On Garbling Schemes with and without Privacy

Carsten Baum*

cbaum@cs.au.dk
Department of Computer Science, Aarhus University, Denmark

Abstract. Garbling schemes allow to construct two-party function evaluation with security against cheating parties (SFE). To achieve this goal, one party (the Garbler) sends multiple encodings of a circuit (called Garbled Circuits) to the other party (the Evaluator) and opens a subset of these encodings, showing that they were generated honestly. For the remaining garbled circuits, the garbler sends encodings of the inputs. This allows the evaluator to compute the result of function, while the encoding ensures that no other information beyond the output is revealed. To achieve active security against a malicious adversary, the garbler in current protocols has to send $O(s)$ circuits (where s is the statistical security parameter).
In this work we show that, for a certain class of circuits, one can reduce this overhead. We consider circuits where sub-circuits depend only on one party's input. Intuitively, one can evaluate these sub-circuits using only one circuit and privacy-free garbling. This has applications to e.g. input validation in SFE and allows to construct more efficient SFE protocols in such cases. We additionally show how to integrate our solution with the SFE protocol of [5], thus reducing the overhead even further.

1 Introduction

Background. In actively-secure Two-party Function Evaluation (SFE) two mutually distrusting parties Alice and Bob (P_a, P_b) want to jointly evaluate a function f based on secret inputs x, y that they choose individually. This is done using an interactive protocol where both parties exchange messages such that, at the end of the protocol, they only learned the correct output $z = f(x, y)$ of the computation and no other information. This also holds if one of the parties arbitrarily deviates from the protocol. The problem was originally stated by Yao in 1982 [20], who also gave the first solution for the setting of honest, but curious parties.
Given a trusted third party $\mathcal{T}$ which both P_a, P_b have access to, one can solve the problem as follows: Both send their inputs as well as a description of f which we call $\mathcal{C}_f$ to $\mathcal{T}$, which then does the following: We consider $\mathcal{C}_f$ to be a boolean circuit with dedicated input and output wires. $\mathcal{C}_f$ consists of gates of fan-in two. $\mathcal{T}$ represents the inputs x, y as assignments of $0, 1$ to the input wires of the circuit, and then the functions of the gates are applied (as soon as both input wires of a gate have an assignment) until all the output wires[1] of $\mathcal{C}_f$ are either 0 or 1. Then $\mathcal{T}$ translates the values on the output wires into z and sends it to both P_a, P_b. Yao showed in his seminal work how to replace this $\mathcal{T}$ with an interactive protocol. This technique became known as *Garbled Circuits.*

Garbled Circuits in a nutshell. In order to obtain a garbled circuit from $\mathcal{C}_f$, the garbler P_a does the following: Each gate of the circuit can be represented as a table, where for each combination of the inputs a value from $\{0, 1\}$ is assigned to the output wire. Now, the rows of this table are first shuffled and then the $0, 1$ values of the inputs and outputs are replaced by random bit strings (*keys*), such that the output key of a gate corresponds to the input key of another gate if its output is wired into the respective input in $\mathcal{C}_f$ and if they both correspond to the same value on the wire. One then stores information such that each

* Supported by The Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; by the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed; and by the Advanced ERC grant MPCPRO.

[1] We let $\mathcal{T}$ accept only descriptions of f where the graph representing the circuit $\mathcal{C}_f$ is directed and acyclic.

output key can be derived if and only if both input keys for the corresponding row are known. Such a gate is called a *Garbled Gate* and by applying this technique recursively to all gates, P_a computes a so-called *Garbled Circuit*. One then considers the gates whose inputs are the input wires of the circuit. These keys are considered as the *input keys* of the circuit. Moreover, P_a also has to store a table of the keys that belong to the output wires of the circuit.
In the next step, P_a sends the garbled circuit and the input keys corresponding to her chosen input to the evaluator P_b. He obtains his input keys from P_a by a so-called *Oblivious Transfer*(OT) protocol, where P_a inputs all possible keys and P_b starts with his input y, such that afterwards P_b only learns the keys that correspond to his input and P_a does not learn y. P_b can now evaluate the circuit gate by gate until he obtains the output keys, which he sends to P_a. Intuitively, the security of the protocol is based on the OT hiding P_b's input while the garbling hides the input of P_a (and to some extend the computed circuit).
P_a can cheat in the above protocol in multiple ways: The circuit that is computed is hidden from P_b, so it may differ from $\mathcal{C}_f$ (or he obtains input keys that do not correspond to his inputs). A solution to this problem is called the *cut-and-choose* approach, where a number of circuits is garbled and sent to P_b. He then chooses a random subset to be opened completely to him and he can check that the circuit indeed computes the right function. For the other garbled instances, the above protocol is then run multiple times in parallel and the evaluator derives the result from the outputs of these instances. This may lead to new problems, see e.g. [16,17] for details.

Garbling Schemes. The garbled circuits-approach has found many applications in cryptography, such as in verifiable computation, private set intersection, zero-knowledge proofs or functional encryption with public keys (to just name a few). Moreover, it has been treated on a more abstract level e.g. in [12] as *Randomized Encodings*. Kamara & Wei [14] discuss the idea of *special purpose garbled circuits* which do not yield full-fledged SFE but can on the other hand efficiently be instantiated using Structured Encryption Schemes and yield smaller overhead compared to directly using GC. Moreover, Bellare et al. [3] discussed garbling as a primitive having potentially different security notions, and studied how these are related. Using their framework one can compare different properties that a garbling scheme can have, such as *privacy, authenticity and obliviousness*. This allows to look for special schemes that may only implement a subset or different properties, which may be of use in certain contexts. As an example for such an application, one can e.g. consider the efficient *zero-knowledge protocol* due to Jawurek et al. [13] where the prover evaluates a garbled circuit in order to prove a certain statement.
Since only the evaluator in [13] has private inputs to the circuit and evaluates it on known values, no privacy of the inputs is necessary. A garbling scheme such as the one from [7] can then be used, which has lower overhead than comparable schemes with privacy.

The problem. In this paper, we address the following question:

Can one construct Secure Function Evaluation protocols based on a combination of garbling schemes both with and without privacy, thus reducing overhead?

The question can be thought of as a generalization of [13]: Those parts of a circuit $\mathcal{C}_f$ that do only depend on one party's input may not need to be computed with active security. Such circuits naturally arise in the case when predicates must be computed on the inputs of each party, which includes the case when signatures must be verified or inputs in a reactive computation are checked for consistency. For such functions this separate evaluation can potentially improve the runtime of SFE, as e.g. shown by [15]. While it seems intuitive that in such a case this evaluation strategy is preferable, it is unclear how to combine those schemes while not introducing new problems. In particular, one has to make sure that the outputs of the privacy-free part correspond to the inputs of the actively-secure computation.

