Reactive Garbling: Foundation, Instantiation, Application

Jesper Buus Nielsen[⋆] and Samuel Ranellucci

Aarhus University

Abstract. Garbled circuits is a cryptographic technique, which has been used among other things for the construction of two and three-party secure computation, private function evaluation and secure outsourcing. Garbling schemes is a primitive which formalizes the syntax and security properties of garbled circuits. We define a generalization of garbling schemes called *reactive garbling schemes*. We consider functions and garbled functions taking multiple inputs and giving multiple outputs. Two garbled functions can be linked together: an encoded output of one garbled function can be transformed into an encoded input of the other garbled function without communication between the parties. Reactive garbling schemes also allow partial evaluation of garbled functions even when only some of the encoded inputs are provided. It is possible to further evaluate the linked garbled functions when more garbled inputs become available. It is also possible to later garble more functions and link them to the ongoing garbled evaluation. We provide rigorous definitions for reactive garbling schemes. We define a new notion of security for reactive garbling schemes called confidentiality. We provide both simulation based and indistinguishability based notions of security. We also show that the simulation based notion of security implies the indistinguishability based notion of security. We present an instantiation of reactive garbling schemes. We present an application of reactive garbling schemes to reactive two-party computation secure against a malicious, static adversary. We demonstrate how garbling schemes can be used to give abstract black-box descriptions and proof of several advanced applications of garbled circuits in the literature, including Minilego and Lindell's forge-and-lose technique.

1 Introduction

Garbled circuits is a technique originating in the work of Yao and later formalised by Bellare, Hoang and Rogaway [3], who introduced the notion of a garbling scheme along with an instantiation. Garbling schemes have found a wide range of applications. However, many of these applications are using specific constructions of garbled circuits instead of the abstract notion of a garbling scheme. One possible explanation is that the notion of a garbling scheme falls short of capturing many of the current uses. In the notion of a garbling scheme,

⋆ Supported by European Research Council Starting Grant 279447.

the constructed garbled function can only be used for a single evaluation and the garbled function has no further use. In contrast, many of the most interesting current applications of garbled circuits have a more granular look at garbling, where several components are garbled, dynamically glued together and possibly evaluated at different points in time. We now give a few examples of this.

In the standard cut-and-choose paradigm for two-party computation, Alice sends s copies of a garbled function to Bob. Half of the garblings (chosen by Bob) are opened to check that they were correctly constructed. This guarantees that the majority of the remaining instances were correctly constructed. Alice and Bob then use the remaining garblings for evaluation. Bob takes the majority output of these evaluations as his output. Although conceptually simple, this introduces a number of problems: Bob must ensure that Alice uses consistent inputs. It is also required that the probability that Bob aborts does not depend on his choice of input. Previous protocols solve these problems by doing white-box modifications of the underlying garbling scheme. We will show how to solve these problems by using reactive garbling schemes in a black-box manner.

In [20], Lindell presents a very efficient protocol for achieving active secure two-party computation from garbled circuits. In the scheme of Lindell, first s circuits are sent. Then a random subset of them are opened up to test that they were correctly constructed and the rest, the so-called evaluation circuits, are then evaluated in parallel. If the evaluations don't all give the same output, then the evaluator can construct a certificate of cheating which can be fed into a small corrective garbled circuit. Another example is a technique introduced simultaneously by Krater, shelat and Shen [18] and Frederiksen, Jakobsen and Nielsen [7], where a part of the circuit which checks the so-called input consistency of one of the parties is constructed *after* the main garbled circuit has been constructed and *after* Alice has given her input. We use a similar technique in our example application, showing that this trick can be applied to (reactive) garbling schemes in general. Another example is the work of Huang, Katz, Kolesnikov, Kumaresan and Malozemoff [16] on amortising garbled circuits, where one of the analytic challenges is a setting where many circuits are garbled prior to inputs being given. Our security notion allows this behaviour and this part of their protocol could therefore be cast as using a general (reactive) garbling scheme. Another example is the work of Huang, Evans, Katz and Malka [15] on fast secure two-party computation using garbled circuits, where they use pipelining: the circuit is garbled and evaluated in blocks for efficiency. Finally, we remark that sometimes the issue of garbling many circuits and gluing them together and having them interact with other security components can also lead to subtle insecurity problems, as demonstrated by the notion of a garbled RAM as introduced by Lu and Ostrovsky in [22], where the construction was later proven to be insecure by Gentry, Halevi, Lu, Ostrovsky, Raykova and Wichs [12]. We believe that having well founded abstract notions of partial garbling and gluing will make it harder to overlook security problems.

Our goal is to introduce a notion of reactive garbling schemes, which is general enough to capture the use of garbled circuits in most of the existing applications and which will hopefully form a foundation for many future applications of garbling schemes. Reactive garbling schemes generalize garbling schemes in several ways. First of all, we allow a garbled evaluation to save a state and use it in further computations. Specifically, when garbling a function f one can link it to a previous garbling of some function g and as a result get a garbling of $f \circ g$. Even more, given two independent garblings of f and g, it is possible to do a linking which will produce a garbling of $f \circ g$ or $g \circ f$. The linking depends only on the output encoding and input encoding of the linked garblings. We also allow garbling of a single function which allows partial evaluation and which allows dynamic input selection based on partial outputs. This can be mixed with linking, so that the choice of which functions to garble and link can be based on partial outputs. This can be important in *reactive* secure computation which allows inputs to arrive gradually and allows branching based on public partial outputs. We introduce the syntax and security definitions for this notion. We give an instantiation of reactive garbling schemes in the random oracle model. We also demonstrate the usefulness of reactive garbling schemes by giving various applications. We construct a reactive, maliciously UC secure two-party computation protocol. We also describe Lindell's reduced circuit optimization by using reactive garbling schemes. These two constructions use reactive garbling schemes in a black-box manner. We also describe the minilego garbling procedure as a reactive garbling scheme.

1.1 Discussion and Motivation

In this section, we describe the purpose of our framework and why certain design choices were made for the framework in this paper.

One of the main goals of garbling schemes was to define a primitive that would be used in constructions without relying on the underlying instantiation. Unfortunately, most secure two-party computation protocols still rely on garbled circuits to provide security. In some sense, the notion of garbling schemes is not able to achieve this goal for the given task. One way of thinking of our result is to note that many techniques that previously only worked for garbled circuits, now work for reactive garbling schemes.

More precisely, to achieve reactive secure computation, the protocol for reactive computation shows how three issues which typically are solved using the underlying instantiation of garbled circuits in cut-and-choose protocols can be solved using reactive garbling schemes. These issues are Alice's input consistency, selective failure attacks and how to run the simulator against a corrupted Bob. We solve these three issues by using the notion of reactive garbling schemes. This means that many protocols in the literature can easily be modified to achieve security by only relying on the properties of reactive garbling schemes.

We now discuss why certain design choices were made. In particular, why we included notions such as linking multiple output wires to a single input wire, partial evaluation and output encoding. The reason that we allow multiple output wires to link to a single input wire is that otherwise we would exclude important constructions such as Minilego [8] and Lindell's reduced circuit optimization [20].

Output encodings are important for many reasons. First, it provides a method for defining linking. Roughly because of this notion, it is easy to define a linking as information which allows an encoded output to be converted into an encoded input. Secondly, in certain cases, constructions based on garbling schemes require a special property of the encoded output which otherwise cannot be described. This is the case of [13] where the encoded input has to be the same size as the encoded output. It is also useful for output reuse, covers pipelining and has applications to protocols where the receiver can use a proof of cheating to extract the sender's input.

We included partial evaluation for two main reasons, first we consider that it can be an important feature for reactive computation, secure outsourcing and secure computation where a partial output would be valuable. A partial output could be used to determine what future computation to run on data. In addition, we could garble blocks of functions and decide to link certain blocks together based on partial outputs.

