one-time verifiable and private outsourcing scheme. The construction, given in Fig. 9, is very direct, and the proof of the following, given in Appendix D.10, is trivial. These points reinforce our claim that the garbling scheme abstraction and adaptive security may be easily used in applications.

Theorem 12. If $\mathcal{G} \in \mathsf{GS}(\mathrm{obv1}, \Phi) \cap \mathsf{GS}(\mathrm{aut1})$ then outsourcing scheme $\Pi[\mathcal{G}]$ is one-time verifiable and also one-time private over Φ.

A benefit of our modular approach is that we may use any obv1+aut1 garbling scheme as a starting point while GGP were tied to the scheme of [21]. However, the latter scheme is not adaptively secure, which brings us to our next point.

Discussion. GGP give a proof that their outsourcing scheme is one-time verifiable assuming the encryption scheme underlying the garbled-circuit construction of Lindell and Pinkas (LP) [21] has semantic security, and elusive and verifiable range. However, their proof has a gap. Quoting [12, p. 12 of Aug 2010 ePrint version]: "For any two values x, x' with $f(x) = f(x')$, the security of Yao's protocol implies that no efficient player P_2 can distinguish if x or x' was used." This claim is correct if both x and x' are chosen independently of the randomness in the garbled circuit. But in their setting, the string x is chosen *after* the adversary sees the garbled circuit, and the security proof given by LP no longer applies.

GGP's proof effectively only shows that the garbled circuit construction of LP is (in our language, if cast as a garbling scheme) aut secure. But we show in Proposition 22 that aut security does not always imply aut1 security. One may try to give a new proof that the LP garbling scheme satisfies aut1 security. However, this seems to be difficult. Intuitively, an adaptive attack on the garbling scheme allows the adversary to mount a key-revealing selective-opening (SOA-K) attack on the underlying encryption scheme. But SOA-K secure encryption is notoriously hard to achieve [3] and not achieved by standard encryption schemes. The only known way to achieve it is via non-committing encryption [8, 9, 11], which is only possible with keys as long as the total number of bits of message ever encrypted [23], making the outsourcing scheme fail to be non-trivial.

This brings us to another discussion of non-triviality. The obv1 + aut1 secure scheme obtained via our all-to-all1 transform has long garbled inputs, so the one-time verifiable outsourcing scheme yielded by Theorem 12, while secure, is not non-trivial. Our ROM transforms yield an ROM obv1 + aut1 secure scheme with short garbled inputs and thence a non-trivial one-time outsourcing scheme but the FHE-based method of GGP of lifting it to a many-time scheme does not work in the ROM. Finding a obv1 + aut1 secure garbling scheme with short garbled inputs in the standard model under standard assumptions is an open problem. This means that right now we know of no correct way to instantiate GPP's construction to get a non-trivial and proven secure outsourcing scheme in the standard model, based on standard assumptions. We think Theorem 12 is still useful because it can be used at any point such a scheme emerges. All this again is an indication of the subtleties and hidden challenges underlying adaptive security of garbled circuits that seem to have been overlooked in the literature.

Acknowledgments

Thanks to the ASIACRYPT reviewers for their helpful comments, and thanks to the NSF for their continuing support: authors were supported in part by NSF grants CNS 0904380, CCF 0915675, CNS 1116800, CNS 1228890 and CNS 1228828.

References

1. 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.
2. B. Applebaum, Y. Ishai, E. Kushilevitz, and B. Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *Proceedings of CRYPTO 2013*. Springer, 2013. Full version as ePrint Archive, Report 2012/693.
3. M. Bellare, R. Dowsley, B. Waters, and S. Yilek. Standard security does not imply security against selective-opening. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 645–662. Springer, Apr. 2012.
4. M. Bellare, V. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In K. Sako and X. Wang, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Dec. 2012. Full version as ePrint Archive, Report 2012/564, October, 2012.
5. M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Computer and Communications Security (CCS'12)*. ACM, 2012. Full version as ePrint Archive, Report 2012/265, May, 2012.
6. 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.

7. 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.
8. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.
9. S. Choi, D. Dachman-Soled, T. Malkin, and H. Wee. Improved non-committing encryption with applications to adaptively secure protocols. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 287–302. Springer, Dec. 2009.
10. K. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Aug. 2010.
11. I. Damgård and J. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Aug. 2000.
12. 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.
13. O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
14. O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989.
15. 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, *STOC*, pages 218–229. ACM, 1987.
16. S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. Manuscript, full version of [17], July 2012.
17. S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Aug. 2008.
18. V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Feb. 2010.
19. 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.
20. S. Kamara and L. Wei. Special-purpose garbled circuits. Manuscript, 2012.
21. 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.
22. 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.
23. J. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Aug. 2002.
24. A. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.
25. A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

A Preliminaries

We review basic definitions and notation.

A.1 Notation and conventions

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

A.2 Code-based games

Our definitions and proofs are expressed via code-based games [7] so we recall here the language and specify the particular conventions we use. A code-based game—see Fig. 2 for an example—consists of an Initialize procedure, procedures that respond to adversary oracle queries, and a Finalize procedure. All procedures are optional. In an execution of game Gm with an adversary $\mathcal{A}$, the latter is given input 1^k where k is the security parameter, and the security parameter k used in the game is presumed to be the same. Procedure Initialize, if present, executes first, and its output is input to the adversary, who may now invoke other procedures. Each time it makes a query, the corresponding game procedure executes, and what it returns, if anything, is the response to $\mathcal{A}$'s query. The adversary's output is the input to Finalize, and the output of the latter, denoted $\mathrm{Gm}^{\mathcal{A}}(k)$, is called the output of the game. Finalize may be absent in which case it is understood to be the identity function, so that the output of the game is the output of the adversary. We let "$\mathrm{Gm}^{\mathcal{A}}(k) \Rightarrow c$" denote the event that this game output takes value c and let "$\mathrm{Gm}^{\mathcal{A}}(k)$" be shorthand for "$\mathrm{Gm}^{\mathcal{A}}(k) \Rightarrow \mathsf{true}$." Boolean flags are assumed initialized to false, sets to $\emptyset$ and integers to 0. $\mathsf{BAD}(\mathrm{Gm}^{A}(k))$ is the event that the execution of game Gm with adversary A sets flag *bad* to true.

A.3 Circuits

Circuits. For completeness, it is necessary to formalize boolean circuits. We do so, directly quoting BHR [5, p. 5–7].

A *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 $\mathrm{Inputs} = \{1, \ldots, n\}$, $\mathrm{Wires} = \{1, \ldots, n+q\}$, $\mathrm{OutputWires} = \{n+q-m+1, \ldots, n+q\}$, and $\mathrm{Gates} = \{n+1, \ldots, n+q\}$. Then $A\colon \mathrm{Gates} \to \mathrm{Wires}\backslash\mathrm{OutputWires}$ is a function to identify each gate's *first* incoming wire and $B\colon \mathrm{Gates} \to \mathrm{Wires}\backslash\mathrm{OutputWires}$ is a function to identify each gate's *second* incoming wire. Finally $G\colon \mathrm{Gates} \times \{0,1\}^2 \to \{0,1\}$ is a function that determines the *functionality* of each gate. We require $A(g) < B(g) < g$ for all $g \in \mathrm{Gates}$.

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 comprises a topological sort.

It is common to 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 $r = n + q$.

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

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

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. 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.

B Indistinguishability-Based Definitions

We define the indistinguishability-based counterparts of our prv1, prv2, obv1, and obv2 definitions in Fig. 10; the prv2.ind and obv2.ind again require garbling schemes to be projective. The top boxes of Fig. 10 are BHR's ind-based notions prv.ind and obv.ind. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme and let Φ be a side-information function. The prv.ind advantage of an adversary $\mathcal{A}$ is defined by $\mathbf{Adv}_{\mathcal{G}}^{\text{prv.ind},\Phi}(\mathcal{A}, k) = 2\Pr[\text{PrvInd}_{\mathcal{G},\Phi}^{\mathcal{A}}(k)] - 1$. Define $\mathbf{Adv}_{\mathcal{G}}^{\text{xxx},\Phi}(\mathcal{A}, k)$ similarly, for any xxx $\in \{\text{prv.ind}, \text{prv1.ind}, \text{prv2.ind}, \text{obv.ind}, \text{obv1.ind}, \text{obv2.ind}\}$. We say that $\mathcal{G}$ is xxx secure over Φ if $\mathbf{Adv}_{\mathcal{G}}^{\text{xxx},\Phi}(\mathcal{A}, k)$ is negligible, for every PT adversary $\mathcal{A}$. Let $\mathsf{GS}(\text{xxx}, \Phi)$ be the set of all garbling schemes that are xxx secure over Φ. Below, we will explore the relations between ind-based and sim-based notions, as illustrated in Fig. 11. It is obvious that prv2.ind ⇒ prv1.ind ⇒ prv.ind and obv2.ind ⇒ obv1.ind ⇒ obv.ind.

Discussion. While most of our ind-based notions directly follow BHR, defining prv2.ind requires care, and merits some discussion. Consider a natural variant prv2.ind.bad in which procedure Finalize(b') of game $\text{Prv2Ind}_{\mathcal{G},\Phi}$ returns

$$(b = b') \wedge (\Phi(f_0) = \Phi(f_1)) \wedge (|Q| = n) \wedge ((\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)),$$

requiring the adversary to fully specify its input strings x_0 and x_1 and get no credit if it only gives, say, the first bits of x_0 and x_1, and makes its guess. Doing so would severely limit the adversary's choice of querying (i, c_0, c_1) to the Input oracle, because it needs to make sure that the bits c_0 and c_1 can end up making strings x_0 and x_1 satisfying $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. In contrast, for prv2.ind security, if the adversary does not fully specify x_0 and x_1 then the bits c_0 and c_1 can be arbitrary, and the adversary will not be "giving up" on the game.

We now show that in fact, prv2.ind.bad is "wrong", insofar as it doesn't imply prv1.ind. Fix a length-preserving permutation $P : \{0,1\}^* \to \{0,1\}^*$ that is *one-way*: for every PT adversary $\mathcal{A}$, the advantage

$$\mathbf{Adv}_P^{\text{ow}}(\mathcal{A}, k) = \Pr[x \leftarrow \{0,1\}^k;\ x' \leftarrow \mathcal{A}(P(x)) \colon\ x' = x]$$

is negligible. For every $f, x \in \{0,1\}^*$, let $\Phi(f) = (f.n, f.m, |f|)$, and let $\mathsf{ev}^P(f, x) = P(b \parallel x)$, where b is the last bit of f. Consider the following projective garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}^P)$. Let

```
proc GARBLE($f_0, f_1, x_0, x_1$)                                    $\mathrm{PrvInd}_{\mathcal{G},\Phi}$
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($f_0, f_1, x_0, x_1$)                                    $\mathrm{ObvInd}_{\mathcal{G},\Phi}$
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($f_0, f_1$)                                              $\mathrm{Prv1Ind}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$
return $(F, d)$

proc INPUT($x_0, x_1$)
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 $\bot$
return $\mathsf{En}(e, x_b)$
```

```
proc GARBLE($f_0, f_1$)                                              $\mathrm{Obv1Ind}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$
return $F$

proc INPUT($x_0, x_1$)
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 $\bot$
return $\mathsf{En}(e, x_b)$
```

```
proc GARBLE($f_0, f_1$)                                              $\mathrm{Prv2Ind}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
$n \leftarrow f_0.n$,  $Q \leftarrow \emptyset$
$(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f_b)$
return $(F, d)$

proc INPUT($i, c_0, c_1$)
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
if $i \notin \{1, \ldots, n\} \setminus Q$ then return $\bot$
$x_{0,i} \leftarrow c_0$,  $x_{1,i} \leftarrow c_1$,  $Q \leftarrow Q \cup \{i\}$
if $|Q| = n$ then $x_0 \leftarrow x_{0,1} \cdots x_{0,n}$,  $x_1 \leftarrow x_{1,1} \cdots x_{1,n}$
return $X_i^{x_{b,i}}$

proc FINALIZE($b'$)
if $\Phi(f_0) = \Phi(f_1)$ and $|Q| = n$ then
   return $\big((b = b') \wedge (\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1))\big)$
else return $(b = b')$
```

```
proc GARBLE($f_0, f_1$)                                              $\mathrm{Obv2Ind}_{\mathcal{G},\Phi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
$n \leftarrow f_0.n$,  $Q \leftarrow \emptyset$
$(F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f_b)$
return $F$

proc INPUT($i, c_0, c_1$)
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\bot$
if $i \notin \{1, \ldots, n\} \setminus Q$ then return $\bot$
$x_{0,i} \leftarrow c_0$,  $x_{1,i} \leftarrow c_1$,  $Q \leftarrow Q \cup \{i\}$
return $X_i^{x_{b,i}}$
```

Fig. 10. Indistinguishability-based privacy notions. Games to define the ind-based static, adaptive, and projective security of $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. In each game, INITIALIZE() samples $b \twoheadleftarrow \{0,1\}$, and when FINALIZE($b'$) is unspecified, it returns $(b = b')$.