Contributions. In this work, we describe a solution to the aforementioned problem. It can be applied for a certain class of functions that are decomposable as shown in Figure 1.
On the left side of the figure, the evaluation without optimization is shown. Here the whole circuit must be evaluated using an actively secure two-party SFE scheme, while on the right side only parts of the circuit

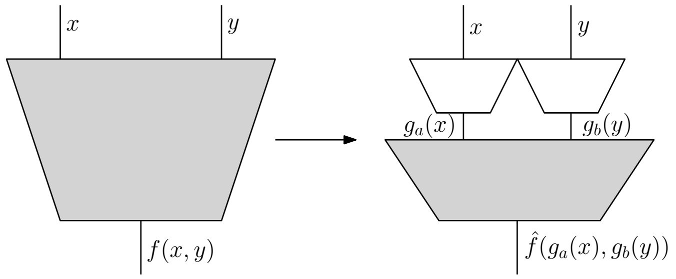

Fig. 1: A graphical depiction of the function decomposition.

(the grey circuit) will be computed with active security. Our solution allows that the evaluation of $\hat{f}$ can be done by an arbitrary SFE scheme. To achieve this goal, we use circuit augmentation for $g_a, g_b, \hat{f}$ which in itself introduces a small overhead. We will show that this overhead can mostly be eliminated using e.g. [5] as SFE scheme.
We start with the following idea: Let P_a compute a privacy-free garbling of g_b and P_b compute a privacy-free garbling of g_a. Both parties exchange and evaluate the privacy-free garbling, whose output in turn will be the input to the evaluation of $\hat{f}$. Now we must verify that both P_a and P_b take the output of their respective functions and do not replace it before inputting it into $\hat{f}$. At the same time, the outputs of g_a, g_b are confidential and we must prevent the garbler from sending an incorrect circuit or wrong input keys. Our solution will deal with the inconsistency problem by checking that the inputs to $\hat{f}$ come indeed from g_a, g_b using a hash function whose output is properly masked. This, in turn, creates new problems since such a mask can be used to tamper with the obtained hash. Therefore, care must be taken about the timing in the protocol. Details follow in Section 3.

Why not just using Zero-Knowledge proofs? Intuitively there is another solution to the above problem that avoids privacy-free garbling altogether: P_a commits to her inputs to $\hat{f}$ as $Com(g_a(x); r)$ and proves in zero-knowledge that this commitment indeed contains a value that lies in the image of g_a (P_b similarly uses g_b in the proof). Now all functions g_a, g_b are assumed to be binary circuits and the most efficient generic zero-knowledge proofs over $\mathbb{Z}_2$ are [9] and [13], where the proof-size is linear in the circuit size[2]. The crucial point is that, to the best of our knowledge, the proof itself must compute either the Com function or some verification function such as to tie the proof together with the SFE input. Computing public key-based primitives over $\mathbb{Z}_2$ incurs a huge blowup in the proof size. If one uses symmetric primitives like e.g. SHA-256 then our approach is still preferable, since computing such a hash function requires significantly more AND gates (see e.g. [19]) than computing the matrix multiplication that is required in our protocol.

Related work. Our problem shares some similarity with *Verifiable Computation* [2,8]. Here, the idea is that a weak client outsources an expensive computation to a computationally stronger but possibly malicious server. This server then performs the computation and delivers a *proof* of correct computation which the client can check (in time significantly smaller than evaluating the function itself). Our setting differs, since we want that the server performs the evaluation of the circuit on his own inputs and these must be kept secret. Moreover, we do only require one evaluation of the circuit.
Our solution, as already mentioned, bears resemblance with the concept of *Zero-Knowledge Proofs* [10,11] where a prover convinces a verifier about the truth of the statement in an interactive protocol without revealing anything but the validity of this statement. In particular (in our setting), P_a proves to P_b that her input to $\hat{f}$ lies in the image of the function g_a and vice versa. In cryptographic protocols, these proofs are often used to show that certain algebraic relations among elements hold. The fact that these proofs can also

[2] Approaches based on SNARKs have smaller proof size but require much more work on the prover's side, which is why we do not mention them.

be used to (efficiently) show that the prover knows a specific input to a circuit was already observed in [13]. In comparison to their work, we exploit this phenomenon in a more general sense.
In concurrent and independent work, Katz et al. [15] described a related approach to enforce input validity in SFE. Their techniques differ significantly from our work: Using a clever combination of OT and ElGamal encryption they can enforce that $\hat{f}$ and g_a, g_b have the same inputs, where g_a, g_b are predicates with public output (that validate the inputs of each party) and $\hat{f}$ is evaluated using SFE. Their approach is using the protocol of Afshar et al. [1] for the evaluation of $\hat{f}$ while we allow for a larger class of SFE schemes to be used.

2 Preliminaries

In this work, we let λ denote the computational and s denote the statistical security parameter. We use the standard definitions for a negligible function $\mathsf{negl}(\cdot)$ and polynomial function $\mathsf{poly}(\cdot)$. Two distributions of random variables are statistically indistinguishable if their distance is negligible in s. If instead distinguishing them breaks a computational assumption (parametrized by λ), then we consider them as computationally indistinguishable, which we denote as $\approx_c$. We use $\mathbb{B}$ as shorthand for $\{0,1\}$.
Let us assume that P_a, P_b agreed to evaluate a function $f : \mathbb{B}^{2n} \to \mathbb{B}^m$, where the first n input bits are provided by P_a and the second n input bits by P_b. We assume that the function can be decomposed into $\hat{f} : \mathbb{B}^{l_a+l_b} \to \mathbb{B}^m$, $g_a : \mathbb{B}^n \to \mathbb{B}^l_a$, $g_b : \mathbb{B}^n \to \mathbb{B}^l_b$ such that

$$\forall x, y \in \mathbb{B}^n : \ \hat{f}(g_a(x), g_b(y)) = f(x, y)$$

To be more applicable in our setting, we have to look at the functions as circuits, and will do so using an approach similar to [3].

2.1 Circuits and the Split-Input Representation

Consider the tuple $\mathcal{C}_f = (n_{in}, n_{out}, n_g, L, R, G)$ where

- $n_{in} \geq 2$ is the number of input wires, $n_{out} \geq 2$ the number of output wires and $n_g \geq 1$ the number of gates. We let $n_w = n_{in} + n_g$ be the number of wires.
- we define the sets $Inputs \leftarrow \{1, ..., n_{in}\}$, $Wires \leftarrow \{1, ..., n_w\}$ as well as $Outputs \leftarrow \{n_w - n_{out} + 1, ..., n_w\}$ and $Gates \leftarrow \{n_{in} + 1, ..., n_w\}$ to identify the respective elements in the circuit.
- the function $L : Gates \mapsto Wires \setminus Outputs$ identifies the left incoming wire and $R : Gates \mapsto Wires \setminus Outputs$ identifies the right incoming wire for each gate, with the restriction that $\forall g \in Gates : L(g) < R(g) < g$.
- the mapping $G : Gates \times \mathbb{B}^2 \mapsto \mathbb{B}$ determines the function that is computed by a gate.