In addition, many schemes in the literature inherently allow partial evaluation and not allowing partial evaluation imposes artificial restrictions on the constructions. For example, fine-grained privacy in [2] cannot be realized by standard schemes precisely because those schemes give out partial outputs.

1.2 Recasting Previous Constructions

The concept of using output encoding and linking has been implicitly used in many previous works. In particular, in cut-and-choose protocols, it has been used in [6,7,19,25] to enforce sender input consistency (ensure that the sender uses the same input in each instance) and to prevent selective failure attacks (an attack that works by having the probability that the receiver aborts depend on his choice of input). These concepts have also been used for different optimizations. Pipelining [15,18] and output reuse [13,23] are examples of direct optimizations. Linking has also been employed to reduce the number of circuits that need to be sent in protocols that apply cut-and-choose at the circuit level [5,20]. This is done by adding a phase where a receiver can extract the input of a cheating sender. Another example is gate soldering [8,24]. This technique works by employing cut-and-choose at the gate level. The gates are then randomly split among different buckets and soldered together. This optimization reduces the replication factor for a security $\ominus(s)$ to $\ominus(\frac{s}{\log(n)})$ where n is the number of non-xor gates. There are many applications that benefit from output encoding and linking in garbling schemes. In addition, if we allow sequences where the input is chosen as a function of the garbling, reactive garbling schemes are also adaptive. The constructions of [14,11] require adaptive garbling.

1.3 Structure of the paper

In Section 2, we give the preliminaries. In Section 3, we define the syntax and security of reactive garbling schemes. In Section 4, we describe an instantiation of a reactive garbling scheme. In Section 4.1, we give a full description of the reactive garbling scheme. In Section 5, we give an intuitive description of the reactive two-party computation protocol based on reactive garbling schemes. In Section 5.1, we provide a full description of the reactive two-party computation protocol. We note that the techniques that we introduce in section 5 can be applied to previous secure two-party computation protocol to convert them into constructions that only use reactive garbling schemes in a black-box manner.

In Appendix A, we prove that our reactive computation protocol is secure. In Appendix B, we prove security of our reactive garbling scheme using the indistinguishability based notion of security. In Appendix C, we recast Lindell's construction using reactive garbling schemes. In Appendix D, we describe Minilego's garbling and soldering as a reactive garbling scheme. In Appendix F, we prove security of our garbling scheme using the simulation based definition of confidentiality. We also show that simulation based definition implies the indistinguishability based definition of security.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers. For $n \in \mathbb{N}$, let $\{0,1\}^n$ be the set of n-bit strings. Let $\{0,1\}^* = \bigcup_{n \in \mathbb{N}} \{0,1\}^n$. We use $\top$ and $\bot$ as the syntax for *true* and *false* and we assume that $\top, \bot \notin \{0,1\}^*$. We use $()$ to denote the empty sequence. For a sequence σ, we use $x \in \sigma$ to denote that x is in the sequence. When we iterate over $x \in \sigma$ in a for-loop, we do it from left to right. For a sequence σ and an element x we use $\sigma \,\|\, x$ to denote that we append x to σ. We use $\|$ to denote concatenation of sequences. When unambiguous, we also use juxtaposition for concatenating and appending. We use $x \overset{\$}{\leftarrow} X$ to denote sampling a uniformly random x from a finite set X. We use $[A]$ to denote the possible legal outputs of an algorithm A. This is just the set of possible outputs, with $\bot$ removed.

We prove security of protocols in the UC framework and we assume that the reader is familiar with the framework. When we specify entities for the UC framework, ideal functionalities, parties in protocol, adversaries and simulators we give them by a set of rules of the form EXAMPLE (which sends (x_1, x_2) to the adversary in its last line). In Figure 1, we give an an example of a rule. A line of the form "**send** m **to** $\mathcal{F}$.R", where $\mathcal{F}$ is another entity and R the name of a rule, the entity will send (R, id, m) to $\mathcal{F}$, where id is a unique identifier of the rule that is sending, including the session and sub-session identifier, in case many copies of the same rule are currently in execution. We then give $(\mathrm{R}, id, ?)$ to the adversary and let the adversary decide when to deliver the message. Here ? is just a special reserved string indicating

```
rule Example
on $(7, x_1)$ from A
on $x_2$ from B
$x \leftarrow ()$
$x \leftarrow x \,\|\, x_1 \,\|\, x_2$
$z \leftarrow 0$
for $y \in (1,2,4)$ do
    if $z \geq y$ then abort
    $z \leftarrow z + y$
send $x$ to $\mathcal{A}$
```

Fig. 1. A rule

that the real input has been removed. When a message of the form (R, id, m) arrives from an entity A, the receiver stores $(\mathrm{R}, \mathsf{A}, id, m)$ in a pool of pending messages and turns the activation over to the adversary. A line of the form "**on** P **from** A" executed in a rule named R running with identifier id and where P is a pattern, is executed as follows. The entity executing the rule stores $(\mathrm{R}, \mathsf{A}, id, P)$ in a pool of pending receives and turns over the activation to the adversary. We say that a pending message $(\mathrm{R}, \mathsf{A}, id, m)$ matches pending receive $(\mathrm{R}, \mathsf{A}, id, P)$ if m can be parsed on the form P. Whenever an entity turns over the activation to the adversary it sends along $(\mathrm{R}, \mathsf{A}, id, ?)$ for all matched $(\mathrm{R}, \mathsf{A}, id, P)$, where ? is just a special reserved bit-string. There is a special procedure INITIALIZE which is executed once, when the entity is created. All other rules begin with an **on**-command. The rule is considered *ready* for id if the first line is of the form "**on** P **from** A" and $(\mathrm{R}, \mathsf{A}, id, P)$ is matched and the rule was never executed with identifier id. In that case $(\mathrm{R}, \mathsf{A}, id, P)$ is considered to be in the set of pending receives. If the adversary sends $(\mathrm{R}, \mathsf{A}, id, ?)$ to an entity that has some pending receive $(\mathrm{R}, \mathsf{A}, id, P)$ matched by some pending message $(\mathrm{R}, \mathsf{A}, id, m)$, then the entity parses m using P and starts executing right after the line "**on** P **from** A" which added $(\mathrm{R}, \mathsf{A}, id, P)$ to the list of pending receives. A line of the form "**await** P" where P is a predicate on the state of the entity works like the **on**-command. The line turns activation over to the adversary along with an identifier, and the entity will report to the adversary which predicates have become true. The adversary can instruct the entity to resume execution right after any "**await** P" where P is true on the state of the entity. If an entity executes a rule which terminates, it turns the activation over to the adversary. The keyword **abort** makes an entity terminate and ignore all future inputs. A line of the form "**verify** P" makes the entity abort if P is not true on the state of the entity. We use $\mathcal{A}$ to denote the adversary and $\mathcal{Z}$ to denote the environment. A line of the form "**on** P" is equivalent to "**on** P **from** $\mathcal{Z}$". When specifying ideal functionalities we use $\mathsf{Corrupt}$ to denote the set of corrupted parties.

We define security of cryptographic schemes via code-based games [4]. The game is given by a set of procedures. There is a special procedure INITIALIZE which is called once, as the first call. There is another special procedure FINALIZE which may be called by the adversary. The output is true or false, $\top$ or $\bot$, where $\top$ indicates that the adversary won the game. In between INITIALIZE and FINALIZE, the adversary might call the other procedures at will. The other procedures might also output $\bot$ or $\top$ at which point the game ends with that output. Other outputs go back to the adversary.