$\mathsf{Gb}(1^k, f) = (b, e, \varepsilon)$, where b is the last bit of f, $n = f.n$, and e is the $2n$-bit vector $(0, 1, \ldots, 0, 1)$. Let $\mathsf{En}(e, x) = x$, $\mathsf{Ev}(b, x) = P(b \parallel x)$, and $\mathsf{De}(\varepsilon, y) = y$. Let an (even computationally-unbounded) adversary $\mathcal{A}$ attack the prv2.ind.bad security of $\mathcal{G}$. Assume that $\mathcal{A}$ eventually produces (f_0, f_1, x_0, x_1) satisfying $\Phi(f_0) = \Phi(f_1)$ and $\mathsf{ev}^P(f_0, x_0) = \mathsf{ev}^P(f_1, x_1)$; otherwise $\mathcal{A}$'s advantage is 0. Since P is a permutation, $\mathsf{ev}^P(f_0, x_0) = \mathsf{ev}^P(f_1, x_1)$ implies that $x_0 = x_1$ and the last bits of f_0 and f_1 are equal, and consequently, $\mathcal{A}$'s advantage is still 0. On the other hand, consider the following adversary $\mathcal{B}$ attacking the prv1.ind security of $\mathcal{G}$. It queries $(0, 1)$ to the GARBLE oracle to receive the answer b. It then outputs b without querying the INPUT oracle, and wins with advantage 1.

RELATIONS AMONG PRIVACY NOTIONS. The following says that, as expected, prv1 security always implies prv1.ind security.

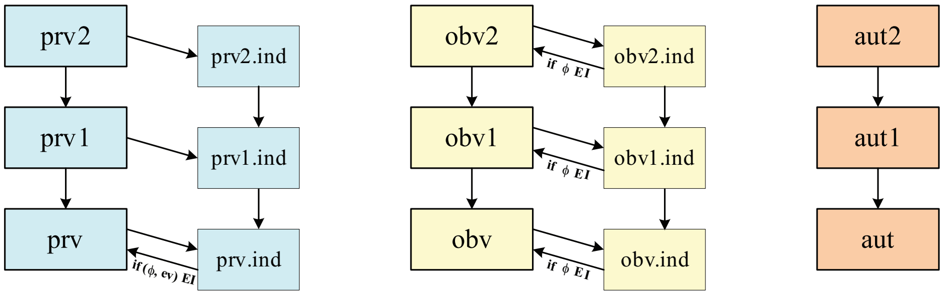

Fig. 11. Relations among security notions. A solid arrow is an implication; an if-labeled arrow, a conditional implication. Besides the implications given by the arrows and those inferred from them, any two notions are separated.

Proposition 13 $\mathsf{GS}(\mathrm{prv1}, \Phi) \subseteq \mathsf{GS}(\mathrm{prv1.ind}, \Phi)$ for any PT Φ.

Proof (Proposition 13). Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv1}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv1.ind}, \Phi)$. Let $\mathcal{A}$ be an adversary attacking the prv1.ind security of $\mathcal{G}$ over Φ. We construct a PT prv1-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to Garble, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own Garble oracle to get back (F, d) and returns this to $\mathcal{A}$. For the next query (x_0, x_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$ or $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ or $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$. Else it queries x_a to its own Input oracle to get X and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then for any $\mathcal{S}$ we have

$$\Pr\left[\, \mathrm{Prv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \;\middle|\; b = 1 \,\right] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{prv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k)$$
$$\Pr\left[\, \neg\mathrm{Prv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \;\middle|\; b = 0 \,\right] = \frac{1}{2}$$

where b denotes the challenge bit in game $\mathrm{Prv1}_{\mathcal{G},\Phi,\mathcal{S}}$. The second claim is true since $\mathcal{S}$ has the same input regardless of a, and thus whatever $\mathcal{A}$ receives is independent of a. Subtracting, we obtain

$$\mathbf{Adv}^{\mathrm{prv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \leq 2 \cdot \mathbf{Adv}^{\mathrm{prv1},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) \ .$$

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

The following says that prv2 security always implies prv2.ind security.

Proposition 14 $\mathsf{GS}(\mathrm{prv2}, \Phi) \subseteq \mathsf{GS}(\mathrm{prv2.ind}, \Phi)$ for any PT Φ.

Proof (Proposition 14). Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{prv2}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv2.ind}, \Phi)$. Let $\mathcal{A}$ be an adversary attacking the prv2.ind security of $\mathcal{G}$ over Φ. We construct a PT prv2-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to Garble, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$, and also returns $\perp$ to $\mathcal{A}$'s subsequent Input queries, if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own Garble oracle to get back (F, d) and returns this to $\mathcal{A}$. Then, for each query (i, c_0, c_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i, c_a) to its own Input oracle and returns the resulting token X_i to $\mathcal{A}$. The latter now returns a bit b'. Let x_0 and x_1 be the input strings resulting from the Input queries of $\mathcal{A}$. If $\Phi(f_0) = \Phi(f_1)$ and $\mathcal{A}$

makes its guess without fully specifying x_0 and x_1, then $\mathcal{B}$ returns 1 only if $b' = a$. Otherwise, $\mathcal{B}$ returns 1 only if $b' = a$ and $\mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. Then for any $\mathcal{S}$ we have

$$\Pr\left[\, \mathrm{Prv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \;\middle|\; b = 1 \,\right] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{prv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k)$$

$$\Pr\left[\, \neg\mathrm{Prv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \;\middle|\; b = 0 \,\right] = \frac{1}{2}$$

where b denotes the challenge bit in game $\mathrm{Prv2}_{\mathcal{G},\Phi,\mathcal{S}}$. The second claim is true since $\mathcal{S}$ has the same input regardless of a, and thus whatever $\mathcal{A}$ receives is independent of a. Subtracting, we see that

$$\mathbf{Adv}^{\mathrm{prv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A}, k) \le 2 \cdot \mathbf{Adv}^{\mathrm{prv2},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B}, k) \ .$$

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

BHR [5] show that for garbling schemes $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ with (Φ, ev) efficiently invertible, prv.ind security over Φ implies prv security over Φ. But analogous claims do not hold for adaptive privacy. Below, we will show that, prv2.ind security does not imply prv1 security even when (Φ, ev) is efficiently invertible.

Let $P : \{0,1\}^* \to \{0,1\}^*$ be a length-preserving permutation. Recall that P has a *hard-core predicate* $h : \{0,1\}^* \to \{0,1\}$ if advantage

$$2\Pr[x \twoheadleftarrow \{0,1\}^k;\ b \leftarrow A(P(x)) \colon\ b = h(x)] - 1$$

is negligible for every PPT adversary $\mathcal{A}$. Starting from any one-way permutation, one can construct another one-way permutation with a hard-core predicate, by the Goldreich-Levin construction [14].

In Proposition 15, for any string $f \in \{0,1\}^*$, we let $f.n = f.m = |f|$, and let $\mathsf{ev}^*(f, x) = f$, for any $x \in \{0,1\}^{|f|}$. Define $\Phi^*(f) = |f|$ for all $f \in \{0,1\}^*$. We note that (Φ^*, ev^*) is efficiently invertible.

Proposition 15 $\mathsf{GS}(\mathrm{prv2.ind}, \Phi^*) \cap \mathsf{GS}(\mathsf{ev}^*) \not\subseteq \mathsf{GS}(\mathrm{prv1}, \Phi^*)$, assuming the existence of a one-way, length-preserving permutation $P : \{0,1\}^* \to \{0,1\}^*$.

Proof. We build a projective scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}^*)$ so that $\mathcal{G} \in \mathsf{GS}(\mathrm{prv2.ind}, \Phi^*) \cap \mathsf{GS}(\mathsf{ev}^*)$ but $\mathcal{G} \notin \mathsf{GS}(\mathrm{prv1}, \Phi^*)$. Let $h : \{0,1\}^* \to \{0,1\}$ be a hard-core predicate of P. On input $(1^k, f)$, algorithm Gb samples $r_1, \ldots, r_n \twoheadleftarrow \{0,1\}^k$, and creates $F = (P(r_1), P(r_2), \ldots, P(r_n), S \oplus f)$, where $n = |f|$ and $S = h(r_1) \| \cdots \| h(r_n)$. It then picks $S_1, \ldots, S_n \twoheadleftarrow \{0,1\}^{kn}$, lets $e = (S_1, S_1, \ldots, S_n, S_n)$ and $d = (r_1 \| \cdots \| r_n) \oplus S_1 \oplus \cdots \oplus S_n$, and returns (F, e, d). Algorithm En is defined, as the scheme $\mathcal{G}$ is projective. Let Ev be the identity. Finally, on input d and $Y = (F, X)$, algorithm De parses $X = (S_1, \ldots, S_n)$ and $F = (s_1, \ldots, s_n, U)$. It then computes $r_1 \| \cdots \| r_n = d \oplus S_1 \oplus \cdots \oplus S_n$, where $k = |s_1|$ and each string r_i has length k. If $P(r_i) = s_i$ for all $i \le n$ then it returns $f = U \oplus S$, where $S = h(r_1) \| \cdots \| h(r_n)$, otherwise it returns $\bot$.

Consider an adversary $\mathcal{A}$ attacking the prv2.ind security of $\mathcal{G}$. Without loss of generality, assume that $\mathcal{A}$ queries (f_0, f_1) such that $\Phi^*(f_0) = \Phi^*(f_1)$. If $f_0 = f_1$ then the advantage of $\mathcal{A}$ will be 0, no matter how it queries oracle INPUT, so assume that $f_0 \ne f_1$. If $\mathcal{A}$ fully specifies its input strings x_0 and x_1 then $\mathsf{ev}^*(f_0, x_0) \ne \mathsf{ev}^*(f_1, x_1)$, and $\mathcal{A}$'s advantage is again 0. Otherwise, if $\mathcal{A}$ does not fully specify its input strings, then the strings S_i it obtains from oracle INPUT are independent random strings, so $\mathcal{A}$ gains nothing from querying Input. Then, $\mathcal{A}$'s advantage is negligible, since h is a hard-core predicate of P.

Next, consider the following adversary $\mathcal{B}(1^k)$ attacking the prv1 security of $\mathcal{G}$. It chooses $f \twoheadleftarrow \{0,1\}$ and queries f to its Garble oracle to get answer (F, d). It then queries 0 to oracle Input to receive answer X. It returns 1 only if $\mathsf{De}(d, \mathsf{Ev}(F, X)) = f$ and $F = (s, U)$, with $|s| = k$ and $|U| = 1$. If the challenge bit is 1 then $\mathcal{B}$'s guess is always correct. Suppose that the challenge bit is 0. Fix a computationally unbounded simulator $\mathcal{S}$. The simulator does not know if f is 0 or 1 until the last query, and thus f is independent of F and d. Let $F = (s, U)$. Since P is permutation, $r = P^{-1}(s)$ is uniquely defined, and thus $h(r)$ is also independent of f. Then $f \neq \mathsf{De}(d, \mathsf{Ev}(F, X)) = U \oplus h(r)$ with probability 1/2, no matter how the simulator chooses X. Thus, $\mathcal{B}$'s guess is correct with probability at least 1/2. Hence $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv1}, \Phi^*, \mathcal{S}}(\mathcal{B}, k) \geq 1/2$. □

Relations among obliviousness notions. Next we consider relations for obliviousness notions. The following says that obv1 security always implies obv1.ind security, and conversely if Φ is efficiently invertible.

Proposition 16 For any PT Φ: (1) $\mathsf{GS}(\mathrm{obv1}, \Phi) \subseteq \mathsf{GS}(\mathrm{obv1.ind}, \Phi)$, and (2) If Φ is efficiently invertible then $\mathsf{GS}(\mathrm{obv1.ind}, \Phi) \subseteq \mathsf{GS}(\mathrm{obv1}, \Phi)$.

