Free IF: How to Omit Inactive Branches and Implement $\mathcal{S}$-Universal Garbled Circuit (Almost) for Free

Vladimir Kolesnikov(✉)

School of Computer Science, Georgia Institute of Technology, Atlanta, USA
kolesnikov@gatech.edu

Abstract. Two-party Secure Function Evaluation (SFE) allows two parties to evaluate a function known to both parties on their private inputs. In some settings, the input of one of the parties *is* the definition of the computed function, and requires protection as well. The standard solution for SFE of private functions (PF-SFE) is to rely on Universal Circuits (UC), which can be programmed to implement *any* circuit of size s. Recent UC optimizations report the cost of UC for s-gate Boolean circuits is $\approx 5s \log s$.

Instead, we consider garbling that allows evaluating one of a given *set* $\mathcal{S}$ of circuits. We show how to evaluate one of the circuits in $\mathcal{S}$ at the communication cost comparable to that of evaluating the largest circuit in $\mathcal{S}$. In other words, we show how to *omit* generating and sending inactive GC branches. Our main insight is that a garbled circuit is just a collection of garbled tables, and as such can be reused to emulate the throw-away computation of an inactive execution branch without revealing to the Evaluator whether it evaluates active or inactive branch.

This cannot be proven within the standard BHR garbled circuits framework because the function description is inseparable from the garbling by definition. We carefully extend BHR in a general way, introducing *topology-decoupling circuit garbling*. We preserve all existing constructions and proofs of the BHR framework, while allowing this and other future constructions which may treat garbled tables separately from function description.

Our construction is presented in the semi-honest model.

1 Introduction

Using circuit representation of the evaluated function brings a significant disadvantage in the SFE world. Indeed, in contrast with the Random-Access Machine (RAM) model, circuits introduce expensive, often crippling, redundancies to SFE by requiring to generate, send and evaluate *all conditional branches*, even if one of the players knows the branch taken. Circuits require unrolling loops, incur linear costs when accessing an array element, etc. Yet, circuit-based SFE is currently the highest-performing technique in most settings, due to extremely high efficiency of the private evaluation of circuit gates.

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11274, pp. 34–58, 2018.
https://doi.org/10.1007/978-3-030-03332-3_2

Addressing the limitations of the circuit-based representation has focused mainly on improving random access to memory. A celebrated line of work on Oblivious RAM (ORAM), started by [GO96], resulted in ORAM being a standard ingredient in MPC.

Our work. We address the need to pay for inactive throw-away conditional branches. We propose an *extremely simple* technique, Free IF, to **fully eliminate inactive GC branches** in scenarios where one of the players knows the executed branch. This is a natural scenario frequently occurring in practice, as we will argue next.

We extend the BHR framework [BHR12] to introduce *topology-decoupling circuit garbling* and present the construction in general terms. The extended BHR framework, which treats garbled circuits as strings, is a contribution of independent interest.

To our knowledge, this is the first such circuit-based technique. We discuss previous work in Sect. 1.2; most well-known prior work on circuit size reduction is generic universal circuit (UC) constructions.

1.1 Motivating Applications

We list several practical applications where our approach can be applied.

Evaluating one of several policy options. In Blind Seer [PKV+14, FVK+15], a GC-based private database (DB) system, private DB search is achieved by two players jointly securely evaluating the query match function on the search tree of the data. Blind Seer does not fully protect query privacy: it leaks the query circuit topology as the full universal circuit is not practical, as admitted by the authors. Applying our solution to that work would hide this important information, *at (almost) no extra cost.* Indeed, say, by policy the DB client is allowed to execute one of several (say, 50) types of queries. The privately executed SQL query can then be a represented as a `switch` of the number of clauses selected by the querier, each corresponding to an allowed query type. With our technique, only a GC corresponding to a single branch will need to be sent instead of the 50 required today. Most of the cost of the Blind Seer DB system is in running SFE of the query match function at a large scale, so improvement to the query circuit will directly translate to overall improvement. We note that the core of the Blind Seer system is in the semi-honest model, but a malicious client is considered in [FVK+15].

Our work can be viewed as secure evaluation of a circuit universal for a set of functions $\mathcal{S} = \{\mathcal{C}_1, ..., \mathcal{C}_k\}$ ($\mathcal{S}$-universal circuit, or $\mathcal{S}$-UC) at the cost similar to that of a single function. The next motivating example provides another illustration of how our work may improve applications where we want to evaluate and hide which function/query was chosen by a player (say, which one of several functions allowed by policy or known because of auxiliary information).

SFE of semi-private functions (SPF-SFE) (see additional discussion in Sect. 1.2) is a notion introduced in [PSS09], bridging the gap between expensive

private function SFE (PF-SFE) based on Universal Circuit [Val76, KS08b, KS16, LMS16], and regular SFE (via GC) that does not hide the evaluated function. SPF-SFE partially hides the evaluated function; namely, given a set of functions, the evaluator will not learn which specific function was evaluated. (The GC Generator does know the evaluated function.) Indeed, often only specific subroutines are sensitive, and it is they that might be sufficiently protected by $\mathcal{S}$-universal circuit for an appropriate set of circuits $\mathcal{S}$. [PSS09] presents a convincing example of privacy-preserving credit checking, where the check function itself needs to be protected, and shows that using $\mathcal{S}$-universal circuits as building blocks is an effective way of approaching this. Further, [PSS09] builds a compiler which assembles GC from the $\mathcal{S}$-universal building blocks (which they call PPB, Privately Programmable Blocks). While [PSS09] provides only a few very simple hand-designed blocks (see our discussion in Sect. 1.2), our work can be viewed as an efficient general way of constructing such blocks.

CPU/ALU emulation. Extending the idea of SPF-SFE, one can imagine a general approach where the players privately emulate a CPU evaluating a *fixed* sequence of complex instructions from a fixed instruction set (instruction choice implemented as a GC `switch`). Additionally, if desired, instructions' inputs can be protected by employing the selection blocks of [KS08b]. Such an approach can be built within a suitable framework (e.g., that of [PSS09]) from $\mathcal{S}$-universal circuits provided by this work. We note that circuit design and optimization is tedious, and not likely to be performed by hand except for very simple instances, such as those considered in [PSS09]. Instead, our approach will result in immediate performance improvement, reducing the cost of the ALU step implementation by a large factor.

For example, in a recent work [WGMK16], a secure and practically efficient MIPS ALU is proposed, where the ALU is implemented as a `switch` over 37 currently supported ALU instructions evaluated on ORAM-stored data. TinyGarble [SHS+15] also design and realize a garbled processor, using the MIPS I instruction set, for private function evaluation. Our constructions would work with [WGMK16, SHS+15] in a drop-in replacement manner, for implementing straight-line functions known to one party. The ALU step will be correspondingly reduced from containing implementations of *all* ALU instructions per step (37 in [WGMK16], of which the output of 36 of them is discarded), to a single instruction with our approach!

The client-server setting. Our approach is particularly attractive in the client-server setting. Indeed, the cost of the GC generator for the $\mathcal{S}$-UC of n circuits, while proportional to n, only involves a simple operation per circuit of $\mathcal{S}$. The bulk of the cost of the GC generator is in garbling and sending (only) the active branch. Because of this, the set $\mathcal{S}$ of the circuits can be very large and still scale well allowing the server being able to service many clients. This is because (essentially) the sole cost of adding more circuits to $\mathcal{S}$ is *the evaluator* having to evaluate each circuit in $\mathcal{S}$. This allows for a variety of trade-offs between efficiency and the level of hiding of the evaluated function.

1.2 Background and Related Work

Garbled Circuit, OT and Universal Circuit. Significant part of SFE research focuses on minimizing the size of the basic GC of Yao [Yao86,LP09], such as garbled row reduction techniques Free-XOR [KS08a] and its enhancements FleXOR [KMR14] and half-gates [ZRE15]. In contrast, in this work, we eliminate the need for evaluation (i.e. sending) of all but one subcircuits in a `switch`.

Asymptotically, Valiant's Universal Circuit [Val76,LMS16,KS16,GKS17] is the optimal underlying technique to fully protect the evaluated function in MPC. Respectively, for sub-circuits of size n, the size of the universal circuit generated by [Val76,KS08b] is $\approx 19n \log n$, and $\approx 1.5n \log^2 n + 2.5n \log n$. Recent works [LMS16,KS16,GKS17] polish and implement Valiant's construction. They report a precise estimate of the cost (in universal gates) of Valiant's UC of $\approx 5n \log n$. We note that UC-based constructions cannot take advantage of Free-XOR (other than gates on permutation subcircuits), since Free-XOR of course identifies positions of XOR gates. Thus, the classical universal circuit approach becomes competitive for a number of clauses far larger than a typical `switch`.

