On Garbling Schemes With And Without Privacy

Carsten Baum*

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

Abstract. In recent years, a lot of progress has been made on speeding up *Actively-secure Two-party Function Evaluation* (SFE) using *Garbled Circuits*. For a given level of security, the amount of information that has to be sent and evaluated has been drastically reduced due to approaches that optimize the garbling method for the gates (such as *Free-XOR* [17], *Flex-OR* [16] and *Half-Gates* [26]). Moreover, the total number of garbled circuits sent to the evaluator dropped by a factor of 3, mostly due to the *Forge-and-Lose*-technique [3,18,12].
In another line of work, Frederiksen et al. [7] introduced the first *special purpose* garbling technique, namely a garbling scheme without privacy guarantees. In this note, we present an approach to combine such a privacy-free garbling scheme with an arbitrary SFE protocol for a certain class of circuits, such that the overall protocol is actively secure. This then yields a SFE protocol that has a smaller overhead in total. We instantiate our approach with the SFE protocol by [5] and show that the combination of both allows saving substantial amounts of network bandwidth for certain classes of circuits.

1 Introduction

Background. Actively-secure Two-party Function Evaluation denotes a problem in cryptography where two mutually distrusting parties Alice and Bob (P_A, P_B) want to jointly evaluate a function h based on secret inputs x, y that they choose individually. This is done using an interactive protocol over a network such that, at the end of the protocol, both parties only learned the correct output $z = h(x, y)$ of the computation and no other information. This restriction shall also hold if one of the parties arbitrarily deviates from the protocol, i.e. sends messages that it is not supposed to send according to the protocol. The problem was originally stated by Yao in 1982 [24,25] and solved for the case that no party deviates from the protocol, but tries to *infer additional information* from the transcript of the computation. This is the so-called *semi-honest* setting, whereas arbitrarily deviating adversaries are referred to as *malicious*.
Let us first describe an *ideal world* solution towards solving the above problem: Here, we assume that both P_A and P_B can send their inputs as well as a description of h which we call $\mathcal{C}_h$ to a trusted third party $\mathcal{T}$. This trusted third party would then do the following: We consider $\mathcal{C}_h$ to be a boolean circuit with dedicated input wires and output wires. $\mathcal{C}_h$ consists of gates having two input wires and one output wire. Each gate computes a function of its input wires and assigns the result to its output wire. $\mathcal{T}$ 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 $\mathcal{C}_h$ are either 0 or 1. Afterwards, $\mathcal{T}$ translates the values on the output wires into z and sends it to both P_A, P_B. What Yao showed in his seminal work was how to replace this $\mathcal{T}$ with an interactive protocol, which is nowadays known as garbled circuits.

Garbled circuits in a nutshell. In order to obtain a garbled circuit from $\mathcal{C}_h$, P_A (also known as *the Garbler*) performs the following operations: Each gate of the circuit can be represented as a table, where for each combination of the inputs (being 0 or 1) an output from $\{0, 1\}$ will be assigned to the output wire

* The author acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation (CTIC) and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council within which part of this work was performed.

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

(this table represents the function being computed by that specific gate). Now, the rows of this table are first shuffled and then the $0, 1$ values are replaced with 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 $\mathcal{C}_h$ and if they both correspond to the same value 0 or[2] 1. One then stores information such that an 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 on 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, and to which value $0, 1$ they belong.

After this *garbling* step is done, P_A sends the garbled gates and the input keys corresponding to her chosen input to *the Evaluator* P_B. He obtains these 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. With all this information, P_B can now evaluate the circuit gate by gate so that he obtains the output of the computation, which he then sends to P_A. Intuitively, the security of the protocol is based on the OT hiding P_Bs input while the garbling hides both the inputs of both parties and the circuit that is actually evaluated.

If P_A actively deviates from the above protocol, then she can cheat in multiple ways: Since the circuit that is computed is hidden from P_B, he can not be sure that the function it computes is the same that he and P_A agreed upon to compute (or that he obtains input keys that correspond to his inputs). This way, information might leak to P_A that was not intended to leak. 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 (so 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. In the end, P_B will take the majority of the outputs as the result of the computation. This approach introduces new problems, such as the *consistency of inputs* over multiple instances or *selective failure attacks*. A thorough proof of security of the protocol sketched above can be found in [20].