3 Syntax and Security of Reactive Garbling Schemes

Section overview We will start by defining the notion of gradual function, this will allow us to describe the type of functions that can be garbled. The functions that we define, in contrast to standard garbling schemes allow multiple inputs and outputs as well as partial evaluation.

Next, we will define the syntax of a reactive garbling scheme in the same way that a garbling scheme was described before. We will describe tags, a way of assigning identities to garbled functions, so that we can refer to them later. We will then describe different algorithms: how to encode inputs, decode outputs, link garblings together and other algorithms. Next, we will define correctness. The work of [3] defined the notion of correctness by comparing it to a plaintext evaluation. We define the notion of garbling sequences which is the equivalent of plaintext evaluation but for reactive garbling. Some garbling sequences don't make sense, for example producing an encoded input for a function that has not been defined. As a result, we will define the concept of legal garbling sequences to avoid sequences that are nonsensical. Finally, we can define correctness by comparing the plaintext evaluation of a garbling sequence with the evaluation of a garbling sequence by applying the algorithms define before. We then use the notion of garbling sequence to define the side-information function for reactive garbling. This is necessary to describe our notion of security which we call confidentiality.

Gradual Functions We first define the notion of a gradual function. A gradual function is an extension of the usual notion of a function $f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$, where we allow to partially evaluate the function on a subset of the input components. Some output components might become available before all input components have arrived. We require that when an output component has become available, it cannot become unavailable or change as more input components arrive. We also require that the set of available outputs depends only on which inputs are ready, not on the exact value of the inputs. In our framework, we only allow garblings of gradual functions. This allows us to define partial evaluation and to avoid issues such as circular evaluation and determining when outputs are defined. These issues would make our framework more complex. The access function will be the function describing which outputs are available when a given set of inputs is ready. We will use $\perp$ to denote that an input is not yet specified and that an output is not yet available. We therefore require that $\perp$ is not a usual input or output of the function. We now formalize these notions. For a function $f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$ we use the following notation. $f.n := n$ and $f.m := m$, $f.A := A_1 \times \cdots \times A_n$, $f.B := B_1 \times \cdots \times B_m$, and $f.A_i := A_i$ and $f.B_i := B_i$.

Definition 1. *We use* component *to denote a set $C = \{0,1\}^\ell \cup \{\perp\}$ for some $\ell \in \mathbb{N}$, where $\perp \notin \{0,1\}^*$. We call ℓ the length of C and we write* $\text{len}(C) = \ell$. *Let $C_1, \ldots, C_n$ be components and let $x', x \in C_1 \times \cdots \times C_n$.*

- *We say that x' is an* extension *of x, written $x \sqsubset x'$ if $x_i \neq \perp$ implies that $x_i = x'_i$ for $i = 1, \ldots, n$.*
- *We say that x and x' are* equivalently undefined*, written $x \bowtie x'$, if for all $i = 1, \ldots, n$ it holds that $x_i = \perp$ iff $x'_i = \perp$.*

Definition 2 (Gradual Function). *Let $A_1, \ldots, A_n, B_1, \ldots, B_m$ be components and let $f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$. We say that f is a* gradual function *if it is monotone and variable defined.*

- *It is* monotone *if for all $x, x' \in A_1 \times \cdots \times A_m$ it holds that $x \sqsubset x'$ implies that $f(x) \sqsubset f(x')$.*
- *It is* variable defined *if $x \bowtie x'$ then $f(x) \bowtie f(x')$.*

We say that an algorithm *computes a gradual function* $f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$ if on all inputs $x \in A_1 \times \cdots \times A_m$ it accepts with output $f(x)$ and on all other inputs it rejects. We define a notion of access function which specifies which outputs components will be available given that a given subset of input components are available.

Definition 3 (Access Function). *The access function of a gradual function $f : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$ is a function $\mathsf{access}(f) : \{\bot, \top\}^n \to \{\bot, \top\}^m$ defined as follows. For $j = 1, \ldots, m$, let $q_j : B_j \to \{\bot, \top\}$ be the function where $q_j(\bot) = \bot$ and $q_j(y) = \top$ otherwise. Let $q : B_1 \times \cdots \times B_m \to \{\bot, \top\}^m$ be the function $(y_1, \ldots, y_m) \mapsto (q_1(y_1), \ldots, q_m(y_m))$. For $i = 1, \ldots, n$, let $p_i : \{\bot, \top\} \to A_i$ be the function with $p_i(\bot) = \bot$ and $p_i(\top) = 0^{\mathrm{len}(A_i)}$. Let $p : \{\bot, \top\}^n \to A_1 \times \cdots \times A_n$ be the function $(x_1, \ldots, x_n) \mapsto (p_1(x_1), \ldots, p_n(x_n))$. Then $\mathsf{access}(f) = q \circ f \circ p$.*

Definition 4 (Gradual functional similarity). *Let f,g be gradual functions. We say that f is* similar *to g ($f \sim g$) if $f.n = g.n$, $f.m = g.m$, $f.A = g.A$, $f.B = g.B$ and $\mathsf{access}(f) = \mathsf{access}(g)$.*

In the following, if we use a function at a place where a gradual function is expected and nothing else is explicitly mentioned, we extend it to be a gradual function by adding $\bot$ to all input and output components and letting all outputs be undefined until all inputs are defined.

Syntax of Algorithms A *reactive garbling scheme* consists of seven algorithms $\mathcal{G} = (\mathsf{St}, \mathsf{Gb}, \mathsf{En}, \mathsf{li}, \mathsf{Ev}, \mathsf{ev}, \mathsf{De})$. The algorithms St, Gb and Li are randomized and the other algorithms are deterministic. Gradual functions are described by strings f. We call f the *original gradual function.* For each such description, we require that $\mathsf{ev}(f, \cdot)$ computes some gradual function $\mathsf{ev}(f, \cdot) : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$. This is the function that f describes. We often use f also to denote the gradual function $\mathsf{ev}(f, \cdot)$.

- On input of a security parameter $k \in \mathbb{N}$, the *setup algorithm* outputs a pair of parameters $(\mathsf{sps}, \mathsf{pps}) \leftarrow \mathsf{St}(1^k)$, where $\mathsf{sps} \in \{0, 1\}^*$ is the *secret parameters* and $\mathsf{pps} \in \{0, 1\}^*$ is the *public parameters.* All other algorithms will also receive 1^k as their first input, but we will stop writing that explicitly.
- On input f, a tag[1] $t \in \{0, 1\}^*$ and the secret parameters sps the *garbling algorithm* Gb produces as output a quadruple of strings (F, e, o, d), where F is the *garbled function*, e is the *input encoding function*, d is the *output decoding function*, which is of the form $d = (d_1, \ldots, d_m)$, and o is the *output encoding function.* When $(F, e, o, d) \leftarrow \mathsf{Gb}(\mathsf{sps}, f, t)$ we use F_t to denote F, we use $d_{t,i}$

[1] Some of the algorithms will take as input values output by other algorithms. To identify where these inputs originate from we use tags.

to denote the i^{th} entry of d, and similarly for the other components. This naming is unique by the *function-tag uniqueness* and *garble-tag uniqueness* conditions described later.