Another technique for Private Function Evaluation (PFE) was proposed by Mohassel and Sadeghian [MS13]. They propose an alternative (to the universal circuit) framework of SFE of a function whose definition is private to one of the players. Their approach is to map each gate outputs to next gate outputs by considering a mapping from all circuit inputs to all outputs, and evaluate it obliviously. For GC, they achieve a factor 2 improvement as compared to Valiant [Val76] and a factor 3–6 improvement as compared to Kolesnikov and Schneider [KS08b]. Similarly to [Val76,KS08b], [MS13] will not be cost-effective for a small number of clauses.

We also mention, but do not discuss in detail, that hardware design considers circuit minimization problems as well. However, their typical goal is to minimize chip area while allowing multiple executions of the same (sub)circuit. Current state-of-the-art in applying to MPC the powerful tool chains from hardware design is producing 10–20% circuit (garble table) reduction [SHS+15,DDK+15, DKS+17], while our approach will achieve large factor performance improvement for the setting it can operate in).

Semi-private function SFE (SPF-SFE) [PSS09]. As discussed above, SPF-SFE is a convincing trade-off between efficiency and the privacy of the evaluated function. Our work on construction of container circuits corresponds to that of privately programmable blocks (PPB) of [PSS09], which were hand-optimized in that work. In our view, the main contribution of [PSS09] is in identifying and motivating the problem of SPF-SFE and building a framework capable of integrating PPBs into a complete solutions. They provide a number of very simple (but nevertheless useful) PPBs, such as $\mathcal{S}_{COMP} = \{<, >, \leq, \geq, \neq\}$. Each of these PPB sets only consists of functions with already identical or near-identical topology; this is what enabled hand-optimization and optimal sizes of the containers. Other than the universal circuit PPB, no attempt was made to investigate construction PPBs of circuits of *a priori* differing topology.

In contrast, we can work with *any set* $\mathcal{S}$ of circuits for $\mathcal{S}$-universal circuit and achieve large factor performance improvement stemming from not having to transmit inactive branches.

Circuit overlay heuristic [KKW17]. Finally, a recent work of Kennedy et al. [KKW17] explored a *heuristic* approach to $\mathcal{S}$-UC circuit generation, based on alignment and overlay of underlying graphs. The authors were able to demonstrate significant reduction in the size of a circuit implementing a `switch` of 32 small circuits. Specifically, for their `switch` of 32 small chosen circuits of total size of $\approx 20,000$ gates, they were able to achieve the $\mathcal{S}$-UC of size $\approx 3,000$ gates, achieving $\approx 6\times$ circuit size reduction.

In contrast, our approach is *much* simpler and is readily implementable. It is *not* a heuristic, and has clean and understandable performance, which will nearly always beat [KKW17] (often by a significant factor!) in our setting where the GC generator knows the evaluated function. This is because of the following. In this case both our and [KKW17] cost consists of GC generator Gen generating and sending a single GC. However, in our case, this circuit size is equal to $\max|C_i|, C_i \in \mathcal{S}$, while the [KKW17] circuit size is $|C_0|$, where C_0 is the circuit universal for all $C_i \in \mathcal{S}$. Clearly, $|C_0| \geq \max|C_i|$, but it is difficult to give a precise comparison since [KKW17] is a heuristic. As reported in [KKW16], the full version of [KKW17], while overlay algorithm performed well on certain pairs (groups) of circuits, it did not do well on others. For example, expansion metric for circuits 29 and 30 (computing functions $B \cdot A + 555$ and $B^2 + A^2 > 1$ respectively on 32 bit values) is reported to be 1.00 (Table 3 in [KKW16]), which means that heuristic did not improve on simple circuit concatenation. In contract, our approach will immediately work for these circuits.

The BHR framework [BHR12]. We present our generic protocol in the terminology of BHR, which we extend to allow formal discussion of our work. We explain the very useful BHR framework at length in Sect. 5.

2 Our Contributions

We present Free `IF`, an extremely simple (and hence easy-to-implement and to adopt) method of eliminating the generation and transmission of *all* inactive branches in a GC computation, when branch is selected by the GC generator. An additional OT round, transferring secrets of size independent of the circuit sizes and concretely small, is required.

Our approach works with state-of-the-art garbling schemes, including half-gates [ZRE15].

We believe that our main idea — viewing GC as a collection of garbled tables and separating the circuit topology from GC thus hiding the computed function — will have other exciting applications, such as improved GC constructions.

Our result in very natural in retrospect; it is surprising it was not discovered earlier, given a substantial body of work on private function evaluation. One explanation is that we challenge "obvious facts" such as that GC is not reusable

or that "GC is a structure, not a string". Both are widely accepted and are at the core of very general BHR framework. Both are challenged in our approach.

As a contribution of independent interest, we carefully extend the BHR framework to support a separation of circuit topology from the cryptographic material (such as garbled tables), and to provide convenient formalization for manipulating output encodings at wire granularity.

3 Technical Overview of Our Approach

Recall, garbled circuit (GC) can be viewed simply as a collection of garbled (encrypted) gate tables. Specifically, it need not include the specification of the evaluation topology (i.e. wire connections among the gates). While topology is needed for the evaluation, it may be conveyed to the evaluator Ev separately from the garbled tables, or by implicit agreement among the participants Gen and Ev. Further, GC may, but need not, provide confirmation to Ev that the obtained garbled label is a valid label.

Let $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ be a set of Boolean circuits. We assume that all circuits consist of fan-in-2 gates and have the same number of inputs and outputs. Let the Generator Gen have $n_{\mathsf{in\text{-}x}}$ input bits, and the Evaluator Ev have $n_{\mathsf{in\text{-}y}}$ input bits (total $n_{\mathsf{in}} = n_{\mathsf{in\text{-}x}} + n_{\mathsf{in\text{-}y}}$). Without loss of generality, let players receive the same output consisting of n_{out} bits. This is a standard and natural setting for GC and universal circuits.

Recall, in our setting, (only) Gen knows which of the circuits in $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ is being evaluated. Let's imagine for now that all the circuits in $\mathcal{S}$ are of the same size s (i.e. consisting of the same total number of gates); we will later show this easily generalizes. We *do not* place any other restrictions on the topologies of the circuits. Suppose Gen wishes to evaluate its target circuit $\mathcal{C}_t \in \mathcal{S}$. Our underlying idea is to have Gen generate a *single* GC $\widehat{\mathcal{C}}$ implementing $\mathcal{C}_t$ and send it to Ev. Ev knows $\mathcal{S}$, but it will not know which of the circuits in $\mathcal{S}$ is the target circuit. Now, for each $\mathcal{C}_i \in \mathcal{S}$, Ev will interpret $\widehat{\mathcal{C}}$ as garbling of $\mathcal{C}_i$ and evaluate it as such, obtaining the garbled output. With a little care in GC design, it is possible to ensure that Ev will not be able to distinguish which of the circuits in $\mathcal{S}$ is the target circuit $\mathcal{C}_t$. We stress that the fact that circuts in $\mathcal{S}$ have varying topologies is not an issue since GCs only contain garbled tables which can be used with any topology.

The next step is for Ev to obliviously discard the wire labels which belong to non-target circuits and to propagate the (encrypted) output of the target circuit. To be more precise, Gen and Ev will run an *output selection* (OS) protocol. Ev will provide as input to the protocol all (active) output labels it obtained in evaluating $|\mathcal{S}|$ circuits, and Gen will provide the indices of the labels corresponding to the target circuit $\mathcal{C}_t$, as well as $\mathcal{C}_t$'s zero labels on output wires. The OS protocol will output (re-encoded) labels corresponding to the output of $\mathcal{C}_t$.

This step is efficiently implemented via GC. We further observe that providing full-length output labels (i.e of length of the computational security parameter κ) as input to OS is not needed; statistical security is sufficient and the labels

can be truncated to σ bits for OS input, improving performance. Even further improvement is possible simply by having Gen ensure the labels on each output wire of $\mathcal{C}_t$ differ in the last bit. Then Ev and Gen can submit only the last bits of the labels they obtain. Note that it does not affect correctness or security if the last bits on the active and inactive wires of circuits other than $\mathcal{C}_t$ are the same, since selection is done based on the index t provided by Gen, and further Ev will not obtain both labels of any wire (and hence won't detect the mismatch between $\widehat{\mathcal{C}}$ and the interpreted evaluation of $\mathcal{C}_i$).

Free XOR and half-gates. Our construction works with the Free XOR garbling [KS08a].

Using half-gates [ZRE15] also works. Intuitively, this is because its garbled tables also look like random strings. We show this in Sects. 6 and 6.1. We note that using half-gates is concretely efficient, since the LSB of labels `true` and `false` is different and hence the garbled output can be cheaply fed into output selection protocol.

Addressing different circuit sizes in $\mathcal{S}$. It is easy to see that our approach does not limit us to considering $\mathcal{S}$ consisting only of the circuits of the same size. Indeed, let $s_{\max} = \max_i |\mathcal{C}_i|$ be the maximum circuit size in $\mathcal{S}$. It is sufficient[1] for Gen to garble the target circuit $\widehat{\mathcal{C}}_t$ *and pad it with randomly generated garbled tables* to obtain $\widehat{\mathcal{C}}'_t$, so as the total number of garbled tables in the produced circuit $\widehat{\mathcal{C}}'_t$ is equal to $s_{\max}$. Then, a simple convention can be easily designed to allow Ev to use only the garbled tables of needed in evaluating each circuit $\mathcal{C}_i$ by appropriately interpreting $\widehat{\mathcal{C}}'_t$.