Garbling schemes. The garbled circuits-approach was initially thought of only to be a technique for SFE, but later has found multiple applications such as in verifiable computation [8], private set intersection [11], zero-knowledge proofs [14] or functional encryption with public keys [22] (to just name a few). Moreover, it has been treated on a more abstract level e.g. in [13] as *Randomized Encodings*. Kamara & Wei [15] 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. [2] discussed garbling as a primitive having potentially different security notions, and studied how these are related. Their framework makes it possible to compare different properties that a garbling scheme can have such as *privacy, authenticity and obliviousness*. This then allows to look for special schemes that may only implement a subset or rather different properties, that 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. [14] where the prover evaluates a garbled circuit in order to prove a certain statement.

Since only the evaluator in [14] has private inputs to the circuit and in turn then evaluates it on known values, the privacy guarantee of e.g. the original Yao garbling is not necessary. One in turn hopes that a garbling scheme without privacy can be constructed with less overhead. Frederiksen et al. [7] showed that one can in fact construct such a tailored scheme.

The problem. From the above exposition, the following question arises:

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

[2] If a wire is 0 or 1 in different rows, then the key for this wire will be the same.

Our idea can be thought of as a generalization of [14]: Those parts of a circuit $\mathcal{C}_h$ that do only depend on one party's input may not need to be computed with active security (because the party knows the outputs already). Such cases can occur e.g. when a part of the circuit must correctly compute a data structure (think of sorting values) or if statements about inputs must be validated. While the idea seems intuitive, 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.

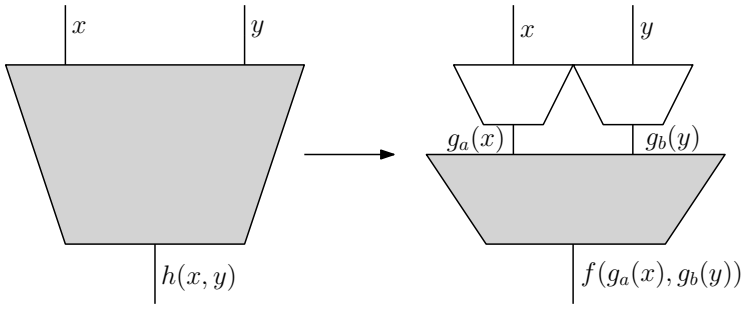

Fig. 1: A graphical depiction about the functions to which our approach can be applied.

Contributions. In this note, 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. For such functions it can then potentially improve the runtime of SFE.
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 (the gray circuit) will be computed with active security. For the sake of simplicity, our solution allows that the evaluation of f can be done by an arbitrary SFE scheme.
To achieve this goal, we use circuit augmentation for g_a, g_b, f which in itself introduces a small overhead. We will show that this overhead can mostly be eliminated using certain SFE protocols.
Intuitively, we start with the following idea:

(1) Let P_A compute a privacy-free garbling of g_b and P_B compute a privacy-free garbling of g_a.
(2) Both parties exchange and evaluate the privacy-free garbling, whose output in turn will be the input to the evaluation of f.

Here, 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 f. At the same time, one must also keep the outputs of g_a, g_b confidential and prevent the garbler from sending an incorrect circuit or wrong input keys. Our solution will deal with the inconsistency problem by comparing that the inputs to 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.

Related work. Our problem shares some similarity with the area of *Verifiable Computation* [8,1,21]. 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. Based upon this proof, the client can then decide whether the computation was done correctly or not. A crucial requirement is that the verification of the proof shall be computationally less expensive than the computation itself. Our setting is different from verifiable computation, since we want that the server performs the evaluation of the circuit on his own inputs correctly, and these must be kept secret. Moreover,

we do only require one evaluation of the circuit while in verifiable computation, the same preprocessed data may be used in multiple instances.
Our solution, as already mentioned above, also bears resemblance with the concept of *Zero-Knowledge Proofs* [10,9] 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 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 be used to (efficiently) show that the prover knows a specific input to a circuit was already observed in [14]. In comparison to their work, we exploit this phenomenon in a more general sense and not just for zero-knowledge proofs.