– The *encoding algorithm* En takes input (e, t, i, x) and produces *encoded input* $X_{t,i}$.
– The *linking algorithm* li takes input of the form $(t_1, i_1, t_2, i_2, o, e)$ and produces an output L_{t_1,i_1,t_2,i_2} called the *encoded linking information.* Think of this as information which allows to take an encoded output Y_{t_1,i_1} for F_{t_1} and turn it into an encoded input X_{t_2,i_2} for F_{t_2}. In other words, we link the output wire with index i_1 of the garbling with tag t_1 to the input wire with index i_2 of the garbling with tag t_2.
– The *garbled evaluation algorithm* Ev takes as input a set $\mathcal{F}$ of pairs (t, F_t) where t is a tag and F_t a garbled function (let T be the set of tags t occurring in $\mathcal{F}$), a set $\mathcal{X}$ of triples $(t, i, X_{t,i})$ where $t \in T$, $i \in [F_t.n]$ and $X_{i,j} \neq \perp$ is an encoded input, and a set $\mathcal{L}$ of tuples $(t_1, i_1, t_2, i_2, L_{t_1,i_1,t_2,i_2})$ with $t_1, t_2 \in T$ and $i_1 \in [F_{t_1}.m]$ and $i_2 \in [F_{t_2}.n]$ and $L_{t_1,i_1,t_2,i_2} \neq \perp$ an encoded linking information. It outputs a set $\mathcal{Y} = \{(t, i, Y_{t,i})\}_{t \in T, i \in [F_t.m]}$, where each $Y_{t,i}$ is an *encoded output.* It might be that $Y_{t,i} = \perp$ if the corresponding output is not ready.
– The *decoding algorithm* takes input $(t, i, d_{t,i}, Y_{t,i})$, and produces a *final output* $y_{t,i}$. We require that $\mathsf{De}(\cdot, \cdot, \cdot, \perp) = \perp$. The reason for this is that $Y_{t,i} = \perp$ is used to signal that the encoded output cannot be computed yet, and we want this to decode to $y_{t,i} = \perp$. We extend the decoding algorithm to work on sets of decoding functions and sets of encoded outputs, by simply decoding each encoded output for which the corresponding output decoding function is given, as follows. For a set δ, called the *overall decoding function*, consisting of triples of the form $(t, i, d_{t,i})$, and a set $\mathcal{Y}$ of triples of the form $(t, i, Y_{t,i})$, we let $\mathsf{De}(\delta, \mathcal{Y})$ output the set of $(t, i, \mathsf{De}(t, i, d_{t,i}, Y_{t,i}))$ for which $(t, i, d_{t,i}) \in \delta$ and $(t, i, Y_{t,i}) \in \mathcal{Y}$.

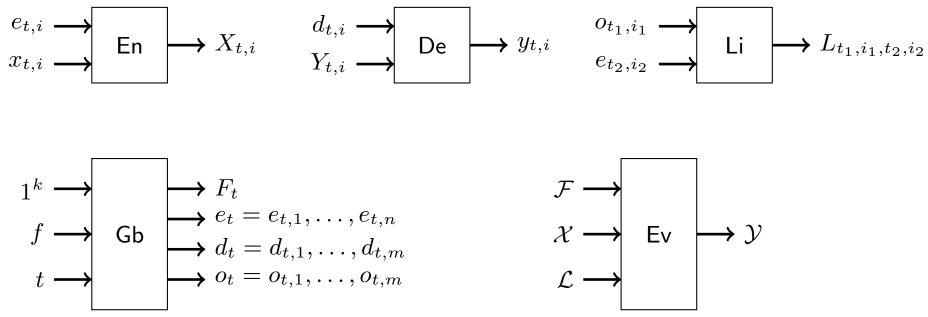


Fig. 2. Input-output behaviour of the central algorithms of a reactive garbling scheme.

Basic requirements We require that $f.n$ and $f.m$ can be computed in linear time from a function description f. We require that $\text{len}(f.A_i)$ and $\text{len}(f.B_j)$ can be computed in linear time for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. We require that the same numbers can be computed in linear time from any garbling F of f. We finally require that one can compute $\mathsf{access}(f)$ in polynomial time given a garbling F of f. We do not impose the length condition and the non-degeneracy condition from [3], i.e., e and d might depend on f. Our security definitions ensure that the dependency does not leak unwarranted information.

Projective Schemes Following [3], we call a scheme projective (on input component i) if all $X \in \{\ En(e,t,i,x) \mid x \in \{0,1\}^n\ \}$ are of the form $\{X_{1,0}, X_{1,1}\} \times \ldots \times \{X_{c,0}, X_{c,1}\}$, where $c = \mathrm{len}(f.X_i)$, and $En(e,t,i,x) = (X_{1,x[1]}, ..., X_{c,x[c]})$. This should hold for all k, f, t, ℓ, $x \in \{0,1\}^c$ and $(\mathsf{sps}, \mathsf{pps}) \in [\mathsf{St}(1^k)]$ and $(F, e, o, d) \in [\mathsf{Gb}(\mathsf{sps}, f, t, \ell)]$. As in [3] being projective is defined only relative to the input encodings. One can define a similar notion for output decodings. Having projective output decodings is needed for capturing some applications using reactive garbling scheme, for instance [20].

Correctness To define correctness, we need a notion of calling the algorithms of a garbling scheme in a meaningful order. For this purpose, we define a notion of *garbling sequence* σ. A garbling sequence is a sequence of *garbling commands*, each command has one of the following forms: $(\texttt{Func}, f, t)$, $(\texttt{Link}, t_1, i_1, t_2, i_2)$, $(\texttt{Input}, t, i, x)$, $(\texttt{Output}, t, i)$, $(\texttt{Garble}, t)$. In the rest of the paper, we will use σ to refer to a garbling sequence. A garbling sequence is called *legal* if the following conditions hold.

Function uniqueness σ does not contain distinct commands $(\texttt{Func}, f_1, t)$ and $(\texttt{Func}, f_2, t)$.
Garble uniqueness Each command $(\texttt{Garble}, t)$ occurs at most once in σ.
Garble legality If $(\texttt{Garble}, t)$ occurs in σ, it is preceded by $(\texttt{Func}, \cdot, t)$.
Linkage legality If the command $(\texttt{Link}, t_1, i_1, t_2, i_2)$ occurs in σ, then the command is preceded by commands of the forms $(\texttt{Func}, f_1, t_1)$, $(\texttt{Garble}, t_1)$, $(\texttt{Func}, f_2, t_2)$ and $(\texttt{Garble}, t_2)$, and $1 \le i_1 \le f_1.m$, $1 \le i_2 \le f_2.n$ and $f_1.B_{i_1} = f_2.A_{i_2}$.
Input legality If $(\texttt{Input}, t, i, x)$ occurs in σ it is preceded by $(\texttt{Func}, f, t)$ and $(\texttt{Garble}, f)$ and $x \in f.A_i \setminus \{\perp\}$.
Output legality If $(\texttt{Output}, t, i)$ occurs in a sequence it is preceded by $(\texttt{Func}, f, t)$ and $(\texttt{Garble}, t)$ and $1 \le i \le f.m$.

Note that if a sequence is legal, then so is any prefix of the sequence. We call a garbling sequence *illegal* if it is not legal. Since we allow to link several output components onto the same input component we have to deal with the case where they carry different values. We consider this an error, and to catch it, we use the following safe assignment operator.

$$(u \hookleftarrow v) := \begin{cases} u \leftarrow \texttt{Error} & \text{if } v = \texttt{Error} \\ u \leftarrow u & \text{if } v = \perp \\ u \leftarrow v & \text{if } u = \perp \vee u = v \\ u \leftarrow \texttt{Error} & \text{otherwise} \end{cases}$$

We now define an algorithm eval, which takes as input a legal garbling sequence σ and outputs a set of tuples $(t, i, y_{t,i})$, one for each command $(\texttt{Output}, t, i)$, where possibly $y_{t,i} = \perp$. The values are computed by taking the least fix point of the evaluation of all the gradual functions, see Figure 3. We call this the *plain evaluation* of σ. We extend the definition of a legal sequence to include the requirement that

```
proc $\mathsf{eval}(\sigma \in \mathbb{L})$
for $(\texttt{Func}, t, f) \in \sigma$, do
    $f_t \leftarrow f$
    for $i = 1, \dots, f_t.n$ do $x_{t,i} \leftarrow \perp$
    for $j = 1, \dots, f_t.m$ do $y_{t,j} \leftarrow \perp$
for $(\texttt{Input}, t, i, x) \in \sigma$ do $x_{t,i} \hookleftarrow x$
$T \leftarrow \emptyset$
repeat
    $U \leftarrow T$
    for $(\texttt{Func}, t, f) \in \sigma$ do
        $(y_{t,1}, \dots, y_{t,f_t.m}) \leftarrow f_t(x_{t,1}, \dots, x_{t,f_t.n})$
        for $(\texttt{Link}, t, i_1, t_2, i_2) \in \sigma$ do $x_{t_2,i_2} \hookleftarrow y_{t,i_1}$
    $T \leftarrow \{(t, i, y_{t,i}) \mid t \in \mathrm{Tags}(\sigma), i = 1, \dots, f_t.m\}$
until $T = U \vee (\cdot, \cdot, \texttt{Error}) \in T$
return $T$
```

Fig. 3. Plaintext evaluation

Input uniqueness $(\cdot, \cdot, \texttt{Error}) \notin \mathsf{eval}(\sigma)$.