On the cost of SFE and OT rounds. Our $\mathcal{S}$-UC GC protocol adds a round of communication for each switch statement. We argue that the associated latency cost is negligible in many practical scenarios. This is because often the latency-related idling will be productively used for computation and communication in the same or another SFE instance. This is the case, e.g., in larger-scale SFE deployments, where many instances will be run in parallel, and where SFE throughput is a far more important parameter than latency.

Composing our protocol with GC. We note that we additionally design a secret-shared-output functionality, where the output of the computation is not reconstructed, but remains shared GC-style. Hence, it can be privately plugged into another GC.

Nesting `switch` *clauses.* Our protocol naturally works for the nested clauses. One way to implement a nested clause is to bring all choice variables to the same level, placing us in the non-nested setting.

Figure 1 illustrates how a nested clause (left) can be rewritten to a single-level branching (right). Again, we note that Gen must know the selection choices.

[1] This holds for main schemes, such as classical Yao, Free-XOR and half-gates, as we show in Sect. 6.1. It is possible to craft garbling schemes where this specific technique won't work. See Sect. 5.3 for a formal discussion.

• if a = 1 then
  • if b = 1 then run F_{11}
  • if b = 2 then run F_{12}
• if a = 2 then
  • if b = 1 then run F_{21}
  • if b = 2 then run F_{22}
	• if a = 1 and b = 1 then run F_{11}
	• if a = 1 and b = 2 then run F_{12}
	• if a = 2 and b = 1 then run F_{21}
	• if a = 2 and b = 2 then run F_{21}

Fig. 1. `switch` nesting rewriting. Left: nested. Right: flat.

We note that this nesting management results in no additional communication rounds due to nesting. In most cases, the nesting would not be deep/wide enough to overwhelm the computational resource. Indeed, the computation cost would be less than implementing the same circuit using standard Yao GC.

To illustrate the costs, consider the above example and let's suppose all functions are of the same size $s_{\max}$, and Gen's choice variables are $a = 2$ and $b = 1$. Then our protocol will require Gen to generate and send a single GC $\widehat{\mathcal{C}}$ implementing F_{21}, and Ev to evaluate the received $\widehat{\mathcal{C}}$ four times. Standard GC will require Gen to generate and send four GCs, and Ev to evaluate four GCs.

3.1 Extending the BHR Framework: Decoupling the Topology

An important conceptual contribution of this work is the departure from thinking of garbled circuits as monolithic objects, but rather, emphasizing that they are strings representing (separately) the computed function and the cryptographic material, such as garbled tables. We formalize this approach by extending the BHR framework to support this vision. We are able to change some of the most fundamental concepts of the framework while preserving it completely and not requiring redefining any of its functions. This allows to reuse all existing body of work in the popular and very useful BHR framework.

Specifically, our main change to BHR is a restriction that the garbled circuit F must consist of two components, the function topology T and cryptographic material E. This adjustment does not affect any of the existing BHR functions, constructions and proofs, but allows us to introduce a new security property related to obliviousness, which formalizes indistinguishability of GC evaluation under different topologies.

We state our main result, Free IF, in the new extended BHR framework.

3.2 Outline of the Presentation

We already described at a high level the technical details of our contribution in Sect. 3. Next, we introduce preliminary notation and definitions. In Sect. 5, we review the relevant aspects of the BHR framework and introduce its extension that allows to reason about circuit garblings separately from the function encoding. We present our GC construction in the above extended framework, prove security and formally discuss suitable garbling schemes, including half-gates, in Sect. 6. Finally, in Sect. 7, we discuss the performance of our improvement.

4 Preliminaries

4.1 Notation

Throughout the paper we use the following notation: the computational and statistical security parameters are denoted by κ and σ, respectively. We will denote circuits by $\mathcal{C}$ and garbled circuits by $\widehat{\mathcal{C}}$. We denote a circuit's gate by $\mathcal{G}_i$ and a garbled gate by $\widehat{G}_i$. We denote by $\mathcal{S}$ the set $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ of circuits with respect to which we design our universal circuit. We denote size of $\mathcal{S}$ by $s = |\mathcal{S}|$. NOT gates are implemented in the standard way by switching the wire label semantics and we don't discuss them further. We count the number of non-XOR gates in a circuit as its size $s_{\mathcal{C}_i} = |\mathcal{C}_i|$. We denote the maximal circuit size in $\mathcal{S}$ by $s_{\max} = \max_{\mathcal{C}_i \in \mathcal{S}} |\mathcal{C}_i|$. We denote by $s_{\mathcal{C}_i}^{\mathsf{Total}} = |\mathcal{C}_i|^{\mathsf{Total}}$ the size of the circuit $\mathcal{C}_i$, *including/counting XOR gates*. We use the notation $\stackrel{c}{=}$ to denote computational indistinguishability of ensembles of random variables.

4.2 Defining $\mathcal{S}$-Universal GC

We introduce security definitions with which we operate in this work. We are interested in efficient $\mathcal{S}$-Universal GC evaluation. In Fig. 2 we formalize the functionality $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}}$ which will serve as the basic definition.

Intuitively, our goal is simple: we wish to evaluate a function chosen by one of the players P_1 among the known set of functions $\mathcal{S}$.

We additionally define a more convenient functionality $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}\text{-}\mathsf{s}}$ for shared-output $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}}$. $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}\text{-}\mathsf{s}}$ requires that the players don't get the output of the function $\mathcal{C}_t$ directly, but rather a GC-style secret sharing of the computed value. The functionality $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}\text{-}\mathsf{s}}$ is presented in Fig. 3. We will present our construction for the simpler $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}}$ functionality; extension to the more convenient $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC}\text{-}\mathsf{s}}$ functionality is simple, and we briefly discuss it in Sect. 6.

5 Extending the BHR Framework

It is beneficial to present the work in the terminology of garbling schemes [BHR12], introduced by Bellare, Hoang and Rogaway (BHR). In our abstraction approach, we aim to find a balance between generality and simplicity, while maximizing the reuse of the thoughtfully designed BHR framework.

We start by reminding the reader of the relevant details of the BHR framework.

5.1 BHR Garbling Schemes

Bellare, Hoang, and Rogaway [BHR12] introduce the notion of a garbling scheme as a cryptographic primitive. We refer the reader to their work for a complete treatment and give a brief summary of relevant aspects here. We note that their definitions apply to any kind of garbling, such as decision trees, automata, etc.

PARAMETERS:

1. Let $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ be a set of Boolean circuits. All circuits $\mathcal{C}_i$ have the same number n_{in} of input wires (with the same number $n_{\mathsf{in\text{-}x}}$ and $n_{\mathsf{in\text{-}y}}$ wires provided by P_1 and P_2 respectively) and the same number n_{out} of output wires.
2. Two players P_1 and P_2 have inputs $x \in \{0,1\}^{n_{\mathsf{in\text{-}x}}}$ and $y \in \{0,1\}^{n_{\mathsf{in\text{-}y}}}$.
3. P_1 provides additional input: index $t \in [1..n]$ of the target circuit $\mathcal{C}_t$ to be evaluated.

FUNCTIONALITY:

Both players obtain the output $\mathcal{C}_t$ (x,y).

Fig. 2. $\mathcal{S}$-UC functionality $\mathcal{F}^{\mathcal{S}\text{-UC}}$

PARAMETERS:

1. Let $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ be a set of Boolean circuits. All circuits $\mathcal{C}_i$ have the same number n_{in} of input wires (with the same number $n_{\mathsf{in\text{-}x}}$ and $n_{\mathsf{in\text{-}y}}$ wires provided by P_1 and P_2 respectively) and the same number n_{out} of output wires.
2. Two players P_1 and P_2 have inputs $x \in \{0,1\}^{n_{\mathsf{in\text{-}x}}}$ and $y \in \{0,1\}^{n_{\mathsf{in\text{-}y}}}$.
3. P_1 provides additional input: index $t \in [1..n]$ of the target circuit $\mathcal{C}_t$ to be evaluated.

FUNCTIONALITY $\mathcal{F}^{\mathcal{S}\text{-UC-s}}$ PROCEEDS AS FOLLOWS:

1. Compute the n_{out}-bit plaintext output $\mathsf{out} = \mathcal{C}_t(x, y)$.
2. Randomly sample $2 \cdot n_{\mathsf{out}}$ bit strings of size κ each: $\ell_i^0, \ell_i^1 \in_R \{0,1\}^\kappa$, where $i \in [1..n_{\mathsf{out}}]$.
3. Send all labels ℓ_i^0, ℓ_i^1 to P_1.
4. For each $i \in [1..n_{\mathsf{out}}]$:
   - Designate label $\ell_i^{\mathsf{out}[i]}$ to be sent to P_2.
5. Send all designated labels to P_2.