To obtain the outputs of the above circuit when evaluating it on an input
$x = x_1 ... x_{n_{in}}$ one evaluates $\mathcal{C}_f$ as follows:

```
$eval(\mathcal{C}_f, x)$:
    (1) For $g = n_{in} + 1, ..., n_w$:
        (1.1) $l \leftarrow L(g), r \leftarrow R(g)$
        (1.2) $x_g \leftarrow G(g, x_l, x_r)$
    (2) Output $x_{n_w - n_{out}+1} ... x_{n_w}$
```

For a function $f : \mathbb{B}^{n_{in}} \mapsto \mathbb{B}^{n_{out}}$, we consider $\mathcal{C}_f = (n_{in}, n_{out}, n_g, L, R, G)$ as a *circuit representation of* f iff $\forall x \in \mathbb{B}^{n_{in}} : \ f(x) = eval(\mathcal{C}_f, x)$.
In order to be able to apply our solution, the circuit in question must be decomposable in a certain way as already outlined in Section 1. We will now formalize what we mean by this decomposability.

Definition 1 (Split-Input Representation (SIR)). *Let* $f : \mathbb{B}^{2n} \to \mathbb{B}^m$, $\hat{f} : \mathbb{B}^{l_a + l_b} \to \mathbb{B}^m$, $g_a : \mathbb{B}^n \to \mathbb{B}^l_a$, $g_b : \mathbb{B}^n \to \mathbb{B}^l_b$ *be functions such that*

$$\forall x, y \in \mathbb{B}^n : \ \hat{f}(g_a(x), g_b(y)) = f(x, y)$$

Let moreover $\mathcal{C}_f, \mathcal{C}_{\hat{f}}, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$ *be their respective circuit representations. Then we call* $\mathcal{C}_{\hat{f}}, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$ *the* Split-input representation *of* $\mathcal{C}_f$*.*

For every function h with $n \geq 2$ such a decomposition always exists, but it is only of interest in our setting if (intuitively) $n_g(\mathcal{C}_{\hat{f}}) \ll n_g(\mathcal{C}_f)$.

2.2 Secure Two-Party Computation and Garbling Schemes

The notion of an SFE protocol is described by a protocol between two parties P_a, P_b that securely implements Figure 2.

Functionality $\mathcal{F}_{\text{SFE\&CommitOT}}$

The input x to the circuit is split up into j blocks $I_1, ..., I_j$, where each block is provided by either P_a, P_b or both.

Initialization:
- On input $(\mathsf{init}, \mathcal{C}, I_1, ..., I_j)$ from both P_a, P_b where $\mathcal{C} = (n, m, g, L, R, G)$ is a circuit, store $\mathcal{C}$. Moreover, the parties agree on a set of disjoint subsets $I_i \subseteq [n]$ such that $\bigcup I_i = [n]$.

Commit:
- Upon input (commit, id, x) from either P_a or P_b and if id was not used before, store (id, x, P_a) if the command was sent by P_a, and (id, x, P_b) otherwise. Then send (commit, id) to both parties.

Open:
- Upon input (open, id) by P_a and if (id, x, P_a) was stored, output (open, id, x) to P_b.
- Upon input (open, id) by P_b and if (id, x, P_b) was stored, output (open, id, x) to P_a.

One-sided Committed OT:
- On input (cotB, id) from P_b and $(\mathsf{cotB}, id, \{y_0^i, y_1^i\}_{i \in [l]})$ by P_a and if there is a (id, x, P_b) stored with $x = x_1...x_l$, then output $(\mathsf{ot}, \{y_{x_i}^i\}_{i \in [l]})$ to P_b.

Input by both parties:
- Upon input (input, id, x) by both parties and if id was not used before, store $(id, x, \sim)$.

Input of P_a:
- Upon input (inputA) from P_a where there is a $(I_i, x_i, \cdot)$ stored for each $i \in [j]$, output (inputA) to P_b.

Input of P_b:
- Upon input (inputB) from P_b where inputA was obtained, load all x_i from $(I_i, x_i, \cdot)$, compute $z \leftarrow eval(\mathcal{C}, x_1...x_j)$ and output (output, z) to P_b.

Fig. 2: SFE, commitments and committed OT for two parties.

Note that $\mathcal{F}_{\text{SFE\&CommitOT}}$ moreover provides *commitments* and[3] *committed OT* [4]. Committed OT resembles OT as depicted in Figure 3, but where the choice of the receiver is determined by a commitment.

The main reason why we need this specific functionality $\mathcal{F}_{\text{SFE\&CommitOT}}$ is that we have to ensure consistency of inputs using the commitments between the actively secure scheme and the privacy-free part, and having all of these as one functionality simplifies the proof.
Out of the framework of [3] we will now recap the notion of *projective verifiable garbling schemes*. We require the properties *correctness, authenticity and verifiability*. These intuitively ensure that the evaluated circuit

[3] These are building blocks are used in many SFE protocols. We hence assume that they are available and cheap.

Functionality $\mathcal{F}_{\mathrm{OT}}$

OT for P_a:

– On input (otA, x) from P_a and $(\mathsf{otA}, \{y_0^i, y_1^i\}_{i\in[l]})$ by P_b and if $x = x_1...x_l$, output $(\mathsf{ot}, \{y_{x_i}^i\}_{i\in[l]})$ to P_a.

OT for P_b:

– On input (otB, x) from P_b and $(\mathsf{otB}, \{y_0^i, y_1^i\}_{i\in[l]})$ by P_a and if $x = x_1...x_l$, output $(\mathsf{ot}, \{y_{x_i}^i\}_{i\in[l]})$ to P_b.

Fig. 3: Functionality for OT.

shall compute the correct function, only leak the output keys that can be obtained using the provided input keys and that one can check after the fact (i.e. when obtaining all the input keys) whether the circuit in fact was a garbling of a certain function.

Let λ be a security parameter and $\mathcal{G} = (Gb, En, De, Ev, Ve)$ be a tuple of (possibly randomized) algorithms such that

$Gb(1^\lambda, \mathcal{C}_f)$**:** On input $1^\lambda, \mathcal{C}_f$ where $n_{in}, n_{out} = \mathsf{poly}(\lambda), n \geq \lambda$ and $|\mathcal{C}_f| = \mathsf{poly}(\lambda)$ the algorithm outputs a triple (F, e, d) where we call F the garbled circuit, e the input encoding information and d the output decoding information.

$En(e, x)$**:** On input e, x where $e = \{X_i^0, X_i^1\}$ is a set of keys representing the input wires, output X such that $X_i \leftarrow X_i^{x_i}$ i.e. output the 0 key for input i if $x_i = 0$ and vice versa for $x_i = 1$.

$Ev(F, X, x)$**:** On input (F, X, x) where F, X are outputs of the above algorithms, evaluate the garbled circuit F on the input keys X to produce output keys Z.

$De(Z, d)$**:** Let Z, d be input to this algorithm, where $d = \{Z_i^0, Z_i^1\}$ and Z contains l elements. The algorithm outputs a string $z \in \{0, 1, \perp\}^l$ where $z_i \leftarrow b$ if $Z_i = Z_i^b$, and $z_i \leftarrow \perp$ if $Z_i \notin \{Z_i^0, Z_i^1\}$.

$Ve(\mathcal{C}_f, F, e)$**:** On input $\mathcal{C}_f, F, e$ with the same semantics as above, the algorithm outputs 1 if F, e is a garbling of $\mathcal{C}_f$.