Therefore the use of the safe assignment in eval is only to conveniently define the notion of legal sequence. In the rest of the paper we assume that all inputs to eval are legal. The values $y_{t,i} \neq \perp$ are by definition the values that are *ready* in σ, i.e., $\mathsf{ready}(\sigma) = \{(t, i) | \exists (t, i, y_{t,i}) \in \mathsf{eval}(\sigma)(y_{t,i} \neq \perp)\}$. Note that since the gradual functions are variable defined, which outputs are ready does not depend on the values of the inputs, except via whether they are $\perp$ or not.

The procedure Eval in Figure 4 demonstrates how a legal garbling sequence is intended to be translated into calls to the algorithms of the garbling scheme. We call the procedure executed by Eval *garbled evaluation* of σ.

```
proc $\mathsf{Eval}(\sigma \in \mathbb{L})$
for $c \in \sigma$ do
    if $c = (\texttt{Func}, t, f)$ then $f_t \leftarrow f$;
    if $c = (\texttt{Garble}, t)$ then
        $(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(\mathsf{sps}, f_t, t)$
        $\mathcal{F} \leftarrow \mathcal{F} \,\|\, (t, F_t)$
    if $c = (\texttt{Input}, t, i, x)$ then
        $X_{t,i} \leftarrow \mathsf{En}(e_t, t, i, x)$
        $\mathcal{X} \leftarrow \mathcal{X} \,\|\, (t, i, X_{t,i})$
    if $c = (\texttt{Link}, t_1, i_1, t_2, i_2)$ then
        $L_{t_1,i_1,t_2,i_2} \leftarrow \mathsf{li}(t_1, i_1, t_2, i_2, o_{t_1}, e_{t_2})$
        $\mathcal{L} \leftarrow \mathcal{L} \,\|\, (t_1, i_1, t_2, i_2, L_{t_1,i_1,t_2,i_2})$
    if $c = (\texttt{Output}, t, i)$ then
        $\delta \leftarrow \delta \,\|\, (t, i, d_{t,i})$
return $\mathsf{De}(\delta, \mathsf{Ev}(\mathcal{F}, \mathcal{X}, \mathcal{L}))$
```

Fig. 4. Garbled Evaluation

Lemma 1. *For a function description f, let $T(f)$ be the worst case running time of $\mathsf{ev}(f, \cdot)$. The algorithm* eval *will terminate in time* $\mathrm{poly}(T|\sigma|(n+m))$, *where* $n = \max_{(\texttt{Func},t,f)\in\sigma} f.n$, $m = \max_{(\texttt{Func},t,f)\in\sigma} f.m$, *and* $T = \max_{(\texttt{Func},t,f)\in\sigma} T(f)$.

Proof. By monotonicity, if the loop in eval does not terminate, another variable $y_{t,i}$ has changed from $\perp$ to $\neq \perp$ and can never change value again. This bounds the number of iterations as needed.

Side-Information Functions We use the same notion of side-information functions as in [3]. A side information function Φ maps function descriptions f into the side information $\Phi = \Phi(f) \in \{0,1\}^*$. Intuitively, a garbling of f should not leak more than $\Phi(f)$. The exact meaning of the side information functions are given by our security definition. We extend a side information function Φ to the set of garbling sequences. For the empty sequence $\sigma = ()$ we let $\Phi(\sigma) = ()$.

For a sequence σ, we define the side-information as $\Phi(\sigma) := \Phi_\sigma(\sigma)$ where for a sequence $\bar{\sigma}$ and a command c: $\Phi_\sigma(\bar{\sigma} \,\|\, c) = \Phi_\sigma(\bar{\sigma}) \,\|\, \Phi_\sigma(c)$, where $\Phi_\sigma(\texttt{Func}, t, f) = (\texttt{Func}, t, \Phi(f))$, $\Phi_\sigma(\texttt{Link}, t_1, i_1, t_2, i_2) = (\texttt{Link}, t_1, i_1, t_2, i_2)$, $\Phi_\sigma(\texttt{Input}, t, i, x) = (\texttt{Input}, t, i, |x|)$, $\Phi_\sigma(\texttt{Garble}, t) = (\texttt{Garble}, t)$ and $\Phi_\sigma(\texttt{Output}, t, i) = (\texttt{Output}, t, i, y_{t,i})$, where $y_{t,i}$ is defined by $\mathsf{eval}(\sigma)$.

Legal Sequence Classes We define the notion of a *legal sequence class* $\mathbb{L}$ (relative to a given side-information function Φ). It is a subset of the legal garbling sequences which additionally has these five properties:

Monotone If $\sigma' \,\|\, \sigma'' \in \mathbb{L}$, then $\sigma' \in \mathbb{L}$.
Input independent If $\sigma' \,\|\, (\texttt{Input}, t, i, x) \,\|\, \sigma'' \in \mathbb{L}$, then $\sigma' \,\|\, (\texttt{Input}, t, i, x') \,\|\, \sigma'' \in \mathbb{L}$ for all $x' \in \{0,1\}^{|x|}$.
Function independent If $\sigma' \,\|\, (\texttt{Func}, t, f) \,\|\, \sigma'' \in \mathbb{L}$, then $\sigma' \,\|\, (\texttt{Func}, t, f') \,\|\, \sigma'' \in \mathbb{L}$ for all f with $\Phi(f') = \Phi(f)$.
Name invariant If $\sigma \in \mathbb{L}$ and σ' is σ with all tags t replaced by $t' = \pi(t)$ for an injection π, then $\sigma' \in \mathbb{L}$.
Efficient Finally, the language $\mathbb{L}$ should be in P, i.e., in polynomial time.

It is easy to see that the set of all legal garbling sequences is a legal sequence class.

Definition 5 (Correctness). *For a legal sequence class* $\mathbb{L}$ *and a reactive garbling scheme* $\mathcal{G}$ *we say that* $\mathcal{G}$ *is* $\mathbb{L}$-correct *if for all* $\sigma \in \mathbb{L}$*, it holds that* $\mathsf{De}(\mathsf{Eval}(\sigma)) \subseteq \mathsf{eval}(\sigma)$ *for all choices of randomness by the randomized algorithms.*

Function Individual Garbled Evaluation The garbled evaluation function Ev just takes as input sets of garbled functions, inputs and linking information and then somehow produces a set of garbled outputs. It is often convenient to have more structure to the garbled evaluation than this.