2 Preliminaries

Let us assume that P_A, P_B agreed to evaluate a function $h : \{0,1\}^{2n} \to \{0,1\}^m$, 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 $f : \{0,1\}^{p+q} \to \{0,1\}^m$, $g_a : \{0,1\}^n \to \{0,1\}^p$, $g_b : \{0,1\}^n \to \{0,1\}^q$ such that

$$\forall x, y \in \{0,1\}^n : \ f(g_a(x), g_b(y)) = h(x, y)$$

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 [2]. We can then go on to make our above definition more formal, but based on a boolean circuit-level.

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 $Inputs = \{1, ..., n_{in}\}$, $Wires = \{1, ..., n_w\}$, $Outputs = \{n_w - n_{out} + 1, ..., n_w\}$ and $Gates = \{n_{in} + 1, ..., n_w\}$ 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 $G : Gates \times \{0,1\}^2 \mapsto \{0,1\}$ 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 $f : \{0,1\}^{n_{in}} \mapsto \{0,1\}^{n_{out}}$, we consider $\mathcal{C}_f = (n_{in}, n_{out}, n_g, L, R, G)$ as a *circuit representation of* f iff $\forall x \in \{0,1\}^{n_{in}} : \ f(x) = eval(\mathcal{C}_f, x)$.

In order to be able to apply our solution, the circuit in question must be decomposable in a certain way as already outlined in the introduction. We will now formalize what we mean by this decomposability.

Definition 1 (Split-input representation (SIR)). *Let* $h : \{0,1\}^{2n} \to \{0,1\}^m, f : \{0,1\}^{p+q} \to \{0,1\}^m$, $g_a : \{0,1\}^n \to \{0,1\}^p$, $g_b : \{0,1\}^n \to \{0,1\}^q$ *be functions such that*

$$\forall x, y \in \{0,1\}^n : \ f(g_a(x), g_b(y)) = h(x, y)$$

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

For every function h with $n \geq 2$ such a decomposition always exists, but it only of interest in our setting if (intuitively) $\mathcal{C}_f$ is has a lot less gates than $\mathcal{C}_h$.

2.2 Secure two-party computation and garbling schemes

The notion of an SFE protocol was already intuitively introduced in Section 1. Formally, it describes a protocol between two parties P_A, P_B that securely implements Figure 2.

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

Initialization:
- On input $(\mathsf{init}, \mathcal{C}, s, I_1, ..., I_j)$ from both P_A, P_B where $\mathcal{C} = (n, m, g, L, R, G)$ is a circuit and s the statistical security parameter, store $\mathcal{C}, s$. 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 $(\mathsf{cotB}, id, \{y_0^i, y_1^i\}_{i\in\{1,...,l\}})$ by P_A and if there is a (id, x, P_B) stored with $x = x_1...x_l$, then output $(\mathsf{ot}, \{y_{x_i}^i\}_{i\in\{1,...,l\}})$ 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 $(I_i, x, \cdot)$ stored for each $i = 1, ..., j$, 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 $z = eval(\mathcal{C}, x_1...x_j)$ and output (output, z) to P_B.

Fig. 2: Secure function evaluation, commitments and committed OT for two parties

Note that the functionality in Figure 2 moreover provides *commitments* and *committed OT* [4]. [3]. Commitments allow a party to lock himself to a certain value (without revealing it) at a point in time, such that he can later reveal this value (without being able to deviate). Committed OT resembles OT as mentioned before, but where the choice of the receiver is determined by a commitment. The main reason why we need this functionalities 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 separate functionalities introduces problems during the proof.
Out of the framework of [2] we will now recap the notion of *projective verifiable garbling schemes*. We require the properties *correctness, authenticity and verifiability* of our garbling scheme. These intuitively ensure that

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

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

OT for P_A:
- On input (otA, x) from P_A and $(\mathsf{otA}, \{y_0^i, y_1^i\}_{i\in\{1,\dots,l\}})$ by P_B and if $x = x_1...x_l$, output $(\mathsf{ot}, \{y_{x_i}^i\}_{i\in\{1,\dots,l\}})$ to P_A.

OT for P_B:
- On input (otB, x) from P_B and $(\mathsf{otB}, \{y_0^i, y_1^i\}_{i\in\{1,\dots,l\}})$ by P_A and if $x = x_1...x_l$, output $(\mathsf{ot}, \{y_{x_i}^i\}_{i\in\{1,\dots,l\}})$ to P_B.