Fig. 3. $\mathcal{S}$-UC-s functionality $\mathcal{F}^{\mathcal{S}\text{-UC-s}}$

We focus the notation on circuits and *circuit* garbling, which BHR consider as a special case, by requiring certain constraints on syntax and semantics of general object in their framework. A circuit garbling scheme consists of the following algorithms: Garble takes a circuit f as input and outputs (F, e, d) where F is a garbled circuit, e is encoding information, and d is decoding information. Encode takes an input x and encoding information e and outputs a garbled input X. Eval takes a garbled circuit F and garbled input X and outputs a garbled output Y. Finally, Decode takes a garbled output Y and decoding information d and outputs a plain circuit-output (or an error $\perp$).

Most relevant in our context are the prv.sim (privacy) and obv.sim (obliviousness) security definitions from [BHR12], which we state below. In the prv.sim and obv.sim games, the Initialize procedure chooses $\beta \leftarrow \{0,1\}$, and the Finalize(β') procedure returns $\beta \stackrel{?}{=} \beta'$. In both games, the adversary can make a single call to the Garble procedure, which is defined below. Additionally, the function Φ

denotes the information about the circuit that is allowed to be leaked by the garbling scheme; the function $\mathcal{S}$ is a simulator, and G denotes a garbling scheme.

$\mathsf{prv.sim}_{G,\Phi,\mathcal{S}}$:	$\mathsf{obv.sim}_{G,\Phi,\mathcal{S}}$:
$\underline{\mathrm{Garble}(f,x)}$:	$\underline{\mathrm{Garble}(f,x)}$:
if $\beta = 0$	if $\beta = 0$
$\quad (F,e,d) \leftarrow \mathsf{Garble}(1^\kappa, f)$	$\quad (F,e,d) \leftarrow \mathsf{Garble}(1^\kappa, f)$
$\quad X \leftarrow \mathsf{Encode}(e,x)$	$\quad X \leftarrow \mathsf{Encode}(e,x)$
else $(F,X,d) \leftarrow \mathcal{S}(1^\kappa, f(x), \Phi(f))$	else $(F,X) \leftarrow \mathcal{S}(1^\kappa, \Phi(f))$
return (F,X,d)	return (F,X)

We then define the advantage of the adversary in the security games:

$$\mathsf{Advtg}^{\mathsf{prv.sim}}_{G,\Phi,\mathcal{S}}(\mathsf{Adv},\kappa) := \left|\Pr[\mathsf{prv.sim}^{\mathsf{Adv}}_{G,\Phi,\mathcal{S}}(\kappa)=1] - \frac{1}{2}\right|;$$

$$\mathsf{Advtg}^{\mathsf{obv.sim}}_{G,\Phi,\mathcal{S}}(\mathsf{Adv},\kappa) := \left|\Pr[\mathsf{obv.sim}^{\mathsf{Adv}}_{G,\Phi,\mathcal{S}}(\kappa)=1] - \frac{1}{2}\right|.$$

We say that a garbling scheme satisfies privacy (resp. obliviousness) if for any polytime adversary Adv, the corresponding advantage Advtg is negligible. We omit restating here the remainder of the BHR framework, and refer the reader to the original work.

5.2 Intuition for Topology Decoupling and Composition

Ability to decouple the topology of evaluated GC, highlighted and used in this work, is related to the standard obliviousness property formalized by BHR. Intuitively, BHR obliviousness means that a party acquiring F and X, but not d, shouldn't learn anything about f, x, or y beyond that is explicitly allowed in the leakage function Φ. This is roughly the property we require as well, but with a different formalization, requiring careful handling.

The following are the technical issues that need to be addressed to enable discussion of our protocols in the (extended) BHR framework.

1. Let F be a string representing a garbled circuit. Firstly, we need to syntactically separate the function encoding (e.g., topology) T from the cryptographic material E included in F, such as garbled tables. That is, we wish to explicitly write $F = (T, E)$, thus enabling consideration of a GC (T', E). We note that in the BHR framework, the function description T is either implicit in Eval or is included in F in an unspecified manner.
2. Secondly, once this syntactic convention is adopted, we need to adjust the definitions to support evaluation under a "wrong" function encoding, and further, to require that Eval will not detect whether it operates with a "right" or "wrong" encoding.
3. Thirdly, the BHR framework naturally treats circuits as "the whole thing," and does not provide for a clean interface to discuss shared output (e.g.

undecoded wire labels which may later be used as encrypted input in another computation). In particular, the BHR decoding function Decode is required to output the correct plaintext value of the computation.

We now sketch a suitable formalization approach addressing the above issues for natural circuit representations. We take the BHR framework as the basis and adjust it as described next. The formal definitions are presented in Sect. 5.3.

We stress that for concreteness and convenience we next discuss specific ways to encode a circuit in GC. We note that that the definitions of Sect. 5.3 are more general, and may use arbitrary encodings.

Topological encoding in GC F. In BHR, $Y = \mathsf{Eval}(F, X)$ takes as input the garbled circuit F and garbled input X. The BHR framework does not discuss how garbled function F encodes information which allows Eval to proceed with the evaluation. In BHR, conventional Boolean circuits are viewed as a tuple (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. A (resp. B) is a function identifying a gate's first (resp. second) incoming wire, and G is a function identifying the gate function of the gate. BHR introduces the notation of *topological circuit* f^-, which is defined to be a conventional circuit f without the gate function component. That is, for a circuit $f = (n, m, q, A, B, G)$, the topological circuit f^- is defined as (n, m, q, A, B).

Let's consider the question of evaluating a GC, given a list of garbled tables. It is easy to see that f^- contains sufficient topological information to evaluate classical Yao's GC, assuming an implicit correspondence between gate id and the garbled table associated with it. Such a correspondence is typically implemented by enforcing a canonical ordering of the gates and garbled tables.

At the same time, f^- does not have sufficient information to evaluate circuits garbled with Free-XOR [KS08a]. This is because the ids of XOR gates are not included in f^- and cannot be inferred from a bare list of garbled tables. To generalize this, we will consider *type-topological* circuits:

Definition 1. *Following the BHR notation, we say that f^* is a* conventional type-topological *circuit, if $f^* = (n, m, q, A, B, G^*)$, where G^* specifies the gate type of each gate. We will often simply say* topological circuit *or* topology *instead of* type-topological circuit *when clear from the context.*

We stress that the topological circuit must provide sufficient information to the Eval function to enable GC evaluation.

A typical example of gate types referred by G^* is the set {XOR, non-XOR}. Topology f^* with G^* defined over this type set allows evaluation of Free-XOR and half-gate garbled circuits. We note that other natural definitions of circuit topology are possible. One example is to include pointers to the garbled tables implementing gates. We choose the representation of Definition 1 as a balance of generality and simplicity which may be convenient to use for known GC schemes.

Topology Decoupling and Evaluation with Different topology. We need to enable parsing and evaluation of GC F with varying topology. As outlined above, for this we need to syntactically decouple the (implicit and deterministically computed) GC topology from the (generated using randomness) cryptographic material, such as encrypted tables. Without loss of generality, we *require* parsing the garbled circuit F as $F = (T, E)$, where T is the type-topological circuit f^* and E is the cryptographic material, such as a set of encrypted tables. We set $f^* = (n, m, q, A, B, G^*)$. We thus require Garble to produce F in the above format, and $\mathsf{Eval}(F, X)$ to accept this format $F = (T, E)$ of garbled circuit.

By extracting the type-topology T out of the GC F we enable evaluating F with an arbitrary topology T'. As a security feature of a circuit garbling scheme, we will require that evaluator is unable to tell whether it is evaluating with the intended topology T or an arbitrary different one T'. This will be ensured by requiring that the ensembles $\{(T, E), X\}$ and $\{(T', E), X\}$ are indistinguishable, where X are the (encoded) input labels, T is the matching topology, and T' is any admissible topology. In Sect. 5.3, we define the intuitive notion of admissible topology (by considering classes of mutually admissible functions) and formalize the above indistinguishability property.

Again, we stress that different topology representations are possible. While the type-topology described above is convenient for known GC schemes, our definitions in Sect. 5.3 do not restrict to using above representation.

Shared Output and Composition in Garbled Circuits. As noted above, the BHR framework does not naturally handle in generality the composition of garbled functions. It has the concept of garbled output, which, together with the decoding information can be seen as the secret-sharing of the function output. However, this representation is too general, and as one consequence, does not support discussing manipulation of the undecoded or partial output and feeding such output into a subsequent execution. In [BHR12], the authors sometimes handle over-generality by parsing standard BHR objects in a certain way. For example, faced with the need to discuss circuit input labels and their use in OT, the authors simply say "parse $(X_1^0, X_1^1, ..., X_n^0, X_n^1) \leftarrow e$." This, of course, assumes a specific garbling scheme, and represents a trade off between simplicity, generality and formalism.