We say that garbled evaluation is *function individual* if each garbled function F is evaluated on its own. Specifically there exist deterministic poly-time algorithms EvI and Li called the *individual garbled evaluation algorithm* and the *garbled linking algorithm*. The input to EvI is a garbled function and some garbled inputs. For each fixed garbled function F with $n = F.n$ and $m = F.m$ the algorithm computes a gradual function $\mathsf{EvI}(F) : A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_m$ and $(X_1, \ldots, X_n) \mapsto \mathsf{EvI}(F, X_1, \ldots, X_n)$, with $\mathsf{access}(\mathsf{EvI}(F)) = \mathsf{access}(f)$, where f is the function garbled by F. We denote the output by $(Y_1, \ldots, Y_m) = \mathsf{EvI}(F, X_1, \ldots, X_n)$. The intention is that the Y_j are garbled outputs (or $\perp$). To say that Ev has individual garbling we then require that it is defined from EvI and Li as in Figure 5.

```
proc $\mathsf{Ev}(\mathcal{F}, \mathcal{X}, \mathcal{L})$
for $(t, F) \in \mathcal{F}$ do
    $F_t \leftarrow F$
    for $i = 1, \ldots, F_t.n$ do $X_{t,i} \leftarrow \perp$
for $(t, i, X) \in \mathcal{X}$ do  $X_{t,i} \leftarrow X$
$T \leftarrow \emptyset$
repeat
    $U \leftarrow T$
    for $(t, F_t) \in \mathcal{F}$ do
        $(Y_{t,1}, \ldots, Y_{t,F_t.m}) \leftarrow$
        $\mathsf{Evl}(F_t, (X_{t,1}, \ldots, X_{t,F_t.n}))$
        for $(t, i_1, t_2, i_2, L) \in \mathcal{L}$ do $X_{t_2,i_2} \leftarrow \mathsf{Li}(L, Y_{t,i_1})$
        $T \leftarrow \{(t, i, Y_{t,i}) \mid t \in \text{Tags}(\sigma) \wedge i = 1, \ldots, F_t.m\}$

until $T = U$
return $T$
```

Fig. 5. Function Individual Evaluation

Security of Reactive Garbling We define a notion of security that we call confidentiality, which unifies privacy and obliviousness as defined in [3]. Obliviousness says that if the evaluator is given a garbled function and garbled inputs but no output decoding function it can learn a garbled output of the function but learns no information on the plaintext value of the output. Privacy says that if the evaluator is given a garbled function, garbled inputs and the output decoding function it can learn the plaintext value of the function, but no other information, like intermediary values from the evaluation. It is necessary to synthesise these properties as we envision protocols where the receiver of the garbled functions might receive the output decoding function for *some* of the output components but *not all* of them. Obliviousness does not cover this case, since the adversary has some of the decoding keys. It is not covered by privacy either, as the receiver should not gain any information about outputs for which he does not have a decoding function.

In the confidentiality (indistinguishability) game, the adversary feeds two sequences σ_0 and σ_1 to the game, which produces a garbling of one of the two sequences, σ_b for a uniform bit b. The adversary wins if it can guess which sequence was garbled. It is required that the two sequences are not trivially distinguishable. For instance, the two commands at position i in the two sequences should have the same type, the side information of functions at the same positions in the sequences should be the same, and all outputs produced by the sequences should be the same. This is formalized by requiring that the side information of the sequences are the same. This is done by checking that $\Phi(\sigma_0) = \Phi(\sigma_1)$ in the rule FINALIZE. If one considers garbling sequences with only one function command, one garbling command, one input command per input component, no linking and where no output command is given, then confidentiality implies obliviousness. If in addition an output command is given for each output component, then confidentiality implies privacy.

In the confidentiality (simulation) game, the adversary feeds a sequence σ to the game. The game samples a uniform bit b. If $b = 0$, then the game uses the reactive garbling scheme to produce values for the sequence. Otherwise, if the bit $b = 1$, the game feeds the output of the side-information function to the simulator and forwards any response to the adversary. We list this notion of security in appendix E.

In appendix F, we show that the simulation-based notion of confidentiality implies the indistinguishability-based notion of indistinguishability.

Definition 6 (Confidentiality). *For a legal sequence class $\mathbb{L}$ relative to side-information function Φ and a reactive garbling scheme $\mathcal{G}$, we say that $\mathcal{G}$ is $(\mathbb{L}, \Phi)$-*confidential *if for all PPT $\mathcal{A}$ it holds that $\mathbf{Adv}^{\text{adp.ind.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k)$ is negligible, where $\mathbf{Adv}^{\text{adp.ind.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k) = \Pr[\mathbf{Game}^{\text{adp.ind.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k) = \top] - \frac{1}{2}$ and $\mathbf{Game}^{\text{adp.ind.con}}_{\mathcal{G},\mathbb{L}',\Phi}$ is given in Figure 6.*

Notice that this security definition is indistinguishability based, which is known to be very weak in some cases for garbling (cf. [3]). Consider for instance garbling a function f where the input x is secret and $y = f(x)$ is made a public output. The security definition then only makes a requirement on the garbling scheme in the case where the adversary inputs two sequences where in sequence one the input is x_1 and in sequence two the input is x_2 and where $f(x_1) = f(x_2)$. Consider then what happens if f is collision resistant. Since no adversary can compute such x_1 and x_2 where $x_1 \neq x_2$, it follows that $x_1 = x_2$ in all pairs of sequences that the adversary can submit to the game. It can then be seen that it would be secure to “garble” collision resistant functions f by simply sending f in plaintext. Despite this weak definition, we later manage to prove that it is sufficient for building secure two-party computation. Looking ahead, when we need to securely compute f, we will garble a function f' which takes an additional input p which is the same length as the output of f and where $f'(x, p) = p \oplus f(x)$ and ask the party that supplies p to always let p be the all-zero string. Our techniques for ensuring active security in general is used to enforce that even a corrupted party does this. Correctness is thus preserved. Clearly f' is not collision resistant even if f is collision resistant. This prevents a secure garbling scheme from making insecure garblings of f'. In fact, note that this trick ensures that f' has the efficient invertibility property defined by [3], which means that the indistinguishability and simulation based security coincide.

4 Instantiating a Confidential Reactive Garbling Scheme

We show that the instantiation of garbling schemes in [3] can be extended to a reactive garbling scheme in the random-oracle (RO) model. We essentially implement the dual-key cipher construction from [3] using the RO. To link a wire with 0-token T_0 and 1-token T_1 to an input wire with tokens I_0 and I_1, we provide the linking information $L_0 = RO(T_0) \oplus I_0$ and $L_1 = RO(T_1) \oplus I_1$ in a random order with each value tagged by the permutation bits of their corresponding input wires and output wires. Evaluation is done using function individual evaluation. Evaluation of a single garbled circuit is done as in [3]. Evaluation of a linking is: given T_b and a permutation bit, the bit is used to retrieve L_b from which $I_b = L_b \oplus RO(T_b)$ is computed. We provide the details

```
proc INITIALIZE()
$b \xleftarrow{\$} \{0,1\}$
$\sigma_0 \leftarrow \emptyset$
$\sigma_1 \leftarrow \emptyset$

proc OUTPUT($t, i$)
for $c \in \{0,1\}$ do
$\sigma_c \leftarrow \sigma_c \,\|(\texttt{Output}, t, i)$
return $d_{t,i}$

proc LINK($t_1, i_1, t_2, i_2$)
for $c \in \{0,1\}$ do
$\sigma_c \leftarrow \sigma_c \,\|(\texttt{Link}, t_1, i_1, t_2, i_2)$
return $\mathsf{li}(t_1, i_1, t_2, i_2, o_{t_1,i_1}, e_{t_2,i_2})$

proc GARBLE($t$)
for $c \in \{0,1\}$ do $\sigma_c \leftarrow \sigma_c \,\|(\texttt{Garble}, t)$
$(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(\mathsf{sps}, f_t, t)$
return $F_t$

proc FUNC($f_0, f_1, t$)
for $c \in \{0,1\}$ do
$\sigma_c \leftarrow \sigma_c \,\|(\texttt{Func}, f_c, t)$
if $f_0 \not\sim f_1$ then return $\perp$

proc INPUT($t, i, x_0, x_1$)
for $c \in \{0,1\}$ do
$\sigma_c \leftarrow \sigma_c \,\|(\texttt{Input}, t, i, x_c)$
return $\mathsf{En}(e_t, t, i, x_b)$

proc FINALIZE($b'$)
if $b = b' \wedge \Phi(\sigma_0) = \Phi(\sigma_1) \wedge \sigma_0 \in \mathbb{L}$
then return $\top$
else return $\perp$
```