Fig. 3: Functionality for oblivious transfer

the evaluated circuit 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. We will make this more formal now.
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 $n_{in}, n_{out} = poly(\lambda), n \geq \lambda$ and $|\mathcal{C}_f| = poly(\lambda)$ 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 $X_i = X_i^{x_i}$ 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 $z_i = b$ if $Z_i = Z_i^b$, and $z_i = \perp$ 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 of the function:

Definition 2 (Correctness). *Let $\mathcal{G}$ be a verifiable projective garbling scheme. Then $\mathcal{G}$ is correct if for all $n_{in}, n_{out} = poly(\lambda), f : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x \in \{0,1\}^{n_{in}}$ it holds that*

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

Authenticity is very important for our later application, since it prevents the adversary from outputting other output keys than those he can derive from the input keys and the garbling. This effectively binds the adversary to his input.

Definition 3 (Authenticity). *Let $\mathcal{G}$ be a verifiable projective garbling scheme. Then $\mathcal{G}$ provides authenticity if for all $n_{in}, n_{out} = poly(\lambda), f : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x \in \{0,1\}^{n_{in}}, y \in \{0,1\}^{n_{out}}$ with $y \neq f(x)$ it holds that*

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

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 $n_{in}, n_{out} = poly(\lambda), f : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$ with circuit representation $\mathcal{C}_f$ and for all $x, y \in \{0,1\}^{n_{in}}, x \neq y, f(x) = f(y)$ it holds that*

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

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

A garbling scheme $\mathcal{G}$ that fulfills 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. On a high level, 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 $\mathcal{H} = \{h : \{0,1\}^m \to \{0,1\}^s\}$, then $\mathcal{H}$ is a family of universal hash functions if*

$$\forall x, y \in \{0,1\}^m, x \neq y \ : \ Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] \leq 2^{-s}$$

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

$$\forall x, y \in \{0,1\}^m, x \neq y, \ \forall z \in \{0,1\}^s \ : \ Pr_{h \in_R \mathcal{H}}[h(x) \oplus h(y) = z] \leq 2^{-s}$$

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

Definition 6. *Let $\boldsymbol{b} \in \{0,1\}^{m+s-1}$ and $\boldsymbol{M} \in \{0,1\}^{s \times m}$ such that $\boldsymbol{M}_{i,j} = \boldsymbol{b}_{i+j-1}$ and define $h_{\boldsymbol{x}} : \boldsymbol{x} \mapsto \boldsymbol{M}\boldsymbol{x}$. Moreover, define the family $\mathbb{H}$ as $\mathbb{H} = \{h_{\boldsymbol{b}} \mid \boldsymbol{b} \in \{0,1\}^{m+s-1}\}$.*

Remark 1. $\mathbb{H}^{In}$ 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 f. To do so, we augment the computed circuits slightly. A graphical depiction of that can be found in Figure 4.
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 h, f, g_a, g_b be functions as defined before. To compute a *proof* that P_A, P_B computed g_a, g_b correctly, we will make the other party compute a *digest* on the output of the function. Therefore, we augment g_a with a universal hash function h_b 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_b is chosen, the inputs $g_a(x), s_a$ to f will differ from the output g'_a with high probability. We observe that b, g'_a can be public inputs to f'.
In the case of P_B, it is not necessary for him to compute an actual hash of g_b. This is because only P_A can arbitrarily send differing inputs for f by choosing different values that blind her input (whereas committed OT is available for P_B to circumvent this). Therefore, P_A will send a privacy-free garbling of the function