Proof (Proposition 16). For part (1), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{obv1}, \Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv1.ind}, \Phi)$. Let $\mathcal{A}$ be an adversary attacking the obv1.ind security of $\mathcal{G}$ over Φ. We construct a PT obv1-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to Garble, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own Garble oracle to get back F and returns this to $\mathcal{A}$. For the next query (x_0, x_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ if $\Phi(f_0) \neq \Phi(f_1)$ or $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$. Else it queries x_a to its own Input oracle to get X and returns this to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then for any $\mathcal{S}$ we have

$$\Pr\left[\,\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{B}} \mid b = 1\,\right] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv1.ind}, \Phi}(\mathcal{A}, k)$$
$$\Pr\left[\,\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{B}} \mid b = 0\,\right] = \frac{1}{2}$$

where b denotes the challenge bit in game $\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}$. Subtracting, we see that

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv1.ind}, \Phi}(\mathcal{A}, k) \leq 2 \cdot \mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv1}, \Phi, \mathcal{S}}(\mathcal{B}, k) \ .$$

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

For part (2), let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) \in \mathsf{GS}(\mathrm{obv1.ind}, \Phi)$ and let M be a Φ-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv1}, \Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the obv1-security of $\mathcal{G}$ over Φ. We define a simulator $\mathcal{S}$ that on input $(1^k, \phi, 0)$, lets $f \leftarrow M(\phi)$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and returns F. For the next query, the simulator chooses $x \twoheadleftarrow \{0,1\}^{f.n}$ and then answers $X = \mathsf{En}(e, x)$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1 to Garble, adversary $\mathcal{A}$ lets $f_0 \leftarrow M(\Phi(f_1))$ and then queries f_0, f_1 to its own Garble oracle to get back F, which it returns to $\mathcal{B}$. When receiving x_1 on the next query, if $x_1 \not\in \{0,1\}^{f_1.n}$ then $\mathcal{A}$ returns $\perp$

to $\mathcal{B}$. Else it chooses $x_0 \twoheadleftarrow \{0,1\}^{f_0.n}$, queries (x_0, x_1) to its Input oracle, and returns the answer X to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. Then

$$\Pr\left[\,\mathrm{Obv1Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b=1\,\right] = \Pr\left[\,\mathrm{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c=1\,\right]$$
$$\Pr\left[\,\neg\mathrm{Obv1Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b=0\,\right] = \Pr\left[\,\neg\mathrm{Obv1}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c=0\,\right]$$

where b and c denote the challenge bit in game $\mathrm{ObvInd}_{\mathcal{G},\Phi}$ and $\mathrm{Obv1}_{\mathcal{G},\Phi,\mathcal{S}}$ respectively. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv1},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k) \le \mathbf{Adv}^{\mathrm{obv1.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)\ .$$

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

The following says that obv2 security always implies obv2.ind security, and conversely if Φ is efficiently invertible.

Proposition 17 For any PT Φ: (1) $\mathsf{GS}(\mathrm{obv2},\Phi) \subseteq \mathsf{GS}(\mathrm{obv2.ind},\Phi)$, and (2) If Φ is efficiently invertible then $\mathsf{GS}(\mathrm{obv2.ind},\Phi) \subseteq \mathsf{GS}(\mathrm{obv2},\Phi)$.

Proof (Proposition 17). For part (1), let $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}) \in \mathsf{GS}(\mathrm{obv2},\Phi)$. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv2.ind},\Phi)$. Let $\mathcal{A}$ be an adversary attacking the obv2.ind security of $\mathcal{G}$ over Φ. We construct a PT obv2-adversary $\mathcal{B}$ as follows. Let $\mathcal{B}(1^k)$ runs $\mathcal{A}(1^k)$. When the latter makes its query f_0, f_1 to Garble, adversary $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$, and also returns $\perp$ to $\mathcal{A}$'s subsequent Input queries, if $\Phi(f_0) \ne \Phi(f_1)$. Else it picks a bit a at random and queries f_a to its own Garble oracle to get back F and returns this to $\mathcal{A}$. Then, for each query (i, c_0, c_1) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i, c_a) to its own Input oracle and returns the resulting token X_i to $\mathcal{A}$. The latter now returns a bit b'. Adversary $\mathcal{B}$ returns 1 only if $b' = a$. Then for any $\mathcal{S}$ we have

$$\Pr\left[\,\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b=1\,\right] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{obv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k)$$
$$\Pr\left[\,\neg\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid b=0\,\right] = \frac{1}{2}$$

where b denotes the challenge bit in game $\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}$. Subtracting, we get

$$\mathbf{Adv}^{\mathrm{obv2.ind},\,\Phi}_{\mathcal{G}}(\mathcal{A},k) \le 2\cdot\mathbf{Adv}^{\mathrm{obv2},\,\Phi,\,\mathcal{S}}_{\mathcal{G}}(\mathcal{B},k)\ .$$

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

For part (2), let $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev}) \in \mathsf{GS}(\mathrm{obv2.ind},\Phi)$ and let M be a Φ-inverter. We want to show that $\mathcal{G} \in \mathsf{GS}(\mathrm{obv2},\Phi)$. Let $\mathcal{B}$ be a PT adversary attacking the obv2 security of $\mathcal{G}$ over Φ. We define a simulator $\mathcal{S}$ that, on input $(1^k, \phi, 0)$, lets $f \leftarrow M(\phi)$ then $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, where M is a Φ-inverter, and returns F. For each subsequent query (i, j), the simulator lets $e = (X^0_1, X^1_1, \ldots, X^0_n, X^1_n)$, chooses $x_i \twoheadleftarrow \{0,1\}$, and answers $X^{x_i}_i$. We define adversary $\mathcal{A}(1^k)$ to run $\mathcal{B}(1^k)$. When the latter makes its query f_1 to Garble, adversary $\mathcal{A}$ lets $f_0 \leftarrow M(\Phi(f_1))$ and then queries f_0, f_1 to its own Garble oracle to get back F, which it returns to $\mathcal{B}$. Then, for each query (i, c_1) of $\mathcal{B}$, the adversary $\mathcal{A}$ chooses $c_0 \twoheadleftarrow \{0,1\}$ and queries (i, c_0, c_1) to its Input oracle, and returns the resulting token X_i to $\mathcal{B}$. When the latter outputs a bit b' and halts, so does $\mathcal{A}$. Then

$$\Pr\left[\,\mathrm{Obv2Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b=1\,\right] = \Pr\left[\,\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c=1\,\right]$$
$$\Pr\left[\,\neg\mathrm{Obv2Ind}^{\mathcal{A}}_{\mathcal{G},\Phi} \mid b=0\,\right] = \Pr\left[\,\neg\mathrm{Obv2}^{\mathcal{B}}_{\mathcal{G},\Phi,\mathcal{S}} \mid c=0\,\right]$$

where b and c denote the challenge bit in game $\mathrm{Obv2Ind}_{\mathcal{G},\Phi}$ and $\mathrm{Obv2}_{\mathcal{G},\Phi,\mathcal{S}}$ respectively. Subtracting, we get

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv2},\Phi,\mathcal{S}}(\mathcal{B},k) \le \mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv2.ind},\Phi}(\mathcal{A},k) \ .$$

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

Equivalence in idealized models. In idealized models, define obv1.prom as obv1 security in which the simulator too has oracle access to the ideal primitives, and obv1.nprom as obv1 security in which the simulator doesn't have oracle access to the ideal primitives, and will itself reply to the oracle queries made by the adversary. Define obv2.prom and obv2.nprom likewise. Then, if Φ is efficiently invertible then (1) obv1.prom and obv1.nprom are equivalent, and (2) obv2.prom and obv2.nprom are equivalent.

For part (1), it suffices to show that obv1.prom security implies obv1.nprom security, since the latter obviously implies the former. By part (1) of Proposition 16, obv1.prom security implies obv1 security. The proof still holds, even if the simulator $\mathcal{S}$ uses the programmability power to collude with the obv1.prom adversary $\mathcal{B}$ to fool the obv1.ind adversary $\mathcal{A}$, because what $(\mathcal{S},\mathcal{B})$ receives is independent of $\mathcal{A}$'s challenge bit. Because Φ is efficiently invertible, by part (2) of Proposition 16, obv1.ind security then implies obv1.nprom security.

For part (2), it suffices to show that obv2.prom security implies obv2.nprom security, since the latter obviously implies the former. By part (1) of Proposition 17, obv2.prom security implies obv2 security. The proof still holds, even if the simulator $\mathcal{S}$ uses the programmability power to collude with the obv2.prom adversary $\mathcal{B}$ to fool the obv2.ind adversary $\mathcal{A}$, because what $(\mathcal{S},\mathcal{B})$ receives is independent of $\mathcal{A}$'s challenge bit. Because Φ is efficiently invertible, by part (2) of Proposition 17, obv2.ind security then implies obv2.nprom security.

C Separations

For each $\mathrm{xxx} \in \{\mathrm{prv}, \mathrm{obv}, \mathrm{aut}\}$, it is obvious that xxx2 security implies xxx1 security and that xxx1 security implies xxx security. We want to prove that the converse directions are not true, even for projective schemes. Moreover, there are separations among our notions of privacy, obliviousness, and authenticity. The relations among notions are illustrated in Fig. 11. Recall that $\mathsf{ev}_{\mathsf{circ}}$ names the canonical circuit-evaluation procedure defined in Appendix A.3.

We will use the scheme described by the top box of Fig. 12 to separate xxx and xxx1 notions. The idea is as follows. We append 0^k to the garbled input (which is harmless), except for the case that x is a "poisoned" point s. There, we instead append a k-bit random string t (which is unlikely to be 0^k). We choose s at random so that a static adversary is unlikely to query $x = s$, but then include s to the garbled function, making it is trivial for an adaptive adversary to choose $x = s$. To make sure that the probability to query $x = s$ (that is 2^{-n}) is negligible in terms of k, we only perform this trick if $n \ge k$. To deal with authenticity, we append the same string t above to d. Procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u); it returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. This creates a loophole for an adversary to win, if it can query (Y, t) for some string Y. However, an aut adversary is unlikely to know t, because t is disclosed only if it queries the poisoned point $x = s$.

To separate xxx1 and xxx2 notions, we will use the scheme described by the bottom box of Fig. 12. The idea is as follows. We choose a random $(n-1)$-bit string $V = v_1 \cdots v_{n-1}$, and want to "poison" points $x \in \{V \parallel 0, V \parallel 1\}$. In order to do this, we choose $(n-1)$-bit strings V_i^0, V_i^1 for every $i \le n$, and append V_i^b to token X_i^b. We let $V_n^0 = V$, making it trivial for a projective adaptive adversary to choose a poisoned point, by querying $(n, 0)$ to its Input oracle. Since V is random, an xxx1 adversary on the other hand may know V only *after* it already specifies its x, which is too late to query $x \in \{V \| 0, V \| 1\}$. Of course it can try to guess V, but then its chance of success is only 2^{1-n}. To make sure that the probability above is negligible in terms of k, we only perform this trick if $n > k$. The other strings V_i^b are independently random (so it's harmless to append to the tokens), except that the checksum of $V_1^{v_1}, \ldots, V_{n-1}^{v_{n-1}}$ (the shares corresponding to a poisoned point) is a random string t whose last bit is 0. To deal with authenticity, we append the same string t above to d. Procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u); it returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. This creates a loophole for an adversary to win, if it can query (Y, t) for some string Y. However, an aut1 adversary is unlikely to know t, because t is disclosed only if it queries a poisoned point $x \in \{V \| 0, V \| 1\}$.

Separations among privacy notions. The following says that prv security does not imply prv1 security, even for circuit-garbling schemes.

Proposition 18 $\mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 18). By assumption, $\mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 12. We claim that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n \ge k$; otherwise $\mathcal{G}'$ will inherit the prv security from $\mathcal{G}$, as it only appends the garbled function and decoding function with independent random strings, and the garbled input with 0^k. Unless $\mathcal{A}$ can query $x = s$, where s is the random string appended to the garbled function, the same argument applies and $\mathcal{G}'$ will again inherit the prv security from $\mathcal{G}$. However, since $s \twoheadleftarrow \{0,1\}^n$, the chance that $x = s$ is $2^{-n} \le 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the prv1 security of $\mathcal{G}'$. Choose two circuits f_0, f_1 of the same topology such that $f_0.n = k$ and $f_0(x) = f_1(\overline{x})$ for every $x \in \{0,1\}^k$. Pick $b \twoheadleftarrow \{0,1\}$ and query $f = f_b$ to the oracle Garble. When receiving answer (F, s), query $x = x_b$ to Input to receive (X, u), where $x_0 = \overline{s}$ and $x_1 = s$. Return 1 only if the last bit of u coincides with b. If the challenge bit is 1 then the adversary answers 0 only if $b = 1$ and the last bit of t is 0, which happens with probability 1/4. Otherwise, since the simulator's inputs are independent of b, the chance that the last bit of u is b is exactly 1/2. Hence the adversary wins with advantage 1/4. □

Similarly, the following proposition says that, even for projective circuit-garbling schemes, prv1 security doesn't imply prv2 security.