The definitions are according to [7]. Correctness is straightforward and implies that combining the above algorithms yields the expected output from evaluating f directly.

Definition 2 (Correctness). *Let $\mathcal{G}$ be a verifiable projective garbling scheme. Then $\mathcal{G}$ is correct if for all $n_{in}, n_{out} = \mathsf{poly}(\lambda), f : \mathbb{B}^{n_{in}} \to \mathbb{B}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x \in \mathbb{B}^{n_{in}}$ it holds that*

$$\Pr\left[De(Ev(F, (X_i^{x_i}), x), d) \neq f(x) \mid (F, e, d) \leftarrow Gb(1^\lambda, \mathcal{C}_f) \ \wedge (X_i^{x_i}) \leftarrow En(e, x)\right] \leq \mathsf{negl}(\lambda)$$

Authenticity is very important for our later application. It prevents the adversary from successfully outputting other output keys than those he can derive from the input keys and the garbling.

Definition 3 (Authenticity). *Let $\mathcal{G}$ be a verifiable projective garbling scheme. Then $\mathcal{G}$ provides authenticity if for all $n_{in}, n_{out} = \mathsf{poly}(\lambda), f : \mathbb{B}^{n_{in}} \to \mathbb{B}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x \in \mathbb{B}^{n_{in}}, y \in \mathbb{B}^{n_{out}}$ with $y \neq f(x)$ it holds that*

$$\Pr\left[De(\mathcal{A}(\mathcal{C}_f, F, (X_i^{x_i}), x), d) = y \mid (F, e, d) \leftarrow Gb(1^\lambda, \mathcal{C}_f) \ \wedge (X_i^{x_i}) \leftarrow En(e, x)\right] \leq \mathsf{negl}(\lambda)$$

for every $\mathcal{A}$ that is running in probabilistic polynomial time in λ.

In the definition of verifiability one has to consider that the Ve algorithm can also output 1 for adversarially chosen garblings F'. In such a case, we require that no information about the input is leaked if the evaluator honestly evaluates the garbled circuit.

Definition 4 (Verifiability). *Let $\mathcal{G}$ be a verifiable projective garbling scheme. Then $\mathcal{G}$ has verifiability if for all $n_{in}, n_{out} = \mathsf{poly}(\lambda), f : \mathbb{B}^{n_{in}} \to \mathbb{B}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x, y \in \mathbb{B}^{n_{in}}, x \neq y, f(x) = f(y)$ it holds that*

$$\Pr\left[Ev(F, (X_i^{x_i}), x) \neq Ev(F, (X_i^{y_i}), y) \mid Ve(\mathcal{C}_f, F, \{X_i^0, X_i^1\}) = 1 \ \wedge (F, \{X_i^0, X_i^1\}) \leftarrow \mathcal{A}(1^\lambda, \mathcal{C}_f)\right] \leq \mathsf{negl}(\lambda)$$

for every probabilistic polynomial-time $\mathcal{A}$.

A garbling scheme $\mathcal{G}$ that fulfils all the above three conditions will from now on be called *privacy-free.*

2.3 Universal Hash Functions

A third ingredient that we need for our protocol are universal hash functions. For such a function two inputs will yield the same output only with small probability for as long as the function itself is randomly chosen *after the inputs are fixed.* This is a rather weak requirement in comparison to e.g. collision-resistant hash functions, but it is strong enough in our setting: If the circuits are first garbled and the inputs are fixed before the hash function is chosen, then the chance of two inputs colliding is very small (even though the universal hash function might be easily invertible).

Definition 5 (Universal Hash Function). *Let $\mathcal{H} = \{h : \mathbb{B}^m \to \mathbb{B}^s\}$, then $\mathcal{H}$ is a family of universal hash functions if*

$$\forall x, y \in \mathbb{B}^m, x \neq y \ : \ \Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] \leq 2^{-s}$$

A family of universal hash functions has the uniform difference property if

$$\forall x, y \in \mathbb{B}^m, x \neq y,\ \forall z \in \mathbb{B}^s \ : \ \Pr_{h \in_R \mathcal{H}}[h(x) \oplus h(y) = z] \leq 2^{-s}$$

An family of functions that we will later use is defined as follows:

Definition 6. *Let $t \in \mathbb{B}^{m+s-1}$ and $M \in \mathbb{B}^{s \times m}$ such that $M_{i,j} = t_{i+j-1}$ and define $h_t : x \mapsto Mx$. Moreover, define the family $\mathbb{H}$ as $\mathbb{H} = \{h_t \mid t \in \mathbb{B}^{m+s-1}\}$.*

Remark 1. $\mathbb{H}$ is a family of universal hash functions with the uniform difference property.

Proof. See [5, Appendix E]

3 Construction

In our protocol, we use the functions defined above to protect against the adversary providing an inconsistent input to $\hat{f}$. To do so, we augment the computed circuits slightly. A graphical depiction of that can be found in Figure 4.

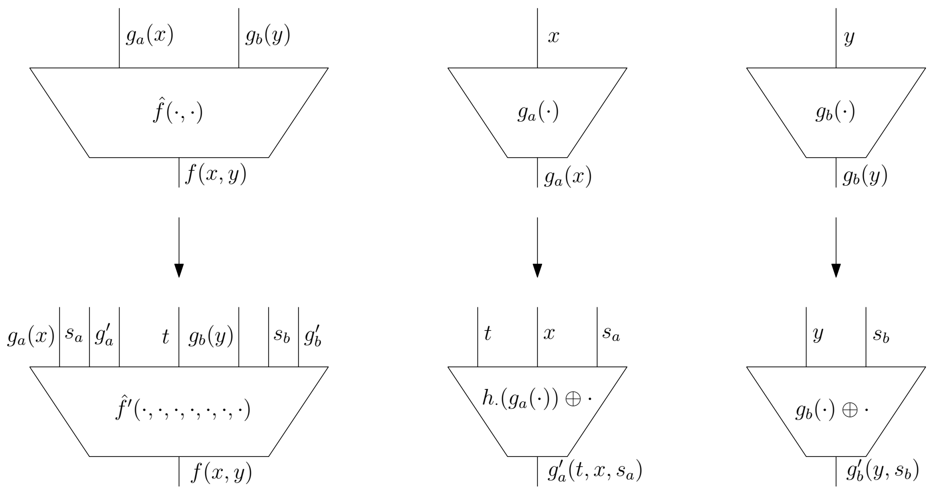

Fig. 4: The functions and how they will be augmented.

The solution is tailored for protocols with one-sided committed OT (which is normally available for SFE schemes based on garbled circuits). If there is committed OT for both or none of the parties, then the protocol and function augmentation can be adjusted in a straightforward manner.