One way of formalizing the required secret-shared output bits is by introducing restrictions on the format of the decoding information Y[2].

It would be convenient to formalize this restriction *as an option* for circuit garbling scheme, by requiring that the garbled output $Y = (Y_1, ...Y_{n_{\mathsf{out}}})$ is a vector of garbled wire outputs and allows for syntactic access to any particular component. Similarly, we require that the decoding information $d = (d_1, ..., d_{n_{\mathsf{out}}})$

[2] Indeed, BHR allows arbitrary representation options, including exotic ones such as $Y = \mathrm{AES}_k(y)$ being an AES encryption of the multi-bit output y, and $d = k$ being the AES decryption key. Clearly such a representation, while secure, is inconvenient for revealing a partial output or providing the unencrypted output for further GC evaluation.

is a vector of output wire decodings, such that d_i allows to decode the garbled output Y_i. We will overload the standard BHR Decode function to take as input any subsets of garbled wire labels and corresponding decoding information.

5.3 Definition of Topology-Decoupling Circuit Garbling

We now formalize the intuition described in Sect. 5.2.

Recall, in the BHR framework [BHR12], the garbling scheme is a five-tuple of algorithms $\mathsf{GS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval})$. A BHR circuit garbling scheme $\mathsf{CGS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval})$ is a garbling scheme with certain natural syntactic restrictions. To reason about our protocol in generality, we introduce a further syntactic restriction on the BHR garbled circuits. Namely, syntactically, we will require the garbled circuit to be specified as $F = (T, E)$. Here T is a function encoding, such as the *conventional type-topologic circuit* (cf. Definition 1), and E is cryptographic material, such as garbled tables. We keep the syntax of all BHR functions $\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval}$.

In the following we use the standard BHR notation, and we only present notions and objects different from standard BHR.

Definition 2 (Circuit Garbling Scheme (CGS)). *We consider Circuit Garbling Scheme as defined by BHR, with the following difference:*

Garbled circuit. We require the garbled circuit $F = (T, E)$ to explicitly define the function encoding component T. T is implicit in and deterministically *obtained from the computed plaintext circuit f. We will often say* topology *instead of function encoding when clear from the context.*

Garble. *On input $(1^\kappa, f)$,* Garble *will output (F, e, d), where $F = (T, E)$. Here $T = T(f)$ is deterministic and hence can be computed by any party, including the GC evaluator.*

Eval. *The garbled evaluation function* Eval *takes garbled input X and $F = (T', E)$ as input.* Eval *outputs garbled output labels Y or a special failure symbol $\perp$.*

Reusing BHR machinery. With the above syntactic restriction, we are able to reuse the existing BHR garbling machinery in defining a generalization of BHR circuit garbling. Importantly, our notion is a *special case* of BHR garbling scheme, and thus we can keep the BHR function definitions and correctness and security requirements as is. This is because we (so far, with a single exception) restricted the syntax of the BHR notions. Our only generalization (allowing to evaluate under different topology), is not exercised in BHR definitions. Therefore, all BHR notation and definitions retain their meaning and are reused.

In other words, the BHR framework (which we retain in full as the foundation!) is sufficient to handle the case of evaluating a circuit with correct topology. We only need to define the behavior of circuit garbling in the generalized case of evaluation under different topology.

Specifically, we only need to define the security properties ensuring that evaluation under an admissible topology is indistinguishable from correct evaluation.

Topology-decoupling circuit garbling schemes. There are several approaches to defining indistinguishability of garbled circuits with matching ("right") and non-matching ("wrong") topology.

In one approach, we could require the Garble function to take an additional parameter s, specifying the size of the maximal circuit. Garble then would produce a garbled circuit implementing the given function f, but which has extra garbled tables, suitable for implementing any circuit of size up to s. Then we would require that the output (T, E) of such Garble function is such that ensembles (T, E) and (T', E) are indistinguishable. This general approach requires either overloading syntax of a standard BHR function, or introducing a new function. In turn, BHR definitions stipulating security guarantees of standard functions will need to be rephrased and stated second time for the new function.

Another approach could be to define a small-family garbling scheme w.r.t. a fixed set of circuits which we intend to use as conditional clauses, and then define generalized obliviousness. This is a natural approach, but it requires showing the security of the garbling scheme for each set of circuits that might be required.

Our approach. Instead, we will take a cleaner and more general definitional approach. We consider classes of mutually topologically admissible circuits. Such a class would be defined by a canonical (for the class) circuit f_{can}. (For example, in known schemes, such as Free-XOR and half-gates, the class of circuits will be defined by the maximal number s of non-XOR gates, and f^s_{can} could be a circuit consisting of s AND gates.) We stress that multiple circuits could be canonical for the same class C.

For convenience, we first introduce the following notation.

Definition 3 (Embedding of cryptographic material). *We denote by* Embed *the procedure of introducing the cryptographic material E of a function f into the cryptographic material E_{can} of the canonical circuit f_{can}. We will write* $\mathsf{Embed}(E, E_{\mathsf{can}})$ *to denote the output of this procedure.*

The idea of the Embed procedure and its use, formalized in the definitions next, is that the cryptographic material $\mathsf{Embed}(E, E_{\mathsf{can}})$ can be used to evaluate f, but is indistinguishable from embeddings of cryptographic material $\mathsf{Embed}(E', E_{\mathsf{can}})$ of any other function f', where f and f' belong to the same class of mutually admissible circuits. The indistinguishability must hold even if the encoded input is given. Embed will be defined as part of garbling scheme description.

We stress that circuit encoding details, such as wiring, number of inputs and outputs, etc., need not be explicitly specified and discussed in the definition. Instead, this is handled by considering a class C of circuits for which the garblings are compatible or mutually admissible (defined below).

We now formally define the indistinguishability requirement.

Definition 4 (Topology-decoupling circuit garbling). *Let* $\mathsf{CGS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval}, \mathsf{Embed})$ *be a circuit garbling scheme as discussed above (Definition 2), with the added* Embed *function. Let f_{can} be a circuit,*

$\mathsf{topo}_{\mathsf{CGS},C,f_{\mathsf{can}}}$:

Garble(f, f', x):
if $f \notin C$ or $f' \notin C$, abort.
$(F_{\mathsf{can}}, e_{\mathsf{can}}, d_{\mathsf{can}}) \leftarrow \mathsf{Garble}(1^\kappa, f_{\mathsf{can}})$, where $F_{\mathsf{can}} = (T_{\mathsf{can}}, E_{\mathsf{can}})$
$(F, e, d) \leftarrow \mathsf{Garble}(1^\kappa, f)$, where $F = (T, E)$
$(F', e', d') \leftarrow \mathsf{Garble}(1^\kappa, f')$, where $F' = (T', E')$
$X \leftarrow \mathsf{Encode}(e, x)$
if $\beta = 1$
    return $((T, \mathsf{Embed}(E, E_{\mathsf{can}})), X)$
else
    return $((T', \mathsf{Embed}(E', E_{\mathsf{can}})), X)$

Fig. 4. Game $\mathsf{topo}_{\mathsf{CGS},C,f_{\mathsf{can}}}$.

and let $C = Class(f_{\mathsf{can}})$ *be a set (or class) of circuits. Consider the distinguishing advantage of the adversary winning the game* topo *of Fig. 4 (cast in the BHR setup with Initialize() and Finalize() procedures as in BHR).*

We say that CGS *is* topology-decoupling *within* C *and that* f_{can} *is* canonical *for* C*, if for every polytime adversary* Adv*, the following is negligible:*

$$\mathsf{Advtg}^{\mathsf{topo},f_{\mathsf{can}}}_{\mathsf{CGS}}(\mathsf{Adv}, \kappa) = |Pr[\mathsf{topo}^{\mathsf{Adv}}_{\mathsf{CGS},C,f_{\mathsf{can}}}(\kappa) = 1] - \frac{1}{2}| \tag{1}$$

We note that the topo game of Definition 4 exactly corresponds to our proposed construction, where the circuit's cryptographic material is evaluated under different function encodings/topologies with the same garbled input.

Finally, we need to require correct evaluation of a circuit $(T, \mathsf{Embed}(E, E'))$, where T, E are matching, i.e. produced by $F \leftarrow \mathsf{Garble}(\kappa)$. Formally:

Definition 5 (Extended correctness). *Let* CGS *be a circuit garbling scheme (Definition 2). Let* $((T, E), e, d) = \mathsf{Garble}(1^\kappa, f)$ *and* $((T_{\mathsf{can}}, E_{\mathsf{can}}), e_{\mathsf{can}}, d_{\mathsf{can}}) = \mathsf{Garble}(1^\kappa, f_{\mathsf{can}})$*, where* f *belongs to a class defined by a canonical function* f_{can}*. We say that* CGS *has extended correctness, if it always holds that:* $\mathsf{Decode}(d, \mathsf{Eval}[(T, \mathsf{Embed}(E, E_{\mathsf{can}})), \mathsf{Encode}(e, x)]) = \mathsf{eval}(f, x)$.