Proposition 19 $\mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 19). By assumption, $\mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 12. We claim that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv1}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathrm{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv2}, \Phi_{\mathrm{topo}})$.

```
proc $\mathsf{Gb}'(1^k, f)$
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
    $t \twoheadleftarrow \{0,1\}^k, \quad s \twoheadleftarrow \{0,1\}^{f.n}$
    return $\big((F, s), (e, s, t), (d, t)\big)$

proc $\mathsf{Ev}'(F', X')$
    $(F, s) \leftarrow F', \ (X, u) \leftarrow X'$
    return $\big(\mathsf{Ev}(F, X), \varepsilon\big)$

proc $\mathsf{En}'(e', x)$
    $(e, s, t) \leftarrow e', \ n \leftarrow |s|, \ k \leftarrow |t|, \ X \leftarrow \mathsf{En}(e, x)$
    if $x = s$ and $n \geq k$ then return $(X, t)$
    return $(X, 0^k)$
```

```
proc $\mathsf{Gb}'(1^k, f)$
    $n \leftarrow f.n, \ (F, (X_1^0, X_1^1, \ldots, X_n^0, X_n^1), d) \leftarrow \mathsf{Gb}(1^k, f)$
    for $i \in \{1, \ldots, n\}$ do $V_i^0, \ V_i^1 \twoheadleftarrow \{0,1\}^{n-1}$
    $v_1 \cdots v_{n-1} \leftarrow V_n^0, \quad t \twoheadleftarrow \{0,1\}^{n-2}0$
    if $n > k$ then $V_1^{v_1} \leftarrow t \oplus V_2^{v_2} \oplus \cdots \oplus V_{n-1}^{v_{n-1}}$
    else $t \twoheadleftarrow \{0,1\}^{k-1}$
    $e' \leftarrow \big((X_1^0, V_1^0), (X_1^1, V_1^1), \ldots, (X_n^0, V_n^0), (X_n^1, V_n^1)\big)$
    return $(F, e', (d, t))$

proc $\mathsf{En}'(e', x)$
    $(T_0^1, T_1^1, \ldots, T_n^0, T_n^1) \leftarrow e', \ x_1 \cdots x_n \leftarrow x$
    return $(T_1^{x_1}, \ldots, T_n^{x_n})$

proc $\mathsf{Ev}'(F, X')$
    $\big((X_1, V_1), \ldots, (X_n, V_n)\big) \leftarrow X'$
    $(X_1, \ldots, X_n) \leftarrow X$
    return $\big(\mathsf{Ev}(F, X), \varepsilon\big)$
```

Fig. 12. In both schemes (top and bottom), procedure $\mathsf{De}'(d', Y')$ parses d' as (d, t) and Y' as (Y, u). It returns 1 if $u = t$, and returns $\mathsf{De}(d, Y)$ otherwise. **Separation between** xxx **and** xxx1 **notions (top):** Garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ that separates prv and prv1 (and, later, separates obv from obv1 , and aut from aut1). It is built from a circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$. **Separation between** xxx1 **and** xxx2 **notions (bottom):** Garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ that separates prv1 from prv2 (and later separates obv1 from obv2, and aut1 from aut2 too). It is built from a projective circuit-garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n > k$; otherwise $\mathcal{G}'$ will inherit the prv1 security from $\mathcal{G}$, as it only appends tokens and decoding function with independent random strings. Let $V = V_n^0$ be the random string appended into token X_n^0. Unless the adversary queries $x \in \{V \| 0, V \| 1\}$, the same argument applies and $\mathcal{G}'$ will again inherit the prv1 security from $\mathcal{G}$. However, as $V \twoheadleftarrow \{0,1\}^{n-1}$, the chance that $x \in \{V \| 0, V \| 1\}$ is $2^{1-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the prv2 security of $\mathcal{G}'$. Choose two circuits f_0, f_1 of the same topology such that $f_0.n = k + 1$ and $f_0(x) = f_1(x \oplus 1^k 0)$ for every $x \in \{0,1\}^{k+1}$. Pick $b \twoheadleftarrow \{0,1\}$ and query $f = f_b$ to the oracle Garble. Then query $(k+1, 0)$ to Input to get answer (X_{k+1}, V_{k+1}). Let $V_{k+1} = v_1 \cdots v_k$. If $b = 1$ then query $(1, v_1), \ldots, (k, v_k)$ to Input. Else query $(1, \overline{v}_1), \ldots, (k, \overline{v}_k)$. Let the answers be $(X_1, V_1), \ldots, (X_k, V_k)$, and let $t = V_1 \oplus \cdots \oplus V_k$. Answer 1 only if the last bit of t is $\overline{b}$. If the challenge bit is 1 then the chance that $\mathcal{A}$ answers 1 is 3/4. If the challenge bit is 0, since the simulator's inputs are independent of b, the chance that the adversary answers 1 is 1/2. Hence $\mathcal{A}$ wins with advantage 1/4. □

Separations among obliviousness notions. The following says that obv security does not imply obv1 security, even for circuit-garbling schemes.

Proposition 20 $\mathsf{GS}(\text{obv}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{obv1}, \Phi_{\text{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 20). By assumption, $\mathsf{GS}(\text{obv}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 12. Following exactly the same security proof and attack in the proof of Proposition 18, we have $\mathcal{G}' \in \mathsf{GS}(\text{obv}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{obv1}, \Phi_{\text{topo}})$. □

Similarly, the following proposition says that, for projective circuit-garbling schemes, obv1 security doesn't imply obv2 security.

Proposition 21 $\mathsf{GS}(\text{obv1}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{obv2}, \Phi_{\text{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 21). By assumption, $\mathsf{GS}(\text{obv1}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 12. Following exactly the same security proof and attack in the proof of Proposition 19, we have $\mathcal{G}' \in \mathsf{GS}(\text{obv1}, \Phi_{\text{topo}}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{obv2}, \Phi_{\text{topo}})$. □

Separations among authenticity notions. The following says that aut security does not imply aut1 security, even for circuit-garbling schemes.

Proposition 22 $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{aut1})$, assuming that LHS is nonempty.

Proof (Proposition 22). By assumption, $\mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the top box of Fig. 12. We claim that $\mathcal{G}' \in \mathsf{GS}(\text{aut}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut1})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Let t be the random string that is appended to the decoding function. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n \geq k$; otherwise $\mathcal{G}'$ will inherit the aut security from $\mathcal{G}$, as it only appends the garbled function with an independent random string, and the garbled input with 0^k, and the chance that adversary can output $Y' = (Y, t)$ is at most 2^{-k}. Unless $\mathcal{A}$ can query $x = s$, where s is the random string appended to the garbled function, the same argument applies and $\mathcal{G}'$ will again inherit the aut security from $\mathcal{G}$. However, since $s \twoheadleftarrow \{0,1\}^n$, the chance that $x = s$ is $2^{-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the aut1 security of $\mathcal{G}'$. Query an arbitrary circuit f, such that $f.n = k$, to Garble to receive (F, s). Then, query $x = s$ to Input to receive (X, t). Then, output $(1, t)$ and win with advantage 1. □

Similarly, the following proposition says that, for projective circuit-garbling schemes, aut1 security does not imply aut2 security.

Proposition 23 $\mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \not\subseteq \mathsf{GS}(\text{aut2})$, assuming that LHS is nonempty.

Proof (Proposition 23). By assumption, $\mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}_{\mathsf{circ}})$ be a member of this set. Consider the garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Ev}', \mathsf{ev}_{\mathsf{circ}})$ described by the bottom box of Fig. 12. We claim that $\mathcal{G}' \in \mathsf{GS}(\text{aut1}) \cap \mathsf{GS}(\text{proj}) \cap \mathsf{GS}(\mathsf{ev}_{\mathsf{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut2})$.

Let us justify the first claim. Consider an adversary $\mathcal{A}$ that attacks $\mathcal{G}'$. Let t be the random string that is appended to the decoding function. Assume that the circuit f in $\mathcal{A}$'s query satisfies $n = f.n > k$; otherwise $\mathcal{G}'$ will inherit the aut1 security from $\mathcal{G}$, as it only appends each token with an independent random string, and the chance that the adversary can output $Y' = (Y, t)$ is 2^{1-k}. Let $V = V_n^0$ be the random string appended into token X_n^0. Unless the adversary queries $x \in \{V \| 0, V \| 1\}$, the same argument applies and $\mathcal{G}'$ will again inherit the aut1 security from $\mathcal{G}$. However, as $V \twoheadleftarrow \{0,1\}^{n-1}$, the chance that $x \in \{V \| 0, V \| 1\}$ is $2^{1-n} \leq 2^{-k}$.

We justify the second claim by constructing an adversary $\mathcal{A}$ that breaks the aut2 security of $\mathcal{G}'$. Query an arbitrary circuit f, such that $f.n = k + 1$, to Garble. Then, query $(k + 1, 0)$ to

Input to receive (X_{k+1}, V_{k+1}). Let $V_{k+1} = v_1 \cdots v_k$. Then, query $(1, v_1), \ldots, (k, v_k)$ to Input to receive $(X_1, V_1), \ldots, (X_k, V_k)$ respectively. Let $t = V_1 \oplus \cdots \oplus V_k$. Then, output $(1, t)$ and win with advantage 1. $\square$

Separations among privacy, obliviousness, and authenticity. The following says that privacy does not imply obliviousness, even when we take the strongest form of privacy (projective adaptive) and the weakest form of obliviousness (static).

Proposition 24 $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \not\subseteq \mathsf{GS}(\mathrm{obv}, \Phi)$ for all Φ, assuming that LHS is nonempty.

Proof (Proposition 24). By assumption, $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev_{circ}})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev_{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{obv}, \Phi)$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ create $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $((F, d), e, d)$. Let $\mathsf{Ev}'((F, d), X) = \mathsf{Ev}(F, X)$. Including d in the description of the garbled function does not harm prv2 security because an adversary is always given the descriptions of the garbled function and the decoding function simultaneously, so $\mathcal{G}'$ inherits the prv2 security of $\mathcal{G}$. On the other hand, scheme $\mathcal{G}'$ fails to achieve obv. Let $f_0 = f_1 = \mathrm{OR}$, $x_0 = 00$ and $x_1 = 11$. An adversary simply picks $b \twoheadleftarrow \{0, 1\}$ and queries (f_b, x_b). On receiving reply $((F, d), X, d)$, it outputs 1 if $\mathsf{De}(d, \mathsf{Ev}(F, X)) = b$ and outputs 0 otherwise. If the challenge bit is 1 then the adversary always answer 1. Otherwise, since the simulator's input is independent of b, the chance that the adversary answers 1 is 1/2. Hence the adversary wins with advantage 1/2. $\square$

The following says that obliviousness does not imply privacy, even when we take the strongest form of obliviousness (projective adaptive) and the weakest form of privacy (static).

Proposition 25 $\mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \not\subseteq \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 25). By assumption, $\mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev_{circ}})$ be a member of this set. We construct a garbling scheme $\mathcal{G}' = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}', \mathsf{Ev}, \mathsf{ev_{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\mathrm{obv2}, \Phi_{\mathrm{topo}}) \cap \mathsf{GS}(\mathsf{ev_{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\mathrm{prv}, \Phi_{\mathrm{topo}})$. The construction is as follows. Let $\mathsf{Gb}'(1^k, f)$ create $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and return $(F, e, (d, e))$. Let $\mathsf{De}'((d, e), Y) = \mathsf{De}(d, Y)$. Including e in the description of the decoding function does not harm obv2 security because an adversary is never given the description of the decoding function, so $\mathcal{G}'$ inherits the obv2 security of $\mathcal{G}$. On the other hand, $\mathcal{G}'$ fails to achieve prv. Let $f_0 = \mathrm{AND}$, $f_1 = \mathrm{OR}$, and $x_0 = x_1 = 11$. An adversary simply chooses $b \twoheadleftarrow \{0, 1\}$ and queries (f_b, x_b). On receiving reply $(F, X, (d, e))$, it outputs 0 if $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, 01))) = 0$ and outputs 1 otherwise. If the challenge bit is 1 then the adversary always answer 1. Otherwise, since the simulator's input is independent of b, the chance that the adversary answers 1 is 1/2. Hence the adversary wins with advantage 1/2. $\square$

The following says that privacy and obliviousness, even in conjunction and in their strongest forms (projective adaptive), do not imply authenticity, even in its weakest form (static).