We let $f, \hat{f}, g_a, g_b$ be functions as defined before. To compute a *proof* that P_a computed g_a correctly, we will make P_a additionally compute a *digest* on the output of g_a. Therefore, we augment g_a with a universal hash function h_t drawn from $\mathbb{H}$ to which P_a then adds a random string s_a that is fixed in advance. As such, the output will not reveal any information about the computed value. On the other hand, since P_a will commit to the input before h_t is chosen, the inputs $g_a(x), s_a$ to $\hat{f}$ will differ from the output g_a' with high probability. We observe that t, g_a' can be public inputs to $\hat{f}'$.

$$\begin{aligned} g_a' : \mathbb{B}^{l_a+s-1} \times \mathbb{B}^n \times \mathbb{B}^{l_a} &\to \mathbb{B}^{l_a} \\ (t, x, s_a) &\mapsto h_t(g_a(x)) \oplus s_a \end{aligned}$$

In the case of P_b, it is not necessary for him to compute an actual hash of $g_b(y)$. This is because only P_a can arbitrarily send differing inputs for $\hat{f}$ by choosing different values that blind her input (whereas committed OT is available for P_b to circumvent this). Nevertheless, P_b adds a one-time-pad s_b to $g_b(y)$, so that we once again can make the value g_b' a public input to $\hat{f}'$.

$$\begin{aligned} g_b' : \mathbb{B}^n \times \mathbb{B}^{l_b} &\to \mathbb{B}^{l_b} \\ (y, s_b) &\mapsto g_b(y) \oplus s_b \end{aligned}$$

The actively secure protocol will evaluate $\hat{f}$ on the inputs $g_a(x), g_b(y)$ as before. The correct value will only be output of $\hat{f}'$ if, given the auxiliary inputs s_a, s_b and the public inputs t, g_a', g_b' it holds that $h_t(g_a(x)) \oplus s_a = g_a'$ and $g_b(y) \oplus s_b = g_b'$. Otherwise, an abort symbol $\perp$ will be delivered:

$$\begin{aligned} \hat{f}' : \begin{pmatrix} \mathbb{B}^{l_a} \times \mathbb{B}^s \times \mathbb{B}^s \times \mathbb{B}^{l_a+s-1} \times \\ \mathbb{B}^{l_b} \times \mathbb{B}^{l_b} \times \mathbb{B}^{l_b} \end{pmatrix} &\to \mathbb{B}^m \cup \{\perp\} \\ (g_a(x), s_a, g_a', t, g_b(y), s_b, g_b') &\mapsto \begin{cases} \hat{f}(g_a(x), g_b(y)) & \text{if } g_b(y) \oplus s_b = g_b' \ \wedge h_t(g_a(x)) \oplus s_a = g_a' \\ \perp & \text{else} \end{cases} \end{aligned}$$

The protocol will be as follows:

Input phase Both parties P_a, P_b first locally compute $g_a(x), g_b(y)$. They then commit to the inputs $x, y, s_a, s_b, g_a(x), g_b(y)$ using $\mathcal{F}_{\text{SFE\&CommitOT}}$.

Function sampling P_b samples a hash function $h_t \in \mathbb{H}$ and sends its description t to $\mathcal{F}_{\text{SFE\&CommitOT}}$. He then sends a privacy-free garbling of $g_a'(\cdot, \cdot, \cdot)$. P_a sends a privacy-free garbling of a circuit computing $g_b'(\cdot, \cdot)$ to P_b.

Privacy-free phase P_b uses committed OT to obtain the input keys that correspond to the his commitments from the input phase. P_a uses $\mathcal{F}_{\text{OT}}$. Afterwards, P_b decommits t and thereby reveals the hash function h_t. They then evaluate the privacy-free garblings locally and commit to the output keys.

Check phase P_a, P_b open the whole privacy-free garbling towards the other party. They each verify that the circuit was constructed correctly and afterwards open the commitments to the output keys. These values are then used as public inputs g_a', g_b' to $\hat{f}'$ in the next step.

Computation phase P_a and P_b evaluate $\hat{f}'$ securely using SFE. The inputs are defined by the commitments from the input phase and the opened commitments from the check phase.

The concrete protocol. We are now ready to present the protocol as outlined in the previous subsection. It can be found in Figure 5.

4 Security

We will now prove the security of the protocol from the previous section. More formally, consider the stripped-down functionality in Figure 6 which focuses on the SFE.

Protocol Π_{SIREVAL}

Both parties P_a, P_b want to evaluate a function $f : \mathbb{B}^{2n} \to \mathbb{B}^m$ and we consider its SIR $\mathcal{C}_{\hat{f}}, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$. P_a has input $x \in \mathbb{B}^n$ and P_b has input $y \in \mathbb{B}^n$.

Input phase:

(1) Let $\mathcal{C}_{\hat{f}'}, \mathcal{C}_{g'_a}, \mathcal{C}_{g'_b}$ be circuits representing $\hat{f}', g'_a, g'_b$ which were defined before.

(2) Both parties send $(\mathsf{init}, \mathcal{C}_{\hat{f}'}, ”g_a(x)”, ”s_a”, ”g'_a”, ”h_t”, ”g_b(y)”, ”s_b”, ”g'_b”)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

(3) P_a computes $g_a(x)$ locally and chooses $s_a \in_R \mathbb{B}^s$. P_b computes $g_b(y)$ locally and chooses $t \in_R \mathbb{B}^{l_a+s-1}$, $s_b \in_R \mathbb{B}^{l_b}$.

(4) P_a sends $(\mathsf{commit}, ”g_a(x)”, g_a(x))$, $(\mathsf{commit}, ”s_a”, s_a)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.
P_b sends $(\mathsf{commit}, ”y”, y)$, $(\mathsf{commit}, ”g_b(y)”, g_b(y))$, $(\mathsf{commit}, ”s_b”, s_b)$, $(\mathsf{commit}, ”h_t”, t)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

Function sampling:

(1) P_a computes $(F_b, \{y^i_0, y^i_1\}_{i\in[n]}\{s^i_{0,b}, s^i_{1,b}\}_{i\in[l_b]}, d_b) \leftarrow Gb(1^s, \mathcal{C}_{g'_b})$ and sends F_b to P_b.

(2) P_b computes $(F_a, \{t^i_0, t^i_1\}_{i\in[l_a+s-1]}\{x^i_0, x^i_1\}_{i\in[n]}\{s^i_{0,a}, s^i_{1,a}\}_{i\in[s]}, d_a) \leftarrow Gb(1^s, \mathcal{C}_{g'_a})$ and sends F_a to P_a.

Privacy-free phase:

(1) P_a sends (otA, x) and P_b sends $(\mathsf{otA}, \{x^i_0, x^i_1\}_{i\in[n]})$ to $\mathcal{F}_{\text{OT}}$, hence P_a obtains $\{x^i\}_{i\in[n]}$. They do the same for $”s_a”$ so P_a obtains $\{s^i_a\}_{i\in[s]}$. Moreover, P_b sends $\{t^i\}_{i\in[l_a]+s-1}$ to P_a.

(2) Conversely, P_b sends $(\mathsf{cotB}, ”y”)$ and P_a sends $(\mathsf{cotB}, ”y”, \{y^i_0, y^i_1\}_{i\in[n]})$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$, hence P_b obtains $\{y^i\}_{i\in[n]}$. They do the same for $”s_b”$ so P_b obtains $\{s^i_b\}_{i\in[l_b]}$.