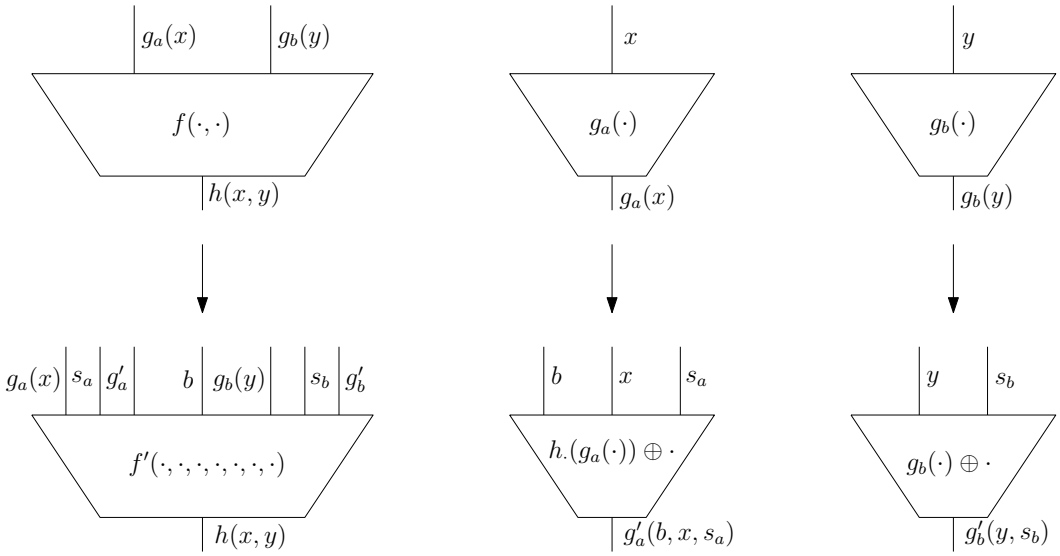

Fig. 4: The functions and how they will be augmented.

computing $g'_b(\cdot) = g_b(\cdot) \oplus s_b$ where s_b is a random value P_B committed to before. We can now once again make the value g'_b a public input to f'.
The actively secure protocol will now evaluate f on the inputs $g_a(x), g_b(y)$ as before. The correct value will only be output of f' if, given the auxiliary inputs s_a, s_b and the public inputs b, g'_a, g'_b it holds that $h_b(g_a(x)) \oplus s_a == g'_a$ and $g_b(y) \oplus s_b == g'_b$. Otherwise, an abort symbol $\perp$ will be delivered.
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 $h_b \in \mathbb{H}$ and sends its description b 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 use 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 b and thereby reveals the hash function h_b. 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 f' in the next step.

Computation phase P_A and P_B evaluate f' securely using SFE. The inputs are defined by the commitments from the input phase and the opened commitments from the check phase.

Based on the transformation that was outlined in Figure 4, the augmented circuits for the functions f', g'_a, g'_b can easily be obtained from the SIR of h. Moreover, these augmentations are small and, with respect to an implementation of SFE using garbled circuits, come almost for free using the Free-XOR technique.

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

4 Security

We will now prove the security of our protocol in Figure 5. More formally, consider the stripped-down functionality in Figure 6 which basically focuses on the SFE. Then we prove the following theorem:

Protocol Π_{SIREval}

Both parties P_A, P_B want to evaluate a function $h : \{0,1\}^{2n} \to \{0,1\}^m$ and we consider its SIR $\mathcal{C}_f, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$. P_A has input $x \in \{0,1\}^n$ and P_B has input $y \in \{0,1\}^n$.

Input phase:

(1) Let $\mathcal{C}'_f, \mathcal{C}'_{g_a}, \mathcal{C}'_{g_b}$ be circuits representing f', g'_a, g'_b which were defined before.
(2) Both parties send $(\mathsf{init}, \mathcal{C}'_f, s, "g_a(x)", "s_a", "g'_a", "h_b", "g_b(y)", "s_b", "g'_b")$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.
(3) P_A computes $g_a(x)$ locally and chooses $s_a \in_R \{0,1\}^s$. P_B computes $g_b(y)$ locally and chooses $b \in_R \{0,1\}^{p+s-1}$, $s_b \in_R \{0,1\}^q$.
(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 $(\mathsf{commit}, "y", y)$, $(\mathsf{commit}, "g_b(y)", g_b(y)), (\mathsf{commit}, "s_b", s_b), (\mathsf{commit}, "h_b", b)$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.