Fig. 6. The game $\mathbf{Game}^{\mathrm{adp.ind.con}}_{\mathcal{G},\mathbb{L},\Phi}(1^k)$ defining *adaptive indistinguishability confidentiality*. In FINALIZE we check that $\sigma_0 \in \mathbb{L}$ and the adversary loses if this is not the case. It is easy to see that when $\mathbb{L}$ is a legal sequence class and $\Phi(\sigma_0) = \Phi(\sigma_1)$, then $\sigma_0 \in \mathbb{L}$ iff $\sigma_1 \in \mathbb{L}$. We can therefore by monotonicity assume that the game returns $\perp$ as soon as it happens that $\sigma_c \notin \mathbb{L}$. We use a number of notational conventions from above. Tags are used to name objects relative to σ_c, which is assumed to be legal. As an example, in $\texttt{Garble}(t)$, the function f_t refers to the function f_c occurring in the command $(\texttt{Func}, f_c, t)$ which was added to σ_c in FUNC by **Garble Legality**. For another example, the $d_{t,i}$ in OUTPUT(t, i) refers to the i^{th} component of the d_t component output by $\mathsf{Gb}(\mathsf{sps}, f_t, t)$ in the execution of GARBLE(t, π) which must have been executed by **Output Legality**.

in Section 4.1 and its proof of security in Appendix B. We use the RO because reactive garbling schemes run into many of the same subtle security problems as adaptive garbling schemes [2], which are conveniently handled by being able to program the RO. We leave as an open problem the construction of (efficient) reactive garbling schemes in the standard model.

4.1 A reactive garbling scheme

We will now give the details of the construction of a confidential reactive garbling scheme based on a random oracle. The protocol is inspired by the construction of garbling schemes from dual-key ciphers presented in [3]. The pseudocode for our reactive garbling scheme is shown in Figure 7 and Figure 8.

To simplify notation, we define lsb as the least significant bit, slsb as the second least significant bit. The operation Root removes the last two bits of a string. The symbol H denotes the random oracle.

We use the notation of [3] to represent a circuit. A circuit is a 6-tuple $f = (n, m, q, A, B, G)$. Here $n \geq 2$ is the number of inputs, $m \geq 1$ is the number

```
proc $\mathsf{Gb}(f_t, t)$
$(n, m, q, A, B, G) \leftarrow f_t$
for $i \in \{1, \ldots, n+q-m\}$ do
    $c \xleftarrow{\$} \{0,1\}$                                  // Type of the zero-encoding
    $X_{t,i,0} \leftarrow \{0,1\}^{k-1} \,\|\, c$
    $X_{t,i,1} \leftarrow \{0,1\}^{k-1} \,\|\, 1-c$
for $i \in \{1, \ldots, m\}$ do
    $c \xleftarrow{\$} \{0,1\}, r_i \xleftarrow{\$} \{0,1\}$                  // Type and mask of zero-encoding
    $Y_{t,i,0} \leftarrow \{0,1\}^{k-2} \,\|\, r_i \,\|\, c$
    $Y_{t,i,1} \leftarrow \{0,1\}^{k-2} \,\|\, 1-r_i \,\|\, 1-c$
    $X_{t,n+q-m+i,0} \leftarrow Y_{t,i,0}$
    $X_{t,n+q-m+i,1} \leftarrow Y_{t,i,1}$
for $(i, u, v) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ do
    $a \leftarrow A(i), b \leftarrow B(i)$                            // Left wire, right wire
    // Left-wire encoding of $u$ and its type.
    $A \leftarrow \mathsf{root}(X_{t,a,u}), \mathfrak{a} \leftarrow \mathsf{lsb}(X_{t,a,u})$
    // Right-wire encoding of $v$ and its type.
    $B \leftarrow \mathsf{root}(X_{t,b,v}), \mathfrak{b} \leftarrow \mathsf{lsb}(X_{t,b,v})$
    // Unique tag
    $T \leftarrow t \,\|\, i \,\|\, \mathfrak{a} \,\|\, \mathfrak{b}$
    // Row of Garbled table associated to gate $i$ and input $(u, v)$
    $P[i, \mathfrak{a}, \mathfrak{b}] \leftarrow \mathsf{H}(T \,\|\, A \,\|\, B) \oplus Y_{t,i,G(i,u,v)}$
$F_t \leftarrow (n, m, q, A, B, P)$
$e_t \leftarrow ((X_{1,0}, X_{1,1}), \ldots, (X_{n,0}, X_{n,1}))$
$o_t \leftarrow ((Y_{1,0}, Y_{1,1}), \ldots, (Y_{m,0}, Y_{m,1}))$
$d_t \leftarrow \{r_1, \ldots, r_m\}$
return $(F_t, e_t, o_t, d_t)$

proc $\mathsf{En}(t, i, x)$
$X_{t,i} \leftarrow e_{t,i,x}$
return $X_{t,i}$

proc $\mathsf{De}(t, i, Y_{t,i}, d_{t,i})$
$y_{t,i} \leftarrow \mathsf{slsb}(Y_{t,i}) \oplus d_{t,i}$
return $y_{t,i}$
```

Fig. 7. Reactive garbling scheme

of outputs and $q \geq 1$ is the number of gates. We let $r = n + q$ be the number of wires. We let Inputs $= \{1, \ldots, n\}$, Wire $= \{1, \ldots, n+q\}$, OutputWires $= \{n+q-m+1, \ldots, n+q\}$ and Gates $= \{n, \ldots, n+q\}$. Then A : Gates $\rightarrow$ Wires $\setminus$ OutputWires is a function to identify each gate's first incoming wire and B : Gates $\rightarrow$ Wires $\setminus$ OutputWires is a function to identify each gate's second incoming wire. Finally, G : Gates $\times \{0,1\}^2 \rightarrow \{0,1\}$ is a function that determines the functionality of each gate. We require that $A(g) < B(g) < g$ for all $g \in$ Gates.