(3) P_b sends $(\mathsf{open}, ”h_t”)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

(4) P_a evaluates the privacy-free garbling as $(g^{i\prime}_a)_{i\in[s]} \leftarrow Ev(F_a, \{t^i\}_{i\in[l_a]+s-1}\{x^i\}_{i\in[n]}\{s^i_a\}_{i\in[s]}, txs_a)$ and then commits to $(g^{i\prime}_a)_{i\in[s]}$.

(5) P_b evaluates the privacy-free garbling as $(g^{i\prime}_b)_{i\in[l_b]} \leftarrow Ev(F_b, \{y^i\}_{i\in[n]}\{s^i_b\}_{i\in[l_b]}, ys_b)$ and then commits to $(g^{i\prime}_b)_{i\in[l_b]}$.

Check phase:

(1) P_a sends $(F_b, \{y^i_0, y^i_1\}_{i\in[n]}\{s^i_{0,b}, s^i_{1,b}\}_{i\in[l_b]}, d_b)$ to P_b who checks that he obtained correct input and output keys and that $Ve(\mathcal{C}_{g'_b}, F_b, \{y^i_0, y^i_1\}_{i\in[n]}\{s^i_{0,b}, s^i_{1,b}\}_{i\in[l_b]}) = 1$. If not, then P_b aborts.

(2) P_b sends $(F_a, \{t^i_0, t^i_1\}_{i\in[l_a+s-1]}\{x^i_0, x^i_1\}_{i\in[n]}\{s^i_{0,a}, s^i_{1,a}\}_{i\in[s]}, d_a)$ to P_a who checks that she obtained correct input and output keys and that $Ve(\mathcal{C}_{g'_a}, F_a, \{x^i_0, x^i_1\}_{i\in[n]}\{s^i_{0,a}, s^i_{1,a}\}_{i\in[s]}) = 1$. If not, then she aborts.

(3) P_a opens her commitments to $(g^{i\prime}_a)_{i\in[s]}$. P_b computes $g'_a \leftarrow De((g^{i\prime}_a)_{i\in[s]}, d_a)$ and aborts if one of the indices is $\perp$. Otherwise, both send $(\mathsf{input}, ”g'_a”, g'_a)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

(4) P_b opens his commitments to $(g^{i\prime}_b)_{i\in[l_b]}$. P_a computes $g'_b \leftarrow De((g^{i\prime}_b)_{i\in[l_b]}, d_b)$ and aborts if one of the indices is $\perp$. Otherwise, both send $(\mathsf{input}, ”g'_b”, g'_b)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

Computation phase:

(1) P_a sends (inputA) to $\mathcal{F}_{\text{SFE\&CommitOT}}$, followed by P_b sending (inputB).

(2) P_b obtains (output, z) from $\mathcal{F}_{\text{SFE\&CommitOT}}$ and outputs z.

Fig. 5: Protocol Π_{SIREVAL} to evaluate SIR of a function.

Theorem 1. *Let $\mathcal{G} = (Gb, En, De, Ev, Ve)$ be a privacy-free garbling scheme, λ its computational security parameter, and s be a statistical security parameter, then Π_{SIREVAL} securely implements $\mathcal{F}_{\text{SFE}}$ in the $\mathcal{F}_{\text{SFE\&CommitOT}}, \mathcal{F}_{\text{OT}}$-hybrid model against static, malicious adversaries corrupting either P_a or P_b.*

We split the proof into two different simulators, one for P_a being corrupt and the other one for a malicious P_b, where the second one is a simplified version of the malicious-P_a simulator. The proof works as follows: In the ideal world, the simulator runs a protocol where it intercepts all the commitments coming from P_a and simulates an honest $\tilde{P}_b$ (with some default input) for the protocol. It aborts when the committed values between the stages do not match up, or when P_a sends keys that she was not supposed to obtain. Then, a hybrid argument proves the claimed statement.

Proof. As in the protocol Π_{SIREVAL} we assume that both parties P_a, P_b want to evaluate a function $f : \mathbb{B}^{2n} \to \mathbb{B}^m$ and we consider its SIR $\mathcal{C}_{\hat{f}}, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$. P_a has input $x \in \mathbb{B}^n$ and P_b has input $y \in \mathbb{B}^n$.

Functionality $\mathcal{F}_{\text{SFE}}$

Initialization:
- On input $(\mathsf{init}, \mathcal{C})$ from both P_a, P_b where $\mathcal{C} = (2n, m, g, L, R, G)$ is a circuit, store $\mathcal{C}$.

Input of P_a:
- Upon input (inputA, x) from P_a where $x \in \mathbb{B}^n$ and where no input was given by P_a before, store x and send (inputA) to P_b.

Input of P_b:
- Upon input (inputB, y) from P_b where $y \in \mathbb{B}^n$ and where no input was given by P_b before and if (inputA) was obtained by P_b, compute $z \leftarrow eval(\mathcal{C}, xy)$ and output z to P_b.

Fig. 6: Secure Function Evaluation.

Proof for malicious P_a**.** We first show a simulator $\mathcal{S}_A$ to prove that from P_a's perspective, $\mathcal{F}_{\text{SFE}} \diamond \mathcal{S}_A \approx \mathcal{F}_{\text{SFE\&CommitOT}} \diamond \mathit{\Pi}_{\text{SIREval}}$.
Let $\mathcal{T}_{P_a Real}$ be the distribution of the transcripts that are obtained by executing $\mathit{\Pi}_{\text{SIREval}}$ and $\mathcal{T}_{P_a Sim}$ be the distribution obtained from $\mathcal{S}_A$ (both of them only for a corrupted P_a), so the goal is to show that $\mathcal{T}_{P_a Real} \approx \mathcal{T}_{P_a Sim}$.

Simulator $\mathcal{S}_A$

Input phase:
(1) Start a copy of $\mathcal{F}_{\text{SFE\&CommitOT}}$ with which P_a will communicate in the simulated protocol.
(2) $\tilde{P}_b$ sends $(\mathsf{init}, \mathcal{C}_{\tilde{f}})$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$. Moreover, the simulator sends $(\mathsf{init}, \mathcal{C}_f)$ to $\mathcal{F}_{\text{SFE}}$.
(3) $\tilde{P}_b$ follows Step $1-3$ of the protocol normally.
(4) In Step 4 of the simulated protocol, extract the inputs that P_a is sending to $\mathcal{F}_{\text{SFE\&CommitOT}}$. Save these values as $o_a, s_{a,1}$ locally. Moreover, let y be a default input for the $\tilde{P}_b$, which $\tilde{P}_b$ uses in Step 4 of the protocol.

Function sampling:
(1) $\tilde{P}_b$ behaves like in the protocol.

Privacy-free phase:
(1) Run Step $1-5$ of the protocol. During Step 1 extract the values that P_a inputs into the $\mathcal{F}_{\text{OT}}$ functionality as x and $s_{a,2}$.