Function sampling:

(1) P_A computes $(F_b, \{y_0^i, y_1^i\}_{i\in1,\ldots,n}\{s_{0,b}^i, s_{1,b}^i\}_{i\in1,\ldots,q}, d_b) \leftarrow Gb(1^s, \mathcal{C}'_{g_b})$ and sends F_b to P_B.
(2) Likewise, P_B computes $(F_a, \{x_0^i, x_1^i\}_{i\in1,\ldots,n}\{s_{0,a}^i, s_{1,a}^i\}_{i\in1,\ldots,s}, d_a) \leftarrow Gb(1^s, \mathcal{C}'_{g_a})$ and sends F_a to P_A.

Privacy-free phase:

(1) P_A sends (otA, x) and P_B sends $(\mathsf{otA}, \{x_0^i, x_1^i\}_{i\in1,\ldots,n})$ to $\mathcal{F}_{\text{OT}}$, hence P_A obtains $\{x^i\}_{i\in1,\ldots,n}$. They do the same for "s_a" so P_A obtains $\{s_a^i\}_{i\in1,\ldots,s}$.
(2) Conversely, P_B sends $(\mathsf{cotB}, "y")$ and P_A sends $(\mathsf{cotB}, "y", \{y_0^i, y_1^i\}_{i\in1,\ldots,n})$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$, hence P_B obtains $\{y^i\}_{i\in1,\ldots,n}$. They do the same for "s_b" so P_B obtains $\{s_b^i\}_{i\in1,\ldots,q}$.
(3) P_B sends $(\mathsf{open}, "h_b")$ to $\mathcal{F}_{\text{SFE\&CommitOT}}$.
(4) P_A evaluates the privacy-free garbling as $(g_a^{i\prime})_{i\in1,\ldots,s} \leftarrow Ev(F_a, \{x^i\}_{i\in1,\ldots,n}\{s_a^i\}_{i\in1,\ldots,s}, xs_a)$ and then commits to $(g_a^{i\prime})_{i\in1,\ldots,s}$.
(5) P_B evaluates the privacy-free garbling as $(g_b^{i\prime})_{i\in1,\ldots,q} \leftarrow Ev(F_b, \{y^i\}_{i\in1,\ldots,n}\{s_b^i\}_{i\in1,\ldots,q}, ys_b)$ and then commits to $(g_b^{i\prime})_{i\in1,\ldots,q}$.

Check phase:

(1) P_A sends $(F_b, \{y_0^i, y_1^i\}_{i\in1,\ldots,n}\{s_{0,b}^i, s_{1,b}^i\}_{i\in1,\ldots,q}, d_b)$ to P_B who checks that he obtained correct input and output keys and that $Ve(\mathcal{C}'_{g_b}, F_b, \{y_0^i, y_1^i\}_{i\in1,\ldots,n}\{s_{0,b}^i, s_{1,b}^i\}_{i\in1,\ldots,q}) = 1$. If not, then P_B aborts.
(2) P_B sends $(F_a, \{x_0^i, x_1^i\}_{i\in1,\ldots,n}\{s_{0,a}^i, s_{1,a}^i\}_{i\in1,\ldots,s}, d_a)$ to P_A who checks that she obtained correct input and output keys and that $Ve(\mathcal{C}'_{g_a}, F_a, \{x_0^i, x_1^i\}_{i\in1,\ldots,n}\{s_{0,a}^i, s_{1,a}^i\}_{i\in1,\ldots,s}) = 1$. If not, then she aborts.
(3) P_A opens her commitments to $(g_a^{i\prime})_{i\in1,\ldots,s}$. P_B computes $g'_a = De((g_a^{i\prime})_{i\in1,\ldots,s}, d_a)$ 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 $(g_b^{i\prime})_{i\in1,\ldots,q}$. P_A computes $g'_b = De((g_b^{i\prime})_{i\in1,\ldots,q}, d_b)$ 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

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

Initialization:

– On input $(\mathsf{init}, \mathcal{C}, s)$ from both P_A, P_B where $\mathcal{C} = (2n, m, g, L, R, G)$ is a circuit and s the statistical security parameter, store $\mathcal{C}, s$.

Input of P_A:

– Upon input (inputA, x) from P_A where $x \in \{0,1\}^n$ 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 $y \in \{0,1\}^n$ and where no input was given by P_B before and if (inputA) was obtained by P_B, compute $z = eval(\mathcal{C}, xy)$ and output z to P_B.