We note that topology-decoupling does not imply extended correctness, since the experiment in the topology decoupling definition does not have access to output encoding.

Notation. Extended correctness is an obvious and default requirement. Therefore, for convenience of notation, we will say *topology-decoupling circuit garbling* to mean "topology-decoupling circuit garbling with extended correctness."

We will use topology-decoupling circuit garbling schemes. To use them in constructing $\mathcal{S}$-UC protocols, one will need to first design a topology-decoupling garbling scheme and then apply the generic construction presented in the next section. Designing a required garbling scheme can be simply done by starting

with an existing scheme, such as half-gates, and showing that it meets the additional requirements (Definitions 4 and 5) or adjust it so that it does. This approach relies on and reuses the existing body of work of proving security in the BHR framework.

5.4 Output Manipulation Extension

We introduce the notation for bitwise output manipulation with the goal of keeping the standard BHR notation intact, while at the same time allowing formalizations and use in generic protocols.

Definition 6 (Topology-decoupling circuit garbling with bitwise decoding). *Let* $\mathsf{CGS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval}, \mathsf{Embed})$ *be a topology-decoupling circuit garbling scheme.*

We say that a topology-decoupling circuit garbling scheme CGS supports bitwise decoding *if the following holds:*

1. *The garbled output* Y *is a vector of garbled wire outputs* $Y = (Y_1, ...Y_{n_{\mathsf{out}}})$ *and allows for syntactic access to any particular component by index* Y_i.
2. *The decoding information* d *is a vector of output wire decodings* $d = (d_1, ..., d_{n_{\mathsf{out}}})$ *and allows syntactic access to any particular component by index* d_i.
3. *Extended correctness of decoding holds per wire. That is, let* $((T, E), e, d) = \mathsf{Garble}(1^\kappa, f)$ *and* $((T_{\mathsf{can}}, E_{\mathsf{can}}), e_{\mathsf{can}}, d_{\mathsf{can}}) = \mathsf{Garble}(1^\kappa, f_{\mathsf{can}})$, *where* f *belongs to a class defined by a canonical function* f_{can}. *Let* $Y = \mathsf{Eval}[(T, \mathsf{Embed}(E, E_{\mathsf{can}})), \mathsf{Encode}(e, x)]$. *Then we require* $\mathsf{Decode}(d_i, Y_i) = \mathsf{eval}(f, x)|_i$.

A similar definition is easily constructed for standard BHR framework.

6 $\mathcal{S}$-UC Construction from Topology-Decoupling Circuit Garbling

We now show a generic $\mathcal{S}$-UC construction built from the generic notion of topology-decoupling circuit garbling schemes with extended correctness, introduced above. In this section, we exclusively work with such schemes. For convenience, we may refer to them in this section simply as circuit garbling, when clear from context.

Let $\mathsf{CGS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval}, \mathsf{Embed})$ be a circuit-garbling scheme with topology-decoupling garbling (Definition 4) and extended correctness (Definition 5). The construction of Fig. 7 is presented in the generic terms of such circuit garbling. In reviewing the construction, it may be instructive to think in terms of a specific garbling scheme, such as the half-gates scheme or classical Yao, and the class of admissible circuits as all circuits with up to s non-XOR gates, where the canonical circuit is the s-gate circuit consisting of AND gates.

We start with presenting a generalized output selection functionality (Fig. 5). Here, instead of the specific way of feeding the shared input into the functionality, we tailor it to work with generic BHR-style sharing, where one party will hold garbled output, and the other will hold the decoding information. We note that protocol $\Pi^{\mathsf{gen\text{-}out}}$ of Fig. 6 implementing $\mathcal{F}^{\mathsf{gen\text{-}out}}$ can be refined if circuit garbling supports bitwise decoding (Definition 6). In the formal construction (Fig. 7), we omit this and other natural enhancements (such as considering specific bits of the output labels or producing shared output *a-la* functionality $\mathcal{F}^{\mathsf{out\text{-}s}}$) for simplicity of presentation.

PARAMETERS:

- Two parties: P_1 and P_2. Circuit garbling scheme CGS.

  P_1's first input $j \in [1..n]$ denotes the selection index. P_1's second input is the decoding information for a circuit with n_{out} output wires.

  P_2's input is a vector of vectors $\mathsf{out} = \mathsf{out}_1, ..., \mathsf{out}_n$, where each out_i is a string representing garbled output Y_i.

FUNCTIONALITY:

Players submit their inputs to $\mathcal{F}^{\mathsf{gen\text{-}out}}$.
$\mathcal{F}^{\mathsf{gen\text{-}out}}$ outputs $\mathsf{Decode}(d, Y_j)$.

Fig. 5. Generalized output selection functionality $\mathcal{F}^{\mathsf{gen\text{-}out}}$

Theorem 1. *Let* $\mathsf{CGS} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{eval}, \mathsf{Embed})$ *be a topology-decoupling circuit garbling scheme (Definitions 4 and 5) with respect to a class* C *and a canonical circuit* $\mathcal{C}_{\mathsf{can}}$*. Then the construction of Fig. 7 securely implements the* $\mathcal{S}$*-UC functionality of Fig. 2.*

Proof. **Security against corrupt** Gen**.** This part of the security proof closely follows the standard Yao GC proof and is omitted. Indeed, the view of Gen only includes the messages sent by the OT executions and the messages received as part of the output selection functionality $\mathcal{F}^{\mathsf{gen\text{-}out}}$, and is easily simulated (in part) by plugging in the output of the corresponding simulators.

Security against corrupt Ev**.** Intuitively, we need to argue that Ev does not gain additional information from evaluating the *same* GC $\widehat{\mathcal{C}}$ when interpreted as garblings of different circuits $\mathcal{C}_i$. We exhibit a simulator $\mathsf{Sim}_{\mathsf{Ev}}$ and prove that its output is indistinguishable from the real execution. We will rely on the topology decoupling property of circuit garbling, as formalized by Definition 4.

Constructing $\mathsf{Sim}_{\mathsf{Ev}}$. Recall, the simulator $\mathsf{Sim}_{\mathsf{Ev}}(y, z)$ knows the set of circuits $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$, where each $\mathcal{C}_i \in C(\mathcal{C}_{\mathsf{can}})$. $\mathsf{Sim}_{\mathsf{Ev}}(y, z)$ takes as input the true input and the output of the real execution and outputs the simulated view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$.

PARAMETERS:

1. Two parties: P_1 and P_2.

   P_1's first input $j \in [1..n]$ denotes the selection index. P_1's second input is the decoding information for a circuit with n_{out} output wires.

   P_2's input is a vector of vectors $\mathsf{out} = \mathsf{out}_1, ..., \mathsf{out}_n$, where each out_i is a string representing garbled output Y_i.

PROTOCOL:

1. Let $\mathcal{C}$ be the output selection circuit, computing (non-shared) Generalized Output Selection Functionality $\mathcal{F}^{\mathsf{gen\text{-}out}}$ of Figure 5.
2. Players P_1 and P_2 execute Yao's GC protocol evaluating $\mathcal{C}$ using classical GC protocol of [LP04], and using the OT extension protocol of [IKNP03].

Fig. 6. Generalized output selection protocol $\Pi^{\mathsf{gen\text{-}out}}$ realizing $\mathcal{F}^{\mathsf{gen\text{-}out}}$ of Fig. 5.

PARAMETERS:

Parties Gen and Ev with inputs $x \in \{0,1\}^{n_{\mathsf{in\text{-}x}}}$ and $y \in \{0,1\}^{n_{\mathsf{in\text{-}y}}}$ respectively. A set of Boolean circuits $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$, where each $\mathcal{C}_i \in C(\mathcal{C}_{\mathsf{can}})$. All circuits $\mathcal{C}_i$ have the same number of input wires n_{in}, $n_{\mathsf{in\text{-}x}}$, $n_{\mathsf{in\text{-}y}}$, and the same number of output wires. Let $\mathcal{C}_t \in \mathcal{S}$ be the circuit Gen wishes to evaluate.

PROTOCOL:

1. Gen runs $((T_{\mathsf{can}}, E_{\mathsf{can}}), e_{\mathsf{can}}, d_{\mathsf{can}}) \leftarrow \mathsf{Garble}(1^\kappa, \mathcal{C}_{\mathsf{can}})$ and $((T, E), e, d) \leftarrow \mathsf{Garble}(1^\kappa, \mathcal{C}_t)$. Gen sends $E' = \mathsf{Embed}(E, E_{\mathsf{can}})$ to Ev.
2. Given encoding information e, Gen sends to Ev active wire labels for the wires on which Gen provides input.
3. For i-th input bit of Ev, Gen and Ev execute Oblivious Transfer (OT), where Gen plays the role of the Sender, and Ev plays the role of the Receiver.
   (a) Gen's two input secrets are the two labels of the target GC $\widehat{\mathcal{C}}_t$ for the wire corresponding to Ev's i-th input. Ev's choice-bit input is its input on that wire.
   (b) Upon completion of OT, Ev receives active wire label on the wire. This label will be used as active wire in each of the circuits in $\mathcal{S}$.
4. Ev had received input encoding X by executing Steps 2 and 3. For each circuit $\mathcal{C}_j \in \mathcal{S}$, Ev obtains its topology T_j by running $((T_j, E_j), e_j, d_j) \leftarrow \mathsf{Garble}(1^\kappa, \mathcal{C}_j)$ and discarding all but T_j. For each T_j, Ev computes output labels $Y_j = \mathsf{Eval}((T_j, E'), X)$.
5. Gen and Ev execute a Generalized Output Selection protocol $\Pi^{\mathsf{gen\text{-}out}}$ of Figure 6. Ev provides as input n candidate garbled outputs Y_j obtained in Step 4. Gen provides as input the index t of the target circuit, as well as the decoding information d.
6. Players output whatever $\Pi^{\mathsf{gen\text{-}out}}$ outputs.