Check phase:
(1) Run Step $1-2$ of the protocol.
(2) In Step 3 compute the keys that P_a should have obtained based on $s_{a,2}, h_t, x$. If P_a opens commitments to different keys, then abort.
(3) In Step 4 follow the protocol normally.

Computation phase:
(1) Run Step $1, 2$ of the protocol, with the following restriction:
  - If $o_a \neq g_a(x)$ where o_a, x are the extracted values above and $g_a(x)$ is the function evaluated on the extracted input, then abort. Also abort if $s_{a,1} \neq s_{a,2}$.
  - If no abort (also not from $\tilde{P}_b$) happened, then send (inputA, x) to $\mathcal{F}_{\text{SFE}}$.

Fig. 7: The simulator for a malicious P_a.

Define the following hybrid distributions:

$\mathcal{T}_{P_a Hybrid1}$ which is obtained from using the simulator $\mathcal{S}_A$ with the following change: In the **Computation phase**, abort in Step 2 only if the output z of $\mathcal{F}_{\text{SFE}}$ would be $z = \perp$, i.e. if the hash function does not detect a differing input.

$\mathcal{T}_{P_a Hybrid2}$ which is obtained from using the simulator generating $\mathcal{T}_{P_a Hybrid1}$ with the following change: In the **Check phase**, do only abort if $\tilde{P}_b$ would abort instead of aborting if P_a opens commitments to wrong, but still valid keys.

Consider the distributions $\mathcal{T}_{P_a Sim}$ and $\mathcal{T}_{P_a Hybrid1}$, then the only difference lies in the outputs when P_a is cheating. In the first case, P_a will always be caught cheating whereas in the second case, she gets away with it as long as $\hat{f}'$ does not output $\perp$. There are three different events to consider:

(1) $o_a = g_a(x)$, but $s_{a,1} \neq s_{a,2}$: In this case, both $o_a, g_a(x)$ hash to the same value, hence $h_t(o_a) \oplus s_{a,1} \neq h_t(g_a(x)) \oplus s_{a,2}$ which will always be detected by $\hat{f}'$, so the success probability is 0.

(2) $o_a \neq g_a(x)$, but $s_{a,1} = s_{a,2}$: Since both $o_a, g_a(x)$ are independent of h_t and since h_t is chosen uniformly at random from the family $\mathbb{H}$, by Remark 1 they will collide with probability 2^{-s}, which is negligible in s.

(3) $o_a \neq g_a(x)$ and $s_{a,1} \neq s_{a,2}$: $\mathcal{F}_{\text{SFE\&CommitOT}}$ will not output $\perp$ iff $h_t(o_a) \oplus s_{a,1} = h_t(g_a(x)) \oplus s_{a,2}$. Hence it must hold that

$$h_t(o_a) \oplus h_t(g_a(x)) = s_{a,1} \oplus s_{a,2} = c$$

and a succeeding P_a will have to fix this c before learning h_t. By Remark 1 the success in doing so is 2^{-s} due to the uniform difference property and therefore negligible in s.

We hence conclude that $\mathcal{T}_{P_a Sim} \approx_s \mathcal{T}_{P_a Hybrid1}$. For the difference of $\mathcal{T}_{P_a Hybrid1}$ and $\mathcal{T}_{P_a Hybrid2}$, the simulator aborts in the first case if P_a commits to the wrong values, whereas it aborts in $\mathcal{T}_{P_a Hybrid2}$ if P_a provides strings that are not valid output keys of $\mathcal{G}$. By assumption, $\mathcal{G}$ provides *Correctness and Authenticity*, meaning that if P_a does not cheat, then she will obtain the correct keys and $\tilde{P}_b$ will continue. On the other hand, she can succeed in providing wrong keys only with probability $\mathsf{negl}(\lambda)$. Therefore, we also obtain that $\mathcal{T}_{P_a Hybrid1} \approx_c \mathcal{T}_{P_a Hybrid2}$.
Now consider the distributions $\mathcal{T}_{P_a Hybrid2}, \mathcal{T}_{P_a Real}$. The output that is delivered to $\mathcal{Z}$ as the output of P_b is the same in both distributions, so we focus on the messages that P_a obtains. The only difference between those is that in the **Check phase**, Step 4 these depend on a fixed input in $\mathcal{T}_{P_a Hybrid2}$ and on the real input of P_b in $\mathcal{T}_{P_a Real}$. In both cases, these keys correspond to values that are uniformly random to P_a since they are obtained by XOR-ing a uniformly random value s_b to $g_b(x)$ if P_a sent a correct garbling. Assume that F_b was not generated by $\mathcal{G}$, but instead chosen arbitrarily by the adversary. Then the output wires may leak some information about the inputs. In Step 1 of the **Check phase** the garbling F_b was verified and by the *Verifiability* of the garbling scheme $\mathcal{G}$ the computed output keys only depend on the output of the function except with probability negligible in λ. For every fixed output g'_b of the circuit and for every y there exists at least one s_b to obtain g'_b from y, and therefore the opened keys differ only with probability $\mathsf{negl}(\lambda)$. Hence $\mathcal{T}_{P_a Hybrid2} \approx_c \mathcal{T}_{P_a Real}$ which proves the statement for a malicious P_a.

Proof for malicious P_b. The proof of security for a malicious P_b goes along the same lines as the proof for P_a. We define the following hybrid distribution:

$\mathcal{T}_{P_b Hybrid}$ which is obtained from using the simulator generating $\mathcal{T}_{P_b Sim}$ with the following change: In the **Check phase**, do only abort if $\tilde{P}_a$ aborts instead of aborting if P_b opens commitments to wrong, but still valid keys.

By the same reasoning as before, we obtain that $\mathcal{T}_{P_b Sim} \approx_s \mathcal{T}_{P_b Hybrid}$. Because committed OT is available from $\tilde{P}_a$ to P_b, we do not have to cope with different values for s_b. In the step between $\mathcal{T}_{P_b Hybrid}$ and $\mathcal{T}_{P_b Real}$, we observe that in both cases, P_b obtains $\perp$ iff the values related to the keys $(g_b^{i'})_{i \in [l_b]}$ do not match $g_b(y) \oplus s_b$ for the extracted values y, s_b so the distributions of the output value z are identical. Moreover, by the same argument as before, the keys $(g_a^{i'})_{i \in [s]}$ do only reveal the value g'_a except with probability negligible in λ due to the *Verifiability* of $\mathcal{G}$ (the keys do reveal no information because s_a was chosen uniformly at random). Therefore $\mathcal{T}_{P_b Hybrid} \approx_c \mathcal{T}_{P_b Real}$, which completes the proof. $\square$

Simulator $\mathcal{S}_B$

Input phase:

(1) Start a copy of $\mathcal{F}_{\text{SFE\&CommitOT}}$ with which $\tilde{P}_a, P_b$ will communicate.

(2) $\tilde{P}_a$ sends $(\mathsf{init}, \mathcal{C}_{\hat{f}})$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$. Moreover, the simulator sends $(\mathsf{init}, \mathcal{C}_f)$ to $\mathcal{F}_{\text{SFE}}$.

(3) $\tilde{P}_a$ follows Step $1-3$ of the protocol normally.