Proposition 26 $\mathsf{GS}(\mathrm{prv2}, \Phi) \cap \mathsf{GS}(\mathrm{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \not\subseteq \mathsf{GS}(\mathrm{aut})$, for all Φ, assuming that LHS is nonempty.

Proof (Proposition 26). By assumption, $\mathsf{GS}(\text{prv2}, \Phi) \cap \mathsf{GS}(\text{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev_{circ}})$ be a member of this set. We construct $\mathcal{G}' = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}', \mathsf{Ev}', \mathsf{ev_{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{prv2}, \Phi) \cap \mathsf{GS}(\text{obv2}, \Phi) \cap \mathsf{GS}(\mathsf{ev_{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{aut})$. The construction is as follows. Let $\mathsf{Ev}'(F, X) = \mathsf{Ev}(F, X)\|0$, and let $\mathsf{De}'(d, Y\|b)$ be $\mathsf{De}(d, Y)$ if $b = 0$, and be 1 otherwise, where $b \in \{0, 1\}$. Appending a constant bit to the garbled output does not harm prv2 security or obv2 security. On the other hand, $\mathcal{G}'$ fails to achieve aut. An adversary simply makes query $(\text{OR}, 00)$ and outputs $1\|1$ to have advantage 1. □

The following says that authenticity, even in its strongest forms (projective adaptive), implies neither privacy nor obliviousness, even in their weakest form (static).

Proposition 27 $\mathsf{GS}(\text{aut2}) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \not\subseteq \mathsf{GS}(\text{prv}, \Phi_{\text{topo}}) \cup \mathsf{GS}(\text{obv}, \Phi_{\text{topo}})$, assuming that LHS is nonempty.

Proof (Proposition 27). By assumption, $\mathsf{GS}(\text{aut2}) \cap \mathsf{GS}(\mathsf{ev_{circ}}) \neq \emptyset$, so we let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev_{circ}})$ be a member of this set. We construct $\mathcal{G} = (\mathsf{Gb}', \mathsf{En}, \mathsf{De}, \mathsf{Ev}', \mathsf{ev_{circ}})$ such that $\mathcal{G}' \in \mathsf{GS}(\text{aut2}) \cap \mathsf{GS}(\mathsf{ev_{circ}})$ but $\mathcal{G}' \notin \mathsf{GS}(\text{prv}, \Phi_{\text{topo}}) \cup \mathsf{GS}(\text{obv}, \Phi_{\text{topo}})$. The construction is as follows. On input $(1^k, f)$, algorithm Gb' creates $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, and then outputs $((F, f), e, d)$. On input $((F, f), X)$, algorithm Ev' returns $\mathsf{Ev}(F, X)$. Appending f to F does no harm to authenticity of $\mathcal{G}'$, as the adversary always knows f in its attack. On the other hand, the garbled function leaks f, so both privacy and obliviousness fail over Φ_{topo}. □

D Postponed proofs

The variables specified in simulator code in these proofs are global ones, part of the state that it maintains and updates across its different invocations.

D.1 Proof of Theorem 2

Given any PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$ we build a PT adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$. The assumption of prv security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\text{prv}, \Phi, \mathcal{S}}(\mathcal{A}, \cdot)$ is negligible. Now we build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$,

$$\mathbf{Adv}_{\mathcal{G}_1}^{\text{prv1}, \Phi, \mathcal{S}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}_{\mathcal{G}}^{\text{prv}, \Phi, \mathcal{S}}(\mathcal{A}, k) \ . \tag{1}$$

This yields the theorem.

The constructions have to deal with some pesky issues related to the fact that the simulator needs to know the lengths of the pads, so let us settle these first. We know that algorithm Gb runs in polynomial time. This means there are polynomials L_1', L_2' such that if $(F, e, d) \in [\mathsf{Gb}(1^k, f)]$ then $|F|$ is at most $L_1'(k, |f|)$ and $|d|$ is at most $L_2'(k, |f|)$. By suitable padding, we assume wlog these lengths are exactly, rather than at most, $L_1'(k, |f|)$ and $L_2'(k, |f|)$, respectively. (Formally, $\mathcal{G}$ would have to be first transformed to ensure this condition via the suitable padding.) A convention we have made in Section 2 is that the side-information $\Phi(f)$ always reveals $f.n$, $f.m$ and $|f|$. This means there are PT functions L_1, L_2 such that $L_1(1^k, \Phi(f)) = L_1'(k, |f|)$ and $L_2(1^k, \Phi(f)) = L_2'(k, |f|)$.

Proceeding now to the constructions, we define $\mathcal{A}$ and $\mathcal{S}_1$ as in the top of Fig. 13. There, adversary $\mathcal{A}$ runs $\mathcal{A}_1$, simulating the latter's Garble and Input oracles via procedures GarbleSim and InputSim, respectively. The last of these invokes the Garble oracle from $\mathcal{A}$'s own $\text{Prv}_{\mathcal{G}, \Phi, \mathcal{S}}$

```
adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\textsc{GarbleSim},\textsc{InputSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\Phi(f))}$
return $(F_1, d_1)$

proc $\textsc{InputSim}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\perp$
$(F, X, d) \leftarrow \textsc{Garble}(f, x)$
$F' \leftarrow F_1 \oplus F$, $d' \leftarrow d_1 \oplus d$
return $(X, d', F')$
```

```
simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$, $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\phi)}$
return $(F_1, d_1)$

simulator $\mathcal{S}_1(y, 1)$
$(F, X, d) \leftarrow \mathcal{S}(1^k, y, \phi)$
$F' \leftarrow F_1 \oplus F$, $d' \leftarrow d_1 \oplus d$
return $(X, d', F')$
```

```
adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\textsc{GarbleSim},\textsc{InputSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$n \leftarrow f.n$, $j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$(F, d) \leftarrow \textsc{Garble}(f)$
return $(F, d)$

proc $\textsc{InputSim}(i, c)$
$x_i \leftarrow c$, $j \leftarrow j + 1$
if $j < n$ then $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
    $x \leftarrow x_1 \cdots x_n$, $(X_1, \ldots, X_n) \leftarrow \textsc{Input}(x)$
    $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
    $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return $T_i$
```

```
simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$, $S \leftarrow 0^\ell$
$(F, d) \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return $(F, d)$

simulator $\mathcal{S}_2(\tau_2, i, j)$
if $j < n$ then $S_i \twoheadleftarrow \{0,1\}^\ell$, $S \leftarrow S \oplus S_i$
else
    $y \leftarrow \tau_2$, $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(y, 1)$
    $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
    $Z \leftarrow (Z_1, \ldots, Z_n)$, $S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return $T_i$
```

Fig. 13. Top: constructed adversary and simulator for proof of Theorem 2. For the first query, return random F_1 and d_1. For the second query, given $y = \mathsf{ev}(f, x)$, produce the real triple (F, X, d), and return X with the one-time pads $F_1 \oplus F$ and $d_1 \oplus d$. **Bottom: constructed adversary and simulator for proof of Theorem 3.** Except for the last query, return random tokens. For the last query, given $y = \mathsf{ev}(f, x)$, produce the real tokens and create the last piece of secret masks so that the shares unmask the real tokens.

game. (The f in the $\textsc{Garble}$ call is the one that was earlier queried to $\textsc{GarbleSim}$.) The two phases of the simulator are specified separately. Letting b, b_1 be the challenge bits in games $\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}$ and $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$, respectively, we observe that

$$\Pr\left[\, \mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k) \;\middle|\; b = 1 \,\right] = \Pr\left[\, \mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k) \;\middle|\; b_1 = 1 \,\right]$$
$$\Pr\left[\, \neg\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k) \;\middle|\; b = 0 \,\right] = \Pr\left[\, \neg\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k) \;\middle|\; b_1 = 0 \,\right] .$$

Subtracting yields Eq. (1).

D.2 Proof of Theorem 3

Given any PT adversary $\mathcal{A}_2$ against the prv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$. Now the assumption of prv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{prv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,\cdot)$ is negligible. Now we build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}_2}^{\mathrm{prv2},\Phi,\mathcal{S}_2}(\mathcal{A}_2,k) \le \mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{prv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,k)\ . \tag{2}$$

This yields the theorem.

We may assume wlog (again, formally, by first transforming the algorithms of $\mathcal{G}_1$ via suitable padding if necessary) that there is a PT function L' such that if $(F,e,d) \in [\mathsf{Gb}_1(1^k,f)]$ and $e = (X_1^0,X_1^1,\ldots,X_{f.n}^0,X_{f.n}^1)$ and $x \in \{0,1\}^{f.n}$ and $X = (X_1,\ldots,X_{f.n}) = \mathsf{En}(e,x)$ and $(\ell,\ell_1,\ldots,\ell_{f.n}) \leftarrow L'(1^k,|f|,f.n)$ then $|X| = \ell$ and $|X_i^0| = |X_i^1| = \ell_i$ for all $1 \le i \le f.n$. Now, since $\Phi(f)$ is assumed to always reveal $f.n, f.m$ and $|f|$, there is a PT function L such that $L(1^k,\Phi(f)) = L'(1^k,|f|,f.n)$.

Proceeding now to the constructions, we define $\mathcal{A}_1$ and $\mathcal{S}_2$ as in the bottom of Fig. 13. There, adversary $\mathcal{A}_1$ runs $\mathcal{A}_2$, simulating the latter's GARBLE and INPUT oracles via procedures GARBLESIM and INPUTSIM, respectively. These invoke GARBLE and INPUT oracles from $\mathcal{A}_1$'s own $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ game. The first phase of the simulator is specified first, and in the second piece of code, $j \in \{1,\ldots,n\}$. The simulator gets n from ϕ. Letting b_1,b_2 be the challenge bits in games $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ and $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$, respectively, we observe that

$$\Pr\left[\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k)\ \mid\ b_1=1\right] = \Pr\left[\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k)\ \mid\ b_2=1\right]$$
$$\Pr\left[\neg\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k)\ \mid\ b_1=0\right] = \Pr\left[\neg\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k)\ \mid\ b_2=0\right]\ .$$

Subtracting yields Eq. (2).

D.3 Proof of Theorem 4

Given any PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$ we build a PT adversary $\mathcal{A}$ against the prv security of $\mathcal{G}$. The assumption of prv security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv},\Phi,\mathcal{S}}(\mathcal{A},\cdot)$ is negligible. Now we build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$,

$$\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{prv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,k) \le \mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv},\Phi,\mathcal{S}}(\mathcal{A},k) + Q(k)/2^k$$

where Q is a polynomial such that $Q(k)$ upper bounds the total number of queries to HASH (made either directly by $\mathcal{A}_1$ or by scheme algorithms) in the execution of game $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}$ with $\mathcal{A}_1$ on input 1^k. This yields the theorem.

Let L_1, L_2 be as in the proof of Theorem 2, that is, L_1 and L_2 are PT functions that give the length of the pads masking the garbled function F and decoding function d respectively. The constructions of $\mathcal{A}$ and $\mathcal{S}_1$ are then provided at the top of Fig. 14, and H is a global variable maintained by the simulator, representing the current state of the simulated RO. Let game Hy be identical to $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k)$ with challenge bit $b=0$, but set a flag *bad* if $\mathcal{A}_1$ can query (ℓ,w) to the random oracle such that R is the suffix of w, prior to receiving R from the garbled input, where R is the seed generating the pads. If game Hy doesn't set *bad* then it is identical to game $\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k)$ with challenge bit $c=0$. Then

$$\Pr[\mathrm{Prv}_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{A}}(k) \mid c=0] - \Pr[\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k) \mid b=0] \le \Pr[\mathrm{Hy}^{\mathcal{A}_1}(k) \text{ sets } \mathit{bad}] \le Q(k)/2^k\ .$$

```
adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\textsc{GarbleSim},\textsc{InputSim},\textsc{HashSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$,  $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\Phi(f))}$
return $(F_1, d_1)$

proc $\textsc{InputSim}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$(F, X, d) \leftarrow \textsc{Garble}(f, x)$,  $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$,  $\mathsf{H}[|d|, 1\|R] \leftarrow d_1 \oplus d$
return $(X, R)$

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

```
simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$,  $d_1 \twoheadleftarrow \{0,1\}^{L_2(1^k,\phi)}$
return $(F_1, d_1)$

simulator $\mathcal{S}_1(y, 1)$
$(F, X, d) \leftarrow \mathcal{S}(1^k, y, \phi)$,  $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$,  $\mathsf{H}[|d|, 1\|R] \leftarrow d_1 \oplus d$
return $(X, R)$

simulator $\mathcal{S}_1(\ell, w, \texttt{ro})$
if $\mathsf{H}[\ell, w] = \bot$ then $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$
```