Fig. 7. $\mathcal{S}$-universal garbled circuit protocol

$\mathsf{Sim}_{\mathsf{Ev}}$ starts by emulating the GC $\widehat{\mathcal{C}}$ received in Step 1 of Fig. 7. To do so, $\mathsf{Sim}_{\mathsf{Ev}}$ simply runs $((T, E), e, d) \leftarrow \mathsf{Garble}(1^\kappa, \mathcal{C}_{\mathsf{can}})$ and adds E to the view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$.

$\mathsf{Sim}_{\mathsf{Ev}}$ proceeds to emulate Step 2 by parsing $(X_1^0, X_1^1, ..., X_{n_{\mathsf{in}}}^0, X_{n_{\mathsf{in}}}^0) \leftarrow e$[3]. For each index i of the wire provided by Gen, $\mathsf{Sim}_{\mathsf{Ev}}$ adds X_i^0, to the view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$.

$\mathsf{Sim}_{\mathsf{Ev}}$ emulates the receiver's view of OT (Step 3) by calling the provided OT simulator $\mathsf{Sim}_{\mathsf{OT}}$. $\mathsf{Sim}_{\mathsf{Ev}}$ provides input y to $\mathsf{Sim}_{\mathsf{OT}}$ as well as the corresponding input wire labels X_i^j generated by Garble above. This is intended to simulate the labels that are output as the result of OT. $\mathsf{Sim}_{\mathsf{Ev}}$ appends the Receiver view generated by $\mathsf{Sim}_{\mathsf{OT}}$ to the view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$.

$\mathsf{Sim}_{\mathsf{Ev}}$ emulates the evaluation of all n circuits of Step 4 by honestly executing Step 4 n times as prescribed in the protocol of Fig. 7. As a result, $\mathsf{Sim}_{\mathsf{Ev}}$ obtains n garbled outputs Y_j. These Y_j are implicit in the already-generated view and hence are not formally included in $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$.

Next $\mathsf{Sim}_{\mathsf{Ev}}$ simulates Step 5 by calling the simulator $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$ for the protocol $\Pi^{\mathsf{gen\text{-}out}}$ realizing $\mathcal{F}^{\mathsf{gen\text{-}out}}$. For this, $\mathsf{Sim}_{\mathsf{Ev}}$ calls the simulator with input n garbled outputs Y_j, as well as the vector of the output labels received as $\mathsf{Sim}_{\mathsf{Ev}}$ input. $\mathsf{Sim}_{\mathsf{Ev}}$ appends the output of the simulator to its view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$, outputs its view and terminates.

Indistinguishability of the simulation. It is easy to verify that the simulation is indistinguishable from the real execution.

Firstly, garbled circuits and the input encoding obtained in Steps 1–3 of the real protocol are indistinguishable from the corresponding simulated view. This is because the circuit garbling scheme CGS is topology-decoupling (cf. Definition 4). Indeed, the real view is $((T, \mathsf{Embed}(E, E_{\mathsf{can}})), X)$, and the simulated view is $((T_{\mathsf{can}}, E_{\mathsf{can}}), X_{\mathsf{can}})$. The game $\mathsf{topo}_{\mathsf{CGS}, f_{\mathsf{can}}}$ of Definition 4 immediately implies that the above distributions are indistinguishable.

As a result, the view $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$ up until the call to a realization $\Pi^{\mathsf{gen\text{-}out}}$ of $\mathcal{F}^{\mathsf{gen\text{-}out}}$ is indistinguishable.

Simulation of $\Pi^{\mathsf{gen\text{-}out}}$ step is done by referring to the $\Pi^{\mathsf{gen\text{-}out}}$ simulator $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$. We need to take care of two details:

1. ensure that input-output relationship of the call to $\Pi^{\mathsf{gen\text{-}out}}$ is consistent with the $\mathsf{View}_{\mathsf{Sim}_{\mathsf{Ev}}}$ generated so far. Indeed, this is consistent. $\mathsf{Sim}_{\mathsf{Ev}}$ will provide to $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$ the output labels it ($\mathsf{Sim}_{\mathsf{Ev}}$) received as input, as well as the collection of the candidate garbled labels Y_j that $\mathsf{Sim}_{\mathsf{Ev}}$ had computed.
2. ensure that simulation provided by $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$ is good, despite the fact that the P_2 input provided to $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$ *different* from the input provided to $\Pi^{\mathsf{gen\text{-}out}}$ in the real execution.
   This is slightly more involved, and will require looking inside how the standard simulators work for GC-based secure computation. We show that there exists a simulator of P_2 which works with our proof. Intuitively, this is possible because Receiver's real input is only used in the OT component of the simulation of $\Pi^{\mathsf{gen\text{-}out}}$, where it is not used in an essential manner.
   Consider the standard simulators of GC evaluator $\mathsf{Sim}_{\mathsf{Ev}-\mathsf{LP}}$ of [LP04] and

[3] While BHR do not formally require that such a parsing is possible, it is quite unnatural to not permit it, and all current garbling schemes allow it. Futher, in their examples BHR implicitly assume existence of such parsing.

of the OT Receiver $\mathsf{Sim}_{\mathsf{R-IKNP}}$ of [IKNP03]. $\mathsf{Sim}_{\mathsf{Ev-LP}}$ accepts as input the player's input y and the function output $f(x, y)$; it uses y only to pass it (together with the needed OT output) to the OT Receiver simulator $\mathsf{Sim}_{\mathsf{R-IKNP}}$. "Inside" $\mathsf{Sim}_{\mathsf{R-IKNP}}$ of [IKNP03], the simulation succeeds independently of what y is.
Finally, even though the input provided to $\mathsf{Sim}_{\Pi^{\mathsf{gen\text{-}out}}}$ *is not the same* as the input provided to $\Pi^{\mathsf{gen\text{-}out}}$ in the real execution, this is not a problem, as they are still computationally indistinguishable when considered as part of the probability ensemble $\{f_1(x, y), S_2(y, f_2(x, y))\}$, as argued above.

□

Achieving shared-output functionality $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC\text{-}s}}$. The protocol of Fig. 7 can be naturally extended to implement $\mathcal{F}^{\mathcal{S}\text{-}\mathsf{UC\text{-}s}}$ of Fig. 3. This is achieved, e.g., by running a shared-output version $\Pi^{\mathsf{gen\text{-}out\text{-}s}}$ of the output selection protocol $\Pi^{\mathsf{gen\text{-}out}}$. In turn, $\Pi^{\mathsf{gen\text{-}out\text{-}s}}$ is derived from $\Pi^{\mathsf{gen\text{-}out}}$ by not sending the GC decoding information to the other player and not reconstructing the plaintext output in the last stage of $\Pi^{\mathsf{gen\text{-}out}}$.

6.1 Standard Garbling Schemes are Topology-Decoupling with Extended Correctness

It is easy to verify that standard GC schemes (classical Yao, Free-XOR and half-gates) satisfy the required notion. For classic Yao, the topology decoupling is with respect to the class C of circuits with maximal circuit size s. For the Free-XOR [KS08a] and half-gates [ZRE15] schemes, C is the set of circuits of $\leq s$ non-XOR gates. In all three cases, the canonical circuit is the circuit consisting of s AND gates and can have any number of inputs and outputs.

In our generic treatment of circuit garbling schemes, we ask the scheme to define the Embed procedure. For the above standard schemes, we define the Embed procedure as follows:

Construction 1 (Embedfor standard schemes). *For classic Yao, Free-XOR and half-gates, we define* Embed *to be the syntactic procedure of replacing the prefix of the cryptographic material E_2 with another cryptographic material E_1. For $|E_1| \leq |E_2|$, we say $E = \mathsf{Embed}(E_1, E_2)$, if E is the string equal to E_2 whose first $|E_1|$ bits are set to be to be the string E_1.*

Theorem 2 (Half-gates garbling is topology-decoupling with extended correctness). *Assuming hashing functions used in the half-gates construction are modeled as a random oracle, the half-gates garbling of [ZRE15] with the above* Embed *satisfies Definition 4 with respect to the class C of circuits with the same number of input and output wires, and with number of non-XOR gates less or equal to s, where canonical circuit is the circuit consisting of s AND gates. The half-gates garbling also satisfies Definition 5.*

Proof. We first observe that syntactically the functions defined by the half-gates scheme will work with switched topology. In particular, the Eval procedure of the half-gates scheme *does not* check that there are leftover garbled tables after completion. We defined the Embed procedure above. No additional syntactic changes are needed to consider the scheme.