(4) In Step 4, extract the inputs that P_b is sending to $\mathcal{F}_{\text{SFE\&CommitOT}}$. Save these values as o_b, s_b locally. Moreover, let x be a default input value for $\tilde{P}_a$. $\tilde{P}_a$ then performs Step 4 honestly with the default input.

Function sampling:

(1) $\tilde{P}_a$ follows Step $1, 2$ of the protocol normally.

Privacy-free phase:

(1) $\tilde{P}_a$ follows Step $1-5$ of the protocol normally.

Check phase:

(1) $\tilde{P}_a$ follows Step $1-3$ of the protocol normally.

(2) In Step 4 compute the keys that P_b should have obtained based on s_b, y. If P_b opens commitments to different keys, then abort.

Computation phase:

(1) Run Step $1, 2$ of the protocol, with the following restriction:
   - If $g'_b(y) \neq g_b(y)$ where $g'_b(y), y$ are the extracted values above and $g_b(y)$ is the function evaluated on the extracted input, then abort.
   - If no abort (also not from $\tilde{P}_a$)s happened, then send (inputB, y) to $\mathcal{F}_{\text{SFE}}$. Upon (output, z) from $\mathcal{F}_{\text{SFE}}$, send (output, z) to P_b.

Fig. 8: The simulator for a malicious P_b.

5 Optimizations

We will now discuss how the overhead from the protocol presented in Section 3 can be reduced. In particular, our construction requires more rounds of interaction and some computational overhead for securely computing the hash function and the committed OT for P_b. We will show that, by making non-trivial use of the SFE protocol by Frederiksen et al. [5] (FJN14) one can avoid parts of these extra computations. Due to the complexity of FJN14, we will just sketch this solution without a proof of security.

A short overview over the FJN14 construction

In Section 1 we sketched how an SFE protocol based on garbled circuits generally works. The presented pattern introduces a number of problems (as mentioned in the introduction), which are addressed in FJN14 using techniques which we will discuss now. We only focus on those techniques that are important with respect to our protocol.

Consistency of P_b's inputs. If one uses standard OT during the above protocol, then P_b may ask for various input keys for different circuits. As an example, he could (for a subset of circuits) decide that the 5th wire shall be 1 whereas it will be 0 for the other instances. This may, depending on the computed function, leak information about P_a's input. To thwart this attack, FJN14 performs OT for longer strings, where all zero- or one-keys for a certain input wire for all circuits will be obtained in one iteration[4].

Consistency of P_a's inputs. Similarly to P_b, also P_a can send different input keys for the instances. A solution similar to the above for P_b does not work, since P_b will then learn P_a's inputs. Instead, one lets P_a commit to her input keys ahead of time. P_b chooses a message digest function from $\mathbb{H}$ and P_a will garble the circuits such that they also compute a digest of her inputs. P_b checks during the evaluation that the hash

[4] To the best of our knowledge, a similar idea was first introduced in [17].

value is the same for all evaluated circuits, and aborts if not. To prevent leakage of information about P_a's input, P_a will *mask the hash* with a fixed string[5].

Using the FJN14 construction with our protocol

Using the OT of FJN14. Let P_b obtain the input keys for the privacy-free circuit *together with the input keys of the actively-secure garbling*, by also including these keys for s_b in the same OT. We therefore have to transfer an only slightly longer string for each input wire related to s_b[6].

Evaluating the Hash in the SFE for free. In the actively secure protocol P_b will choose the hash function for the consistency check. We can let this be the same hash function that is used in our protocol with the same random padding s_a. This means that we will use a lightweight version of our suggested $\hat{f}'$ function that only checks for consistency of P_b's input, while P_a's consistency is implicitly checked during the evaluation of the actively secure protocol. Note that in the case of a cheating P_a the protocol will then be aborted before the actual output is computed by P_b. Therefore, P_a must send her input keys for FJN14 and must have obtained her keys for the privacy-free garbling *before* h_t is revealed to her.

Public inputs. An approach to implement public inputs is to let the SFE protocol have a *second input phase* where P_a can submit the keys for the public inputs. Like in the FJN14 protocol, the input keys will be linked to a polynomial (whose evaluations are linked to either the 0-keys or 1-keys for each wire i) which is of degree $s/2$. Before the evaluation, P_b checks that all such points for the keys lie on the same polynomial (using the already opened circuits and keys from the cut-and-choose phase as well as the newly obtained keys). Now P_b can identify to which wire the keys sent by P_a belong by taking one of the submitted keys for both the $0, 1$-wires, interpolating the polynomial and checking whether all other keys belong to the polynomial that is linked to the correct bit of the publicly chosen input. We require that these public input keys, the polynomials and the links are generated by P_a during the garbling phase. They are sampled the same way as in the original protocol, and P_a is committed to the keys.

Acknowledgements

We want to thank Ivan Damgård and Tore Frederiksen for helpful discussions.

References

1. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology-EUROCRYPT 2014*, volume 8441, page 387. Springer, 2014.
2. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Automata, Languages and Programming*, pages 152–163. Springer, 2010.
3. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.
4. Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology—CRYPT0'95*, pages 110–123. Springer, 1995.
5. Tore Kasper Frederiksen, Thomas P Jakobsen, and Jesper Buus Nielsen. Faster maliciously secure two-party computation using the gpu. In *Security and Cryptography for Networks*, pages 358–379. Springer, 2014.
6. Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the gpu. Cryptology ePrint Archive, Report 2013/046, 2013. http://eprint.iacr.org/.

[5] We used the same technique, but for a different reason, in Π_{SIREVAL}. It was first introduced in the context of SFE with garbled circuits in [6,18].

[6] This means that we have to change the function $g'_b(\cdot,\cdot)$ slightly, due to a technique that avoids selective failure-attacks in FJN14. This change does not increase the size of the privacy-free circuit that is sent, since only XOR gates are added.

7. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology-EUROCRYPT 2015*, pages 191–219. Springer, 2015.
8. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.
9. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. Cryptology ePrint Archive, Report 2016/163, 2016. http://eprint.iacr.org/.
10. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.
11. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304. ACM, 1985.
12. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 294–304. IEEE, 2000.
13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 955–966. ACM, 2013.
14. Seny Kamara and Lei Wei. Garbled circuits via structured encryption. In *Financial Cryptography and Data Security*, pages 177–188. Springer, 2013.
15. Jonathan Katz, Alex J. Malozemoff, and Xiao Wang. Efficiently enforcing input validity in secure two-party computation. Cryptology ePrint Archive, Report 2016/184, 2016. http://eprint.iacr.org/2016/184.
16. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology–CRYPTO 2013*, pages 1–17. Springer, 2013.
17. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology-EUROCRYPT 2007*, pages 52–78. Springer, 2007.
18. Chih-hao Shen and abhi shelat. Fast two-party secure computation with minimal assumptions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 523–534. ACM, 2013.
19. Stefan Tillich and Nigel Smart. Circuits of basic functions suitable for mpc and fhe. Available at https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/, accessed on June 25th 2016.
20. Andrew C Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.