```
adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\textsc{GarbleSim},\textsc{InputSim},\textsc{HashSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$n \leftarrow f.n$,  $j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$(F, d) \leftarrow \textsc{Garble}(f)$
return $(F, d)$

proc $\textsc{InputSim}(i, c)$
$x_i \leftarrow c$,  $j \leftarrow j + 1$,  $S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ then
    $x \leftarrow x_1 \cdots x_n$,  $(X_1, \ldots, X_n) \leftarrow \textsc{Input}(x)$
    $S \leftarrow S_1 \oplus \cdots \oplus S_n$
    for $t \in \{1, \ldots, n\}$ do $\mathsf{H}[\ell_t, 1\|t\|S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return $T_i$

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

```
simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
$(F, d) \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return $(F, d)$

simulator $\mathcal{S}_2(\tau_2, i, j)$
$S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ then
    $y \leftarrow \tau_2$,  $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(y, 1)$
    $S \leftarrow S_1 \oplus \cdots \oplus S_n$
    for $t \in \{1, \ldots, n\}$ do $\mathsf{H}[\ell_t, 1\|t\|S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return $T_i$

simulator $\mathcal{S}_2(r, w, \texttt{ro})$
if $\mathsf{H}[r, w] = \bot$ then $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$
```

Fig. 14. Top: constructed adversary and simulator for proof of Theorem 4. For the first query, return random F_1 and d_1. For the second query, given $y = \mathsf{ev}(f, x)$, produce the real triple (F, X, d), choose a random seed $R \twoheadleftarrow \{0,1\}^k$, and program the RO so that the pads $F_1 \oplus F$ and $d_1 \oplus d$ are indeed $\textsc{Hash}(|F|, 0\|R)$ and $\textsc{Hash}(|d|, 1\|R)$ respectively. **Bottom: constructed adversary and simulator for proof of Theorem 5.** Except for the last query, return random tokens. For the last query, given $y = \mathsf{ev}(f, x)$, produce the real tokens, choose a random seed $S \twoheadleftarrow \{0,1\}^k$, and program the RO so that the shares unmask the real tokens.

On the other hand, $\Pr[\mathrm{Prv}^{\mathcal{A}}_{\mathcal{G},\Phi,\mathcal{S}}(k) \mid c = 1] = \Pr[\mathrm{Prv1}^{\mathcal{A}_1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}(k) \mid b = 1]$. Subtracting, we get the claimed bound.

D.4 Proof of Theorem 5

Given any PT adversary $\mathcal{A}_2$ against the prv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the prv1 security of $\mathcal{G}_1$. Now the assumption of prv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{prv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,\cdot)$ is negligible. Now we build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}_2}^{\mathrm{prv2},\Phi,\mathcal{S}_2}(\mathcal{A}_2,k) \le \mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{prv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,k) + Q(k)/2^k \ .$$

where Q is a polynomial such that $Q(k)$ upper bounds the total number of queries to HASH (made either directly by $\mathcal{A}_2$ or by scheme algorithms) in the execution of game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}$ with $\mathcal{A}_2$ on input 1^k. This yields the theorem.

Let L be as in the proof of Theorem 3, that is, L is a PT function that gives the lengths of the pads masking the tokens. The constructions of $\mathcal{A}_1$ and $\mathcal{S}_2$ are then provided at the bottom of Fig. 14. Let game Hy be identical to game $\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k)$ with challenge bit $b = 0$, but sets *bad* if $\mathcal{A}_2$ can query (r, w) to the random oracle such that S is the suffix of w, prior to receiving the entire garbled input, where S is the seed generating the pads. If Hy doesn't set *bad* then it is identical to game $\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}_1}^{\mathcal{A}_1}(k)$ with challenge bit $c = 0$. Then

$$\Pr[\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}}^{\mathcal{A}_1}(k) \mid c=0] - \Pr[\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k) \mid b=0] \le \Pr[\mathrm{Hy}^{\mathcal{A}_2}(k) \text{ sets } bad] \le Q(k)/2^k \ .$$

On the other hand, $\Pr[\mathrm{Prv1}_{\mathcal{G}_1,\Phi,\mathcal{S}}^{\mathcal{A}_1}(k) \mid c=1] = \Pr[\mathrm{Prv2}_{\mathcal{G}_2,\Phi,\mathcal{S}_2}^{\mathcal{A}_2}(k) \mid b=1]$. Subtracting, we get the claimed bound.

D.5 Proof of Theorem 8

For part (1), by adapting the proof of Theorem 2, we can show that if $\mathcal{G}$ is obv secure then scheme $\mathcal{G}' = \textsf{prv-to-prv1}[\mathcal{G}]$ is obv1 secure. Concretely, given any PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}'$ we build a PT adversary $\mathcal{A}$ against the obv security of $\mathcal{G}$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\Phi,\mathcal{S}}(\mathcal{A},\cdot)$ is negligible. We build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}'}^{\mathrm{obv1},\Phi,\mathcal{S}_1}(\mathcal{A}_1,k) \le \mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\Phi,\mathcal{S}}(\mathcal{A},k) \ . \tag{3}$$

The code of $\mathcal{A}$ and $\mathcal{S}_1$ is shown in the top box of Fig. 15, with L_1 as in the proof of Theorem 2, that is, L_1 is a PT function that gives the length of the pad masking the garbled function F. The analysis is analogous to the proof of Theorem 2.

Now for each xxx $\in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G}$ is xxx secure then $\mathcal{G}'$ is xxx1 secure. In scheme $\mathcal{G}'$, the decoding function is $d \oplus d'$, and the garble input is (X, d', F'), whereas in scheme $\mathcal{G}_1 = \textsf{all-to-all1}[\mathcal{G}]$, the former is $(d \oplus d', K)$, and the latter is $(X, d', F', \mathsf{F}_K(d'))$, with $K \twoheadleftarrow \{0,1\}^k$. Scheme $\mathcal{G}_1$ thus can be re-interpreted as scheme $\mathcal{G}'$, with a different encoding of the garbled input and decoding function. Hence $\mathcal{G}_1$ is also xxx1 secure.

For part (2), fix an adversary $\mathcal{A}_1$. We claim that there are adversaries $\mathcal{A}$ and $\mathcal{B}$ such that

$$\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{aut1}}(\mathcal{A}_1,k) \le 2^{-k} + \mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A}) + \mathbf{Adv}_{\mathsf{F}}^{\mathrm{prf}}(\mathcal{B},k) \ .$$

Moreover, the running time of $\mathcal{A}$ is at most that of $\mathcal{A}_1$ plus the time to garble $\mathcal{A}_1$'s queries, and so is the running time of $\mathcal{B}$. Let L_1 and L_2 be as in the proof of Theorem 2, that is, L_1 and L_2 are PT functions that give the length of the pads masking the garbled function F and decoding function d

```
adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\textsc{GarbleSim},\textsc{InputSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return $F_1$

proc $\textsc{InputSim}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$(F, X) \leftarrow \textsc{Garble}(f, x),\ \ F' \leftarrow F_1 \oplus F$
return $(X, F')$
```

```
simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$
return $F_1$

simulator $\mathcal{S}_1(1)$
$(F, X) \leftarrow \mathcal{S}(1^k, \phi),\ \ F' \leftarrow F_1 \oplus F$
return $(X, F')$
```

```
adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\textsc{GarbleSim},\textsc{InputSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$n \leftarrow f.n,\ \ j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f)),\ \ S \leftarrow 0^\ell$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$F \leftarrow \textsc{Garble}(f)$
return $F$

proc $\textsc{InputSim}(i, c)$
$x_i \leftarrow c,\ \ j \leftarrow j + 1$
if $j < n$ then $S_i \twoheadleftarrow \{0,1\}^\ell,\ \ S \leftarrow S \oplus S_i$
else
    $x \leftarrow x_1 \cdots x_n,\ \ (X_1, \ldots, X_n) \leftarrow \textsc{Input}(x)$
    $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
    $Z \leftarrow (Z_1, \ldots, Z_n),\ \ S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return $T_i$
```

```
simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f)),\ \ S \leftarrow 0^\ell,\ \ F \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return $F$

simulator $\mathcal{S}_2(i, j)$
if $j < n$ then $S_i \twoheadleftarrow \{0,1\}^\ell,\ \ S \leftarrow S \oplus S_i$
else
    $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(1)$
    $(Z_1, \ldots, Z_n) \leftarrow (X_1 \oplus U_1, \ldots, X_n \oplus U_n)$
    $Z \leftarrow (Z_1, \ldots, Z_n),\ \ S_i \leftarrow Z \oplus S$
$T_i \leftarrow (U_i, S_i)$
return $T_i$
```

Fig. 15. Top: constructed adversary and simulator from part (1) of the proof of Theorem 8. For the first query, return random F_1. For the second query, produce the real pair (F, X), and return X with the one-time pad $F_1 \oplus F$. **Bottom: constructed adversary and simulator from part (1) of the proof of Theorem 9.** Except for the last query, return random tokens. For the last query, produce the real tokens and create the last piece of secret masks so that the shares unmask the real tokens.

respectively. Consider the games $\mathrm{G}_0 - \mathrm{G}_2$ in Fig. 17. In each game, $\mathcal{A}_1$ has oracle access to procedure GarbleSim to get the garbled function and decoding function, and to procedure InputSim to get the garbled input. Game G_0 corresponds to game $\mathrm{Aut1}_{\mathcal{G}_1}$. We explain the game chain up until the terminal game. $\triangleright$ $\mathrm{G}_0 \to \mathrm{G}_1$: we use the technique of "swapping dependent and independent variables". Namely, instead of sampling F' and then computing $F_1 \leftarrow F \oplus F'$, we sample F_1 and then let $F' \leftarrow F \oplus F_1$. Then, we can move the garbling of f and the construction of K, d_1 to procedure Input, as the outputs of those commands are not used until then. The transition is conservative. Hence $\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{aut1}}(\mathcal{A}_1, k) = \Pr[\mathrm{G}_0^{\mathcal{A}_1}(k)] = \Pr[\mathrm{G}_1^{\mathcal{A}_1}(k)]$. $\triangleright$ $\mathrm{G}_1 \to \mathrm{G}_2$: in game G_1 we use $\mathcal{A}_1$'s output to recover the decoding function d, but in game G_2 we retrieve the correct d from memory. The two games are identical until G_2 sets *bad*.

Adversary $\mathcal{A}(1^k)$ runs $\mathcal{A}_1(1^k)$. When the latter makes queries, the former replies via the code of the top box of Fig. 16. Then, $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A}, k) = \Pr[\mathrm{G}_2^{\mathcal{A}_1}(k)]$, since the decoding function to authenticate $\mathcal{A}$'s output is the *correct* one instead of the one recovered from $\mathcal{A}_1$'s output. Adversary $\mathcal{B}(1^k)$

```
adversary $\mathcal{A}(1^k)$
    $Y_1 \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim}}(1^k)$
    $(Y, \text{tag}, V) \leftarrow Y_1$
    if $\text{tag} \neq \mathsf{F}_K(V)$ then return $\bot$
    return $Y$

proc $\text{GarbleSim}(f)$
    $F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$,   $d' \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$
    return $F_1$

proc $\text{InputSim}(x)$
    if $x \notin \{0,1\}^{f.n}$ then return $\bot$
    $(F, X) \leftarrow \text{Garble}(f, x)$,   $K \twoheadleftarrow \{0,1\}^k$,   $F' \leftarrow F \oplus F_1$
    return $(X, d', F', \mathsf{F}_K(d'))$
```

```
adversary $\mathcal{B}(1^k)$
    $Y_1 \leftarrow \mathcal{A}_1^{\text{GarbleSim},\text{InputSim}}(1^k)$
    $(Y, \text{tag}, V) \leftarrow Y_1$
    if $V \neq d'$ and $\text{Fn}(V) = \text{tag}$ then return 1
    return 0

proc $\text{GarbleSim}(f)$
    $F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$,   $d' \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$
    return $F_1$

proc $\text{InputSim}(x)$
    if $x \notin \{0,1\}^{f.n}$ then return $\bot$
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,   $X \leftarrow \mathsf{En}(e, x)$,   $F' \leftarrow F \oplus F_1$
    return $(X, d', F', \text{Fn}(d'))$
```

Fig. 16. Top: constructed adversary $\mathcal{A}$ for part (2) of the proof of Theorem 8. Its aut advantage is $\mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k)$ if $\mathcal{A}_1$ decides to output $V = d'$, as their outputs will be authenticated by the same d. Otherwise, $\mathcal{A}_1$ must forge (V, tag) that bypasses the test $\text{tag} = \mathsf{F}_K(V)$. **Bottom: constructed PRF adversary $\mathcal{B}$ for part (2) of the proof of Theorem 8.** It feeds $\mathcal{A}_1$ with correct F_1 and X_1. When $\mathcal{A}_1$ outputs $Y_1 = (Y, \text{tag}, V)$, if $V \neq d'$ then $\mathcal{B}$ queries V to its Fn oracle to test if $\text{tag} = \text{Fn}(V)$.

has an oracle Fn that implements either $\mathsf{F}_K(\cdot)$ or a truly random function. It runs $\mathcal{A}_1(1^k)$, and follows the code of the bottom box of Fig. 16. If the challenge bit b of the PRF game is 0, that is, the oracle Fn implements a truly random function, then $\mathcal{B}$ will answer 1 with probability 2^{-k}. On the other hand, if $b = 1$ then $\mathcal{B}$ answers correctly if only if $\mathcal{A}_1$ can forge a pair (V, tag) that bypasses the test $\mathsf{F}_K(V) = \text{tag}$. This, in other words, means that $\mathcal{A}$ can set the flag *bad* in game G_2 to be true. Then,

$$\begin{aligned}\Pr[\text{PRF}_{\mathsf{F}}^{\mathcal{B}}(k) \mid b = 1] &= \Pr[\mathsf{BAD}(\text{G}_2^{\mathcal{A}_1}(k))] \\ &\geq \Pr[\text{G}_2^{\mathcal{A}_1}(k)] - \Pr[\text{G}_1^{\mathcal{A}_1}(k)] = \mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k) - \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k)\ .\end{aligned}$$

Hence $\mathbf{Adv}_{\mathsf{F}}^{\text{prf}}(\mathcal{B}, k) \geq \mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k) - \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k) - 2^{-k}$, as claimed.