Extended correctness is immediate. Evaluation of the embedded half-gates GC will proceed identically to the standard half-gates evaluation.

Topology-decoupling. Recall, in the half-gates scheme, the insight is for the generator Gen to generate a uniformly random bit r, and to transform the original AND gate $v_c = v_a \wedge v_c$ into two half gates involving r:

$$v_c = (v_a \wedge r) \oplus (v_a \wedge (r \oplus v_b))$$

This has the same value as $v_a \wedge v_b$ since it distributes to $v_a \wedge (r \oplus r \oplus v_b)$. Observe that in the first conjunction $(v_a \wedge r)$ the generator Gen knows r, and in the second conjunction $(v_a \wedge (r \oplus v_b))$, the evaluator Ev is allowed to learn $(r \oplus v_b)$. In both conjunctions one of the players knows one of the inputs in plaintext. ZRE call them half-gates, a generator half-gate and an evaluator half-gate.

ZRE [ZRE15] then use a standard construction of two-row tables for each of the half-gates. Further, a standard garbled-row reduction technique is applied to reduce size of each to a single row. Finally, the $v_c = v_a \wedge v_c$ is computed via the Free-XOR technique. Importantly for our proof, the garbled rows look random and do not contain any redundant information allowing evaluator to verify that it is evaluating a correctly garbled circuit.

Specifically, let p_a, p_b be the random permutation bits selected by Gen, and Δ is the Free-XOR offset also selected by Gen. The generator half-table row T_{Gen} is set to: $T_{\mathsf{Gen}} \leftarrow H(W_a^0) \oplus H(W_a^1) \oplus p_b \Delta$. The evaluator half-table row T_{Ev} is set to $T_{\mathsf{Ev}} \leftarrow H(W_b^0) \oplus H(W_b^1) \oplus W_a^0$. The garbled output wire W_c^0 is set to be $W_c^0 \leftarrow H(W_a^0) \oplus p_a T_{\mathsf{Gen}} \oplus H(W_b^0) \oplus p_b(T_{\mathsf{Ev}} \oplus W_a^0)$, and the other output label is set $W_c^1 = W_c^0 \oplus \Delta$. It is easy to trace this and to verify that the garbled tables output by the half-gates Gen are random-looking, assuming H is a random oracle, even given an input encoding X. That is, given input encoding X, a circuit consisting from randomly generated garbled tables is indistinguishable from the correctly generated half-gates GC corresponding to X. This immediately implies the theorem statement. □

Similar theorems can be easily proven for Free-XOR, classical Yao, and many standard constructions. The proof relies on the property that the garbled circuit generated by Gen looks random, even given an input encoding.

Theorem 3. *Assuming hashing functions used in the Free-XOR construction are modeled as a random oracle, the Free-XOR garbling of [KS08a] with the above* Embed *satisfies Definition 4 with respect to the class C of circuits with the same number of input and output wires, and with number of non-XOR gates less or equal to s, where canonical circuit is the circuit consisting of s AND gates. The half-gates garbling also satisfies Definition 5.*

Theorem 4. *Assuming hashing function used in the classical Yao construction are modeled as a random oracle, the standard 4-row Yao garbling of [LP09] with the above* Embed *satisfies Definition 4 with respect to the class C of circuits with maximal circuit size s and with the same number of input and output wires, where canonical circuit is the circuit consisting of s AND gates. This garbling also satisfies Definition 5.*

7 Performance Calculation and Comparison

It is easy to evaluate performance of our $\mathcal{S}$-UC scheme. For a set $\mathcal{S}$ of n circuits of (non-XOR gate) sizes $s_{\mathcal{C}_i}$, our communication consists of transmitting *a single* GC of size $s_{\max} = \max s_{\mathcal{C}_i}$, plus the communication needed for $\Pi^{\text{gen-out}}$. $\Pi^{\text{gen-out}}$ requires $n_{\mathsf{in}} \cdot n$ OTs and sending a circuit implementing $\Pi^{\text{gen-out}}$, which is a simple multiplexer and has approximately $n_{\mathsf{out}} \cdot n$ gates.

The multiplexer circuit can process each of n_{out} wires at cost n non-XOR gates, e.g. as follows. The selector boolean input can be a vector of zeros except with a 1 in target position t. Then, the multiplexer computes XOR of all n_{out} conjunctions (of the selector bit and branch bit).

In contrast, the standard approach involves evaluating *all* n circuits, and the total communication cost will consist of sending $\sum_{i=1}^{n} |\mathcal{C}_i|$. We note that a recent heuristic circuit overlay approach [KKW17] can reduce this cost, sometimes significantly. As discussed in Sect. 1.2, our approach is more efficient and much simpler than [KKW17] in the case considered in this work (where the evaluated function is known to Gen).

For the special and representative case where all circuits are of the same size, our approach will require sending approximately $\max |\mathcal{C}_i|$ gates, while prior work required sending $n \cdot |\mathcal{C}_i|$ gates. (The cost of [KKW17] varies depending on the effectiveness of the heuristic and is between $\max |\mathcal{C}_i|$ and $n \cdot |\mathcal{C}_i|$ gates sent.) The auxiliary costs we incur (extra OTs and evaluating $\Pi^{\text{gen-out}}$ circuit) can often be ignored, e.g. when $\mathcal{C}_i$ is large relative to the auxiliary costs.

We note that if the branch clauses are very small circuits, then the cost of output selection may overweigh the benefit of Free IF. Precise determination of the break-even point mainly depends on the relative cost of the extra round of communication we require. Beyond the cost of the extra round, there is little we pay. Our output selection algorithm involves a circuit size similar to the multiplexer that would be used in regular GC; because we use OT as input, our multiplexer requires approximately twice the number of bits as the regular GC multiplexer.

Computation costs. We stress that the computation cost is reduced for the Generator (it would generate a single branch). However it remains the same as in the standard GC for the Evaluator. This is because the Evaluator must evaluate all branches of the circuit. We stress that in typical GC deployments communication is by far the most significant bottleneck, and runtime will usually be proportional to the amount of communication required.

Acknowledgements. I would like to thank anonymous reviewers of this paper and Viet Tung Hoang for valuable suggestions on presentation.

References

[BHR12] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 12, pp. 784–796. ACM Press, October 2012

[DDK+15] Demmler, D., Dessouky, G., Koushanfar, F., Sadeghi, A.-R., Schneider, T., Zeitouni, S.: Automated synthesis of optimized circuits for secure computation. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 15, pp. 1504–1517. ACM Press, October 2015

[DKS+17] Dessouky, G., Koushanfar, F., Sadeghi, A.-R., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: 24 Annual Network and Distributed System Security Symposium (NDSS 2017). The Internet Society, February 26-March 1, 2017. To appear

[FVK+15] Fisch, B.A., et al.: Malicious-client security in blind seer: a scalable private DBMS. In: 2015 IEEE Symposium on Security and Privacy, pp. 395–410. IEEE Computer Society Press, May 2015

[GKS17] Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_16

[GO96] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)

[IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9

[KKW16] Kennedy, W.S., Kolesnikov, V., Wilfong, G.: Overlaying circuit clauses for secure computation. Cryptology ePrint Archive, Report 2016/685 (2016). http://eprint.iacr.org/2016/685

[KKW17] Kennedy, W.S., Kolesnikov, V., Wilfong, G.: Overlaying conditional circuit clauses for secure computation. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 499–528. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_18

[KMR14] Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_25

[KS08a] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40

[KS08b] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_7

[KS16] Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27

[LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant's universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016). http://eprint.iacr.org/2016/017

[LP04] Lindell, Y., Pinkas, B.: A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175 (2004). http://eprint.iacr.org/2004/175

[LP09] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)

[MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33

[PKV+14] Pappas, V., et al.: Blind seer: a scalable private DBMS. In: 2014 IEEE Symposium on Security and Privacy, pp. 359–374. IEEE Computer Society Press, May 2014

[PSS09] Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_6

[SHS+15] Songhori, E.M., Hussain, S.U., Sadeghi, A.-R., Schneider, T., Koushanfar, F.: TinyGarble: highly compressed and scalable sequential garbled circuits. In: 2015 IEEE Symposium on Security and Privacy, pp. 411–428. IEEE Computer Society Press, May 2015

[Val76] Valiant, L.G.: Universal circuits (preliminary report). In: STOC, pp. 196–203. ACM Press, New York (1976)

[WGMK16] Wang, X., Gordon, S.D., McIntosh, A., Katz, J.: Secure computation of MIPS machine code. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 99–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_6

[Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press, October 1986

[ZRE15] Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8