Fig. 6: Secure function evaluation, commitments and committed OT for two parties

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 intercepts all the commitments coming from P_A and simulates an honest P_B. 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. We consider two distributions to be indistinguishable if their distance is negligible in either s or λ, where $\approx_c$ means computational and $\approx_s$ statistical indistinguishability.

Proof. As in the protocol Π_{SIREVAL} we assume that both parties P_A, P_B want to evaluate a function $h : \{0,1\}^{2n} \to \{0,1\}^m$ and we consider its SIR $\mathcal{C}_f, \mathcal{C}_{g_a}, \mathcal{C}_{g_b}$. P_A has input $x \in \{0,1\}^n$ and P_B has input $y \in \{0,1\}^n$.

Proof for malicious P_A:
We first show a simulator $\mathcal{S}_A$ to prove that from P_As perspective, $\mathcal{F}_{\text{SFE}} \diamond \mathcal{S}_A \approx \mathcal{F}_{\text{SFE\&COMMITOT}} \diamond \Pi_{\text{SIREVAL}}$.

Simulator $\mathcal{S}_A$

Input phase:
(1) Start a local version of $\mathcal{F}_{\text{SFE\&COMMITOT}}$ with which P_A will communicate.
(2) Send $(\mathsf{init}, \mathcal{C}_f, s)$ for both P_A, P_B to $\mathcal{F}_{\text{SFE\&COMMITOT}}$. Moreover, send $(\mathsf{init}, \mathcal{C}_h, s)$ to $\mathcal{F}_{\text{SFE}}$.
(3) Follow Step $1-3$ of the protocol.
(4) In Step 4, extract the inputs that P_A is sending to $\mathcal{F}_{\text{SFE\&COMMITOT}}$. Save these values as $g_a(x)', s_{a,1}$ locally. Moreover, let y be a default input for the simulated P_B. Compute $g_b(y), s_b, h_b$ as in the protocol and send $(\mathsf{commit}, "y", y), (\mathsf{commit}, "g_b(y)", g_b(y)), (\mathsf{commit}, "s_b", s_b), (\mathsf{commit}, "h_b", b)$ in the name of P_B to $\mathcal{F}_{\text{SFE\&COMMITOT}}$

Function sampling:
(1) Simulate Step $1, 2$ as in the protocol.

Privacy-free phase:
(1) Simulate Step $1-5$ as in 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) Simulate Step $1-2$ as in the protocol.
(2) In Step 3 compute the keys that P_A should have obtained based on $s_{a,2}, h_b, x$. If P_A opens commitments to different keys, then abort.
(3) In Step 4 follow the protocol.

Computation phase:
(1) Follow Step $1, 2$ in the protocol, with the following restriction:
- If $g_a(x)' \neq g_a(x)$ where $g_a(x)', x$ 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 happened, then send (inputA, x) to $\mathcal{F}_{\text{SFE}}$.

Fig. 7: A simulator for a malicious P_A

Let $\mathcal{T}_{P_A Real}$ be the distribution of the transcripts that are obtained by executing Π_{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 $\mathcal{T}_{P_A Real} \approx \mathcal{T}_{P_A Sim}$. 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 P_B would abort instead of aborting if P_A opens commitments to wrong, but still valid keys.

Simulator $\mathcal{S}_B$

Input phase:

(1) Start a local version of $\mathcal{F}_{\text{SFE\&COMMITOT}}$ with which P_B will communicate.
(2) Send $(\mathsf{init}, \mathcal{C}_f, s)$ for both P_A, P_B to $\mathcal{F}_{\text{SFE\&COMMITOT}}$. Moreover, send $(\mathsf{init}, \mathcal{C}_h, s)$ to $\mathcal{F}_{\text{SFE}}$.
(3) Follow Step $1-3$ of the protocol.
(4) In Step 4, extract the inputs that P_B is sending to $\mathcal{F}_{\text{SFE\&COMMITOT}}$. Save these values as $g_b(y)', s_b$ locally. Moreover, let x be a default input value for P_A. Compute $g_a(x), s_a$ as in the protocol and send $(\mathsf{commit}, "x", x), (\mathsf{commit}, "g_a(x)", g_a(x)), (\mathsf{commit}, "s_a", s_a)$ in the name of P_A to $\mathcal{F}_{\text{SFE\&COMMITOT}}$

Function sampling:

(1) Simulate Step $1, 2$ as in the protocol.