For part (3), if the encoding function e of $\mathcal{G}$ encodes $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$ then the encoding function e_1 of $\mathcal{G}_1$ can be re-interpreted as $(T_1^0, T_1^1, \ldots, T_n^0, T_n^1)$, where $T_i^b = (X_i^b, d', F', \mathsf{F}_K(d'))$ if $i = n$, and $T_i^b = X_i^b$ if $i < n$, for $b \in \{0,1\}$. Then, for $x = x_1 \cdots x_n$, the garbled input of $\mathcal{G}_1$ is

$$\big((X_1^{x_1}, \ldots, X_n^{x_n}), d', F', \mathsf{F}_K(d')\big) = (T_1^{x_1}, \ldots, T_n^{x_n}),$$

and the scheme $\mathcal{G}_1$ is therefore projective.

D.6 Proof of Theorem 9

Let $\mathcal{G}_2 = \mathsf{all1\text{-}to\text{-}all2}[\mathcal{G}_1]$. For part (1), it suffices to give the proof for the obliviousness case. The proof is similar to that of Theorem 3. Given any PT adversary $\mathcal{A}_2$ against the obv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}_1$. Now the assumption of obv1 security

```
Game $G_0$

proc GarbleSim($f$)
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$, $K \twoheadleftarrow \{0,1\}^k$
    $F' \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$,   $d' \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$
    $F_1 \leftarrow F \oplus F'$,   $d_1 \leftarrow (d \oplus d', K)$
    return $F_1$

proc InputSim($x$)
    if $x \notin \{0,1\}^{f.n}$ then return $\bot$
    $X \leftarrow \mathsf{En}(e, x)$
    return $(X, d', F', \mathsf{F}_K(d'))$

proc Finalize($d_1, Y_1$)
    if $x \notin \{0,1\}^{f.n}$ then return 0
    $(D, K) \leftarrow d_1$,   $(Y, \text{tag}, V) \leftarrow Y_1$
    if $\text{tag} \neq \mathsf{F}_K(V)$ then return false
    $d \leftarrow D \oplus V$
    return $(\mathsf{De}(d, Y) \neq \bot \wedge Y \neq \mathsf{Ev}(F, X))$
```

```
Games $G_1$/$G_2$

proc GarbleSim($f$)
    $F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k, \Phi(f))}$,   $d' \twoheadleftarrow \{0,1\}^{L_2(1^k, \Phi(f))}$
    return $F_1$

proc Input($x$)
    if $x \notin \{0,1\}^{f.n}$ then return $\bot$
    $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,   $X \leftarrow \mathsf{En}(e, x)$
    $F' \leftarrow F \oplus F_1$,   $K \twoheadleftarrow \{0,1\}^k$,   $d_1 \leftarrow (d \oplus d', K)$
    return $(X, d', F', \mathsf{F}_K(d'))$

proc Finalize($d_1, Y_1$)
    if $x \notin \{0,1\}^{f.n}$ then return 0
    $(D, K) \leftarrow d_1$,   $(Y, \text{tag}, V) \leftarrow Y_1$
    if $\text{tag} \neq \mathsf{F}_K(V)$ then return false
    if $d' \neq V$ then $bad \leftarrow$ true,  $V \leftarrow d'$      $\longleftarrow$ Use this in $G_2$
    $d \leftarrow D \oplus V$
    return $(\mathsf{De}(d, Y) \neq \bot \wedge Y \neq \mathsf{Ev}(F, X))$
```

Fig. 17. Games used in part (2) of the proof of Theorem 8. In procedure Finalize of game G_2, we make sure that V must be $d' = d \oplus D$ before we do the assignment $d \leftarrow D \oplus V$, and thus keep d unchanged.

yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}_{\mathcal{G}_1}^{\text{obv1}, \Phi, \mathcal{S}_1}(\mathcal{A}_1, \cdot)$ is negligible. We build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}_2}^{\text{obv2}, \Phi, \mathcal{S}_2}(\mathcal{A}_2, k) \leq \mathbf{Adv}_{\mathcal{G}_1}^{\text{obv1}, \Phi, \mathcal{S}_1}(\mathcal{A}_1, k) \ . \tag{4}$$

The code of $\mathcal{A}_1$ and $\mathcal{S}_2$ is given in the bottom box of Fig. 15, with L as in the proof of Theorem 3 (that is, L is a PT function that gives the length of the pads masking the tokens).

For part (2), we reuse the procedures GarbleSim and InputSim in part (1). Let $\mathcal{A}_2$ attack the aut2 security of $\mathcal{G}_2$. Adversary $\mathcal{A}_1(1^k)$ runs $\mathcal{A}_2(1^k)$, simulating the latter's Garble and Input oracles via procedures GarbleSim and InputSim respectively. When $\mathcal{A}_2$ outputs Y, adversary $\mathcal{A}_1$ also outputs Y. Let X and T be the garbled input given to $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Then

$$\begin{aligned}\mathbf{Adv}_{\mathcal{G}_2}^{\text{aut2}}(\mathcal{A}_2, k) &= \Pr[Y \neq \mathsf{Ev}_2(F, T) \wedge \mathsf{De}_1(d, Y) \neq \bot] \\ &= \Pr[Y \neq \mathsf{Ev}_1(F, X) \wedge \mathsf{De}_1(d, Y) \neq \bot] = \mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k);\end{aligned}$$

the second equality holds because $\mathsf{Ev}_2(F, T) = \mathsf{Ev}_1(F, X)$.

D.7 Proof of Theorem 10

For part (1), by adapting the proof of Theorem 4, we can show that if $\mathcal{G}$ is obv secure then scheme $\mathcal{G}' = \mathsf{rom\text{-}all\text{-}to\text{-}all1}[\mathcal{G}]$ is obv1 secure. Concretely, given any PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}'$ we build a PT adversary $\mathcal{A}$ against the obv security of $\mathcal{G}$. Now the assumption of

```
adversary $\mathcal{A}(1^k)$
$b' \leftarrow \mathcal{A}_1^{\textsc{GarbleSim},\textsc{InputSim},\textsc{HashSim}}(1^k)$
return $b'$

proc $\textsc{GarbleSim}(f)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return $F_1$

proc $\textsc{InputSim}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$(F, X) \leftarrow \textsc{Garble}(f, x)$,  $R \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$
return $(X, R)$

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

simulator $\mathcal{S}_1(1^k, \phi, 0)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\phi)}$
return $F_1$

simulator $\mathcal{S}_1(1)$
$(F, X) \leftarrow \mathcal{S}(1^k, \phi)$,  $R \twoheadleftarrow \{0,1\}^k$
 $\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$
return $(X, R)$

simulator $\mathcal{S}_1(\ell, w, \mathtt{ro})$
if $\mathsf{H}[\ell, w] = \bot$ then $\mathsf{H}[\ell, w] \twoheadleftarrow \{0,1\}^\ell$
return $\mathsf{H}[\ell, w]$
```

```
adversary $\mathcal{A}(1^k)$
$Y_1 \leftarrow \mathcal{A}_1^{\textsc{GarbleSim},\textsc{InputSim},\textsc{HashSim}}(1^k)$
$(Y, R', \mathrm{tag}) \leftarrow Y_1$
if $\mathrm{tag} \neq \textsc{HashSim}(k, K \parallel R')$ then return $\bot$
return $Y$

proc $\textsc{GarbleSim}(f)$
$F_1 \twoheadleftarrow \{0,1\}^{L_1(1^k,\Phi(f))}$
return $F_1$

proc $\textsc{InputSim}(x)$
if $x \notin \{0,1\}^{f.n}$ then return $\bot$
$(F, X) \leftarrow \textsc{Garble}(f, x)$,  $R \twoheadleftarrow \{0,1\}^k$,  $K \twoheadleftarrow \{0,1\}^k$
$\mathsf{H}[|F|, 0\|R] \leftarrow F_1 \oplus F$,  $\mathrm{tag} \leftarrow \textsc{HashSim}[k, K \parallel R]$
return $(X, R, \mathrm{tag})$

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

Fig. 18. Top: constructed adversary and simulator used in part (1) of the proof of Theorem 10. For the first query, return random F_1. For the second query, produce the real pair (F, X), choose a random seed $R \twoheadleftarrow \{0,1\}^k$, and program the RO so that the pad $F_1 \oplus F$ is indeed $\textsc{Hash}(|F|, 0\|R)$. **Bottom: constructed adversary for part (2) of the proof of Theorem 10.** When $\mathcal{A}_1$ outputs $Y_1 = (Y, R', \mathrm{tag})$, perform the test $\mathrm{tag} \neq \textsc{HashSim}(k, K \parallel R')$ as in algorithm De_1 of $\mathcal{G}_1$, and output Y if this test is passed.

obv1 security yields a PT simulator $\mathcal{S}$ for $\mathcal{A}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\,\Phi,\,\mathcal{S}}(\mathcal{A}, \cdot)$ is negligible. We build from $\mathcal{S}$ a PT simulator $\mathcal{S}_1$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}'}^{\mathrm{obv1},\,\Phi,\,\mathcal{S}_1}(\mathcal{A}_1, k) \leq \mathbf{Adv}_{\mathcal{G}}^{\mathrm{obv},\,\Phi,\,\mathcal{S}}(\mathcal{A}, k) + Q(k)/2^k,$$

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to $\textsc{Hash}$ (made either directly by $\mathcal{A}_1$ or by the scheme algorithms) in the execution of game $\mathrm{Obv1}_{\mathcal{G}',\Phi,\mathcal{S}_1}$ with $\mathcal{A}_1$ on input 1^k. The code of $\mathcal{A}$ and $\mathcal{S}_1$ is shown in the top box of Fig. 18, with L_1 as in the proof of Theorem 2 (that is, L_1 is a PT function that gives the length of the pad masking the garbled function F). The analysis is analogous to the proof of Theorem 4.

Now for each $\mathrm{xxx} \in \{\mathrm{prv}, \mathrm{obv}\}$, if $\mathcal{G}$ is xxx secure then $\mathcal{G}'$ is xxx1 secure. In scheme $\mathcal{G}'$, the decoding function is d_1, and the garble input is (X, R), whereas in scheme $\mathcal{G}_1 = \textsf{rom-all-to-all1}[\mathcal{G}]$, the former is (d_1, K), and the latter is $(X, R, \textsc{Hash}(k, K \parallel R))$, with $K \twoheadleftarrow \{0,1\}^k$. Scheme $\mathcal{G}_1$ thus can be re-interpreted as scheme $\mathcal{G}'$, with a different encoding of the garbled input and decoding function. Hence $\mathcal{G}_1$ is also xxx1 secure.

For part (2), given any PT adversary $\mathcal{A}_1$ against the aut1 security of $\mathcal{G}_1$, we build a PT adversary $\mathcal{A}$ against the aut security of $\mathcal{G}$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}_1}^{\mathrm{aut1}}(\mathcal{A}_1, k) \leq \mathbf{Adv}_{\mathcal{G}}^{\mathrm{aut}}(\mathcal{A}, k) + (2Q(k) + 1)/2^k,$$

```
adversary $\mathcal{A}_1(1^k)$
$b' \leftarrow \mathcal{A}_2^{\textsc{GarbleSim},\textsc{InputSim},\textsc{HashSim}}(1^k)$
return $b'$

proc GarbleSim($f$)
$n \leftarrow f.n,\ \ j \leftarrow 0$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f))$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
$F \leftarrow \textsc{Garble}(f)$
return $F$

proc InputSim($i, c$)
$x_i \leftarrow c,\ \ j \leftarrow j+1,\ \ S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ then
    $x \leftarrow x_1 \cdots x_n,\ \ (X_1, \ldots, X_n) \leftarrow \textsc{Input}(x)$
    $S \leftarrow S_1 \oplus \cdots \oplus S_n$
    for $t \in \{1, \ldots, n\}$ do $\mathsf{H}[\ell_t, 1 \| t \| S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return $T_i$

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

```
simulator $\mathcal{S}_2(1^k, \phi, 0)$
$(\ell, \ell_1, \ldots, \ell_n) \leftarrow L(1^k, \Phi(f)),\ \ F \leftarrow \mathcal{S}_1(1^k, \phi, 0)$
for $i \in \{1, \ldots, n\}$ do $U_i \twoheadleftarrow \{0,1\}^{\ell_i}$
return $F$

simulator $\mathcal{S}_2(i, j)$
$S_i \twoheadleftarrow \{0,1\}^k$
if $j = n$ then
    $(X_1, \ldots, X_n) \leftarrow \mathcal{S}_1(1)$
    $S \leftarrow S_1 \oplus \cdots \oplus S_n$
    for $t \in \{1, \ldots, n\}$ do $\mathsf{H}[\ell_t, 1 \| t \| S] \leftarrow X_t \oplus U_t$
$T_i \leftarrow (U_i, S_i)$
return $T_i$

simulator $\mathcal{S}_2(r, w, \texttt{ro})$
if $\mathsf{H}[r, w] = \bot$ then $\mathsf{H}[r, w] \twoheadleftarrow \{0,1\}^r$
return $\mathsf{H}[r, w]$
```

Fig. 19. Constructed adversary and simulator used in part (1) of the proof of Theorem 11. Except for the last query, return random tokens. For the last query, produce the real tokens, choose a random seed $S \twoheadleftarrow \{0,1\}^k$, and program the RO so that the shares unmask the real tokens,

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_1$ or by the scheme algorithms) in the execution of game $\text{Aut1}_{\mathcal{G}_1}$ with $\mathcal{A}_1$ on input 1^k. The code of the adversary $\mathcal{A}$ is given in the bottom of Fig. 18. Let Bad be the even that $\mathcal{A}$ can query (ℓ, w) to the random oracle such that either (i) R is a suffix of w, and this query is made prior to receiving R from the garbled input, or (ii) K is a prefix of w. Then $\Pr[\text{Bad}] \leq 2Q(k)/2^k$. Suppose than Bad does not happen. Let $Y_1 = (Y, R', \text{tag})$ be the output of $\mathcal{A}_1$. If $R' \neq R$ then the chance that tag $= \textsc{HashSim}(k, K \parallel R')$ is at most 2^{-k}. If $R = R'$ then $\mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k) = \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k)$. Hence, totally,

$$\mathbf{Adv}_{\mathcal{G}_1}^{\text{aut1}}(\mathcal{A}_1, k) \leq \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k) + \Pr[\text{Bad}] + 2^{-k} \leq \mathbf{Adv}_{\mathcal{G}}^{\text{aut}}(\mathcal{A}, k) + (2Q(k)+1)/2^k \ .$$

D.8 Proof of Theorem 11

Let $\mathcal{G}_2 = \textsf{rom-all1-to-all2}[\mathcal{G}_1]$. For part (1), it suffices to give the proof for the obliviousness case. The proof is similar to that of Theorem 5. Given any PT adversary $\mathcal{A}_2$ against the obv2 security of $\mathcal{G}_2$ we build a PT adversary $\mathcal{A}_1$ against the obv1 security of $\mathcal{G}_1$. Now the assumption of obv1 security yields a PT simulator $\mathcal{S}_1$ for $\mathcal{A}_1$ such that $\mathbf{Adv}_{\mathcal{G}_1}^{\text{obv1}, \Phi, \mathcal{S}_1}(\mathcal{A}_1, \cdot)$ is negligible. We then build from $\mathcal{S}_1$ a PT simulator $\mathcal{S}_2$ such that for all $k \in \mathbb{N}$ we have

$$\mathbf{Adv}_{\mathcal{G}_2}^{\text{obv2}, \Phi, \mathcal{S}_2}(\mathcal{A}_2, k) \leq \mathbf{Adv}_{\mathcal{G}_1}^{\text{obv1}, \Phi, \mathcal{S}_1}(\mathcal{A}_1, k) + Q(k)/2^k,$$

where Q is a polynomial such that $Q(k)$ bounds the total number of queries to Hash (made either directly by $\mathcal{A}_2$ or by the scheme algorithms) in the execution of game $\text{Obv2}_{\mathcal{G}_2, \Phi, \mathcal{S}_2}$ with $\mathcal{A}_2$ on input 1^k. The code of $\mathcal{A}_1$ and $\mathcal{S}_2$ is given in the bottom box of Fig. 18, with L as in the proof of Theorem 5 (that is, L is a PT function that gives the length of the pads masking the tokens).

For part (2), we reuse the procedures GARBLESIM and INPUTSIM in part (1). Let $\mathcal{A}_2$ attack the aut2 security of $\mathcal{G}_2$. Adversary $\mathcal{A}_1(1^k)$ runs $\mathcal{A}_2(1^k)$, simulating the latter's GARBLE and INPUT oracles via procedures GARBLESIM and INPUTSIM respectively. When $\mathcal{A}_2$ outputs Y, adversary $\mathcal{A}_1$ also outputs Y. Let Bad be the event that $\mathcal{A}_2$ queries (r, w) to the random oracle such that S is the suffix of w, prior to receiving the entire garbled input, where S is the seed generating the pads. Then $\Pr[\text{Bad}] \le Q(k)/2^k$, where Q is a polynomial such that $Q(k)$ bounds the total number of queries to HASH (made either directly by $\mathcal{A}_2$ or by the scheme algorithms) in the execution of game $\text{Aut2}_{\mathcal{G}_2}$ with $\mathcal{A}_2$ on input 1^k. Let X and T be the garbled input given to $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. If Bad does not happen then

$$\begin{aligned}\mathbf{Adv}^{\text{aut2}}_{\mathcal{G}_2}(\mathcal{A}_2, k) &= \Pr[Y \ne \mathsf{Ev}_2(F, T) \wedge \mathsf{De}_1(d, Y) \ne \bot] \\ &= \Pr[Y \ne \mathsf{Ev}_1(F, X) \wedge \mathsf{De}_1(d, Y) \ne \bot] = \mathbf{Adv}^{\text{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k),\end{aligned}$$

the second equality holds because $\mathsf{Ev}_2(F, T) = \mathsf{Ev}_1(F, X)$. Hence totally,

$$\mathbf{Adv}^{\text{aut2}}_{\mathcal{G}_2}(\mathcal{A}_2, k) \le \mathbf{Adv}^{\text{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) + \Pr[\text{Bad}] \le \mathbf{Adv}^{\text{aut1}}_{\mathcal{G}_1}(\mathcal{A}_1, k) + Q(k)/2^k \enspace .$$

D.9 Proof of Theorem 7

Let $\mathcal{S}'$ be a simulator to which $\mathcal{G}$ is prv2 secure. Fix an adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$. Consider the following simulator $\mathcal{S}$. On input 1^k and $\phi = \Phi(f)$, it gets $(F, d) \leftarrow \mathcal{S}'(1^k, \phi, 0)$, initializes $Q = \emptyset$ and $\tau = \bot$, and then runs $\mathcal{A}(1^k, (F, d))$. Let $n = f.n$. Whenever $\mathcal{A}$ queries (i, b), the simulator $\mathcal{S}$ proceeds as follows:

```
**if** $i \notin \{1, \ldots, n\} \backslash Q$ **then return** $\bot$
$Q \leftarrow Q \cup \{i\},\ \ x_i \leftarrow b$
**if** $|Q| = n$ **then** $x \leftarrow x_1 \cdots x_n,\ \ \tau \leftarrow y \leftarrow \mathrm{OTP}_f(x)$
$X_i \leftarrow \mathcal{S}'(\tau, i, |Q|)$
```

and then returns X_i to $\mathcal{A}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. Consider the following adversary $\mathcal{B}(1^k)$ attacking $\mathcal{G}$. It runs $\mathcal{D}(1^k)$. When the latter queries f, the former queries f to its oracle GARBLE to get (F, d). It then runs $\mathcal{A}(1^k, (F, d))$. For each query (i, b) of $\mathcal{A}$, the adversary $\mathcal{B}$ queries (i, b) to its oracle INPUT, and gives the answer to $\mathcal{A}$. Finally, $\mathcal{B}$ returns $\mathcal{A}$'s output to $\mathcal{D}$. If the challenge bit c of game $\text{Prv2}_{\mathcal{G},\Phi,\mathcal{S}'}$ is 1 then $\mathcal{B}$ is giving $\mathcal{D}$ the distribution $\mathsf{Real}_{\text{OTC}[\mathcal{G}],\mathcal{A},f}(k)$. Otherwise, if $c = 0$ then $\mathcal{B}$ is giving $\mathcal{D}$ the distribution $\mathsf{Fake}_{\text{OTC}[\mathcal{G}],\Phi,\mathcal{S},f}(k)$. Hence $\mathbf{Adv}^{\text{prv2},\Phi,\mathcal{S}'}_{\mathcal{G}}(\mathcal{B}, k) = \mathbf{Adv}^{\text{otc}}_{\text{OTC}[\mathcal{G}],\Phi,\mathcal{A},\mathcal{S},\mathcal{D}}(k)$.

D.10 Proof of Theorem 12

Let $\mathcal{A}_{\text{os}}$ be a PT one-time adversary attacking the verifiability of $\Pi[\mathcal{G}]$. We construct another PT adversary $\mathcal{A}_{\text{gs}}$ such that $\mathbf{Adv}^{\text{osvf}}_{\Pi[\mathcal{G}]}(\mathcal{A}_{\text{os}}, k) \le \mathbf{Adv}^{\text{aut1}}_{\mathcal{G}}(\mathcal{A}_{\text{gs}}, k)$ for all $k \in \mathbb{N}$, which proves the first claim in the theorem. Adversary $\mathcal{A}_{\text{gs}}(1^k)$ runs $\mathcal{A}_{\text{os}}(1^k)$, answering the GETPK query via GARBLE and the (single) INPUT query via INPUT. When $\mathcal{A}_{\text{os}}$ halts with output Y, j, adversary $\mathcal{A}_{\text{gs}}$ outputs Y.

Let $\mathcal{B}_{\text{os}}$ be a PT one-time adversary attacking the privacy of $\Pi[\mathcal{G}]$. We construct another PT adversary $\mathcal{B}_{\text{gs}}$ as follows. Adversary $\mathcal{B}_{\text{gs}}(1^k)$ runs $\mathcal{B}_{\text{os}}(1^k)$, answering the GETPK query via GARBLE and the (single) Input query via Input. When $\mathcal{B}_{\rm os}$ halts with output c', adversary $\mathcal{B}_{\rm gs}$ outputs c'. By the assumption that $\mathcal{G} \in \mathsf{GS}({\rm obv1}, \Phi)$ there is PT simulator $\mathcal{S}_{\rm gs}$ such that $\mathbf{Adv}_{\mathcal{G}}^{{\rm obv1}, \Phi, \mathcal{S}_{\rm gs}}(\mathcal{B}_{\rm gs}, \cdot)$ is negligible. Let $\mathcal{S}_{\rm os} \equiv \mathcal{S}_{\rm gs}$. Then $\mathbf{Adv}_{\Pi[\mathcal{G}]}^{{\rm ospr}, \Phi, \mathcal{S}_{\rm os}}(\mathcal{B}_{\rm os}, k) \leq \mathbf{Adv}_{\mathcal{G}}^{{\rm obv1}, \Phi, \mathcal{S}_{\rm gs}}(\mathcal{B}_{\rm gs}, k)$ for all $k \in \mathbb{N}$, which proves the second claim in the theorem.