Privacy-free phase:

(1) Simulate Step $1-5$ as in the protocol.

Check phase:

(1) Simulate Step $1-3$ as in the protocol.
(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) Follow Step $1, 2$ in 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 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: A simulator for a malicious P_B

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 f' does not output $\perp$. There are three mutually different events to consider:

(1) $g_a(x)' = g_a(x)$, but $s_{a,1} \neq s_{a,2}$: In this case, both $g_a(x)', g_a(x)$ hash to the same value, hence $h_b(g_a(x)') \oplus s_{a,1} \neq h_b(g_a(x)) \oplus s_{a,2}$ which will always be detected by f', so the success probability is 0.
(2) $g_a(x)' \neq g_a(x)$, but $s_{a,1} = s_{a,2}$: Since both $g_a(x)', g_a(x)$ are independent of h_b and since h_b 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) $g_a(x)' \neq g_a(x)$ and $s_{a,1} \neq s_{a,2}$: $\mathcal{F}_{\text{SFE\&COMMITOT}}$ will not output $\perp$ iff $h_b(g_a(x)') \oplus s_{a,1} = h_b(g_a(x)) \oplus s_{a,2}$. Hence it must hold that
$$h_b(g_a(x)') \oplus h_b(g_a(x)) = s_{a,1} \oplus s_{a,2} = c$$
and a succeeding P_A will have to fix this c before learning h_b. 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 the simulated P_B will continue. On the other hand, she can succeed in providing wrong keys only with probability $negl(\lambda)$. 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 $negl(\lambda)$. 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 P_A would abort 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 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 $(g_b^{i\prime})_{i\in 1,\ldots,q}$ 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 $(g_a^{i\prime})_{i\in 1,\ldots,s}$ 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. □

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. We describe those concepts of the FJN14 protocol that we will exploit in a non-black box way in this section. Familiarity with the protocol may ease understanding.

A short overview over the FJN14 construction

As mentioned already in Section 1, an SFE protocol based on garbled circuits generally works as follows:

(1) P_A garbles a number of circuits and sends them to P_B.
(2) P_B obtains his input keys from P_A using oblivious transfer.
(3) P_B chooses a random subset of circuits to be completely opened (he obtains all input keys) and checks whether these circuits are well-formed.
(4) P_A sends her inputs for the remaining circuits and P_B evaluates these.

This scheme introduces a number of problems, which are solved in FJN14 mostly using techniques which we will mention now. We only focus on those techniques that are important with respect to our protocol.

Consistency of P_Bs 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_As 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].

[4] To the best of our knowledge, a similar idea was first introduced in [19].

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

Using the FJN14 construction with our protocol

The above properties can be used to interleave our protocol from Section 3 with an FJN14 instance. This allows to save on network load or rounds of communication, which we will now show.

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 f' function that only checks for consistency of P_Bs input, while P_As 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_b 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[7] (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 would like to thank Ivan Damgård and Tore Frederiksen for helpful discussions.

References

1. 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.

[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 [6,23].

[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] The polynomials are sampled as in FJN14, meaning that a polynomial is chosen uniformly at random for the 0-key of each wire. Then its shift by the Free-XOR distance Δ of the specific garbled circuit yields the polynomial for the 1-keys.

2. 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.
3. Luís TAN Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique. In *Advances in Cryptology-ASIACRYPT 2013*, pages 441–463. Springer, 2013.
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/.
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. 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.
10. 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.
11. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
12. Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology–CRYPTO 2013*, pages 18–35. Springer, 2013.
13. 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.
14. 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.
15. Seny Kamara and Lei Wei. Garbled circuits via structured encryption. In *Financial Cryptography and Data Security*, pages 177–188. Springer, 2013.
16. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In *Advances in Cryptology–CRYPTO 2014*, pages 440–457. Springer, 2014.
17. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.
18. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology–CRYPTO 2013*, pages 1–17. Springer, 2013.
19. 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.
20. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.
21. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.
22. Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM, 2010.
23. 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.
24. Andrew C Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.
25. Andrew C Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.
26. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Advances in Cryptology-EUROCRYPT 2015*, pages 220–250. Springer, 2015.