Our protocol will also follow the approach of [3]. To garble a circuit, two tokens are selected for each wire, one denoted by $X_{t,i,0}$ which shall encode the

```
proc $\mathsf{li}(o_{t_1,i_1}, e_{t_2,i_2})$
// Type of zero-encoding
$c \leftarrow \mathsf{lsb}(o_{t_1,i_1,0})$
$K_0 \leftarrow \mathsf{root}(o_{t_1,i_1,0})$
$K_1 \leftarrow \mathsf{root}(o_{t_1,i_1,1})$
$T \leftarrow (t_1, i_1, t_2, i_2)$
// Encryption of encoded input whose
   associated output encoding has
   type 0
$U_0 \leftarrow \mathsf{H}(T \,\|\, k_c) \oplus e_{t_2,i_2,c}$
// Encryption of input encoding whose
   associated output encoding has
   type 1
$U_1 \leftarrow \mathsf{H}(T \,\|\, k_{1\oplus c}) \oplus e_{t_2,i_2,1\oplus c}$
$L_{t_1,i_1,t_2,i_2} \leftarrow (U_0, U_1)$
return $L_{t_1,i_1,t_2,i_2}$

proc $\mathsf{Li}(L_{t_1,i_1,t_2,i_2}, Y_{t_1,i_1})$
$r \leftarrow \mathsf{lsb}(Y_{t_1,i_1})$
$K \leftarrow \mathsf{root}(Y_{t_1,i_1})$
$T \leftarrow (t_1, i_1, t_2, i_2)$
$X_{t,i} \leftarrow \mathsf{H}(T \,\|\, k) \oplus L_{t_1,i_1,t_2,i_2,r}$
return $X_{t,i}$

proc $\mathsf{Evl}(F_t, X_1, \ldots, X_n)$
$(n, m, q, A, B, P) \leftarrow F_t$
for $i \leftarrow n+1$ to $n+q$ do
    $a \leftarrow A(i), b \leftarrow B(i)$
    $A \leftarrow X_{t,a}, B \leftarrow X_{t,b}$
    if $A \neq \perp \wedge B \neq \perp$ then
        $\mathfrak{a} \leftarrow \mathsf{lsb}(A), \mathfrak{b} \leftarrow \mathsf{lsb}(B)$
        $T \leftarrow t \,\|\, i \,\|\, \mathfrak{a} \,\|\, \mathfrak{b}$
        $X_g \leftarrow P[g, \mathfrak{a}, \mathfrak{b}] \oplus \mathsf{H}(T \,\|\, A \,\|\, B)$
$(Y_{t,i}, \ldots, Y_{t,m}) \leftarrow (X_{n+q-m+1}, \ldots, X_{n+q})$
return $(Y_{t,1}, \ldots, Y_{t.m})$
```

Fig. 8. Reactive garbling scheme (continued)

value 0 and the other denoted by $X_{t,i,1}$ which will encode the value 1, we refer to this mapping as the semantic of a token.

The encoding of an input for a value x is simply the token of the given wire with semantic x. The decoding of an output is the mask for that wire. We decouple the decoding from the linking to simplify the proof of security. The simulator will be able to produce linking without having to worry about the semantics of the output encoding.

For each wire, the two associated tokens will be chosen such that the least significant bit (the type of a token) will differ. It is important to note that the semantics and type of a token are independent. The second least significant bit is called the mask and will have a special meaning later when the tokens are output tokens. We use $\mathsf{root}(X)$ to denote the part of a token that is not the type bit or the mask bit.

Each gate g will be garbled by producing a garbled table. A garbled table will consist of four ciphertexts $p[g, a, b]$ where $a, b \in \{0, 1\}$, The ciphertext $P[g, a, b]$ will be produced in the following way: first find the token associated to the left input wire (i_1) with type a, denote the semantic of this token as x. Secondly, find the token associated to the right input wire (i_2) with type b, denote the semantic of this token as y. The ciphertext will be an encryption of the token of $z \leftarrow G(g, x, y)$. We will denote $T \leftarrow t \,\|\, g \,\|\, a \,\|\, b$. The encryption will be $P[g, a, b] \leftarrow \mathsf{H}(T \,\|\, \mathsf{root}(X_{t,i_1,x}) \,\|\, \mathsf{root}(X_{t,i_2,y})) \oplus (X_{t,i,z})$

For each non-output wire, the token with semantic 0 will be chosen randomly and the token with semantic 1 will be chosen uniformly at random except for the last bit which will be chosen to be the negation of the least significant bit of the token with semantic 0 for the same wire.

For each output wire, the first token will also be chosen uniformly at random. The token with semantic 0 will be chosen randomly and the token with semantic 1 will be chosen uniformly at random except for the least significant bit and the second least significant bit. For both of these positions, the second token will be chosen so that they differ from the value in the 0-token for the same position. We refer to the second least significant bit of the 0-token of an output token as the mask of an output wire.

A linking between output (t_1, i_1) and input (t_2, i_2) consists of two ciphertexts: let c be the type of the 0-token for the output wire. In this case, we set $T = t_1 \,\|\, i_1 \,\|\, t_2 \,\|\, i_2$. The linking is simply

$$L \leftarrow (E^T_{\mathsf{root}(Y_{t_1,i_1,c})}(X_{t_2,i_2,c}), E_{\mathsf{root}(Y_{t_1,i_1,1-c})}(X_{t_2,i_2,1-c}))$$

where $E^T_k(z) = \mathsf{H}(T||k) \oplus z$. Converting an encoded output into an encoded input follows naturally.

In Appendix B we prove the following theorem.

Theorem 1. *Let $\mathbb{L}$ be the set of all legal garbling sequence, let Φ denote the circuit topology of a function. Then RGS is $(\mathbb{L}, \Phi)$-confidential in the random oracle model.*

5 Application to Secure Reactive Two-Party Computation

We now show how to implement reactive two-party computation secure against a malicious, static adversary using a projective reactive garbling scheme. For simplicity we assume that $\mathbb{L}$ is the set of all legal sequences. It can, however, in general consist of a set of sequences closed under the few augmentations we do of the sequence in the protocol. The implementation could be optimized using contemporary tricks for garbling based protocols, but we have chosen to not do this, as the purpose of this section is to demonstrate the use of our security definition, not efficiency.

```
rule INITIALIZE
$\sigma \leftarrow \{\}$

rule INPUT_A
on $(\texttt{Input}, t, i, x)$ from A
on $(\texttt{Input}, t, i, ?)$ from B
await $(\texttt{Garble}, t) \in \sigma$
on $(\texttt{Input}, t, i, x')$ from $\mathcal{S}$
if A $\in$ Corrupt then $x \leftarrow x'$
send $(\texttt{Input}, t, i, \texttt{done})$ to A
send $(\texttt{Input}, t, i, \texttt{done})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, x)$

rule LINK
await $(\texttt{Garble}, t) \in \sigma)$
on $(\texttt{Link}, t_1, i_1, t_2, i_2)$ from A
on $(\texttt{Link}, t_1, i_1, t_2, i_2)$ from B
await $(\texttt{Garble}, t) \in \sigma)$
send $(\texttt{Link}, t_1, i_1, t_2, i_2, \texttt{done})$ to A
send $(\texttt{Link}, t_1, i_1, t_2, i_2, \texttt{done})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Link}, t_1, i_1, t_2, i_2)$

rule OUTPUT
on $(\texttt{Output}, t, i)$ from A
on $(\texttt{Output}, t, i)$ from B
await $\exists(t, i, y_{t,i} \neq \perp) \in \mathsf{eval}(\sigma)$
send $(\texttt{Output}, t, i, \texttt{done})$ to A
send $(\texttt{Output}, t, i, y_{t,i})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Output}, t, i)$
```

```
rule FUNC
on $(\texttt{Func}, t, f)$ from A
on $(\texttt{Func}, t, f)$ from B
$\sigma \leftarrow \sigma \| (\texttt{Func}, t, f)$

rule INPUT_B
on $(\texttt{Input}, t, i, ?)$ from A
on $(\texttt{Input}, t, i, x)$ from B
await $(\texttt{Garble}, t) \in \sigma)$
on $(\texttt{Input}, t, i, x')$ from $\mathcal{S}$
if B $\in$ Corrupt then $x \leftarrow x'$
send $(\texttt{Input}, t, i, \texttt{done})$ to A
send $(\texttt{Input}, t, i, \texttt{done})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, x)$

rule GARBLE
on $(\texttt{Garble}, t)$ from A
on $(\texttt{Garble}, t)$ from B
await $(\texttt{Func}, t, f) \in \sigma$
send $(\texttt{Garble}, t, \texttt{done})$ to A
send $(\texttt{Garble}, t, \texttt{done})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Garble}, t)$
```

Fig. 9. Ideal Functionality $\mathcal{F}_{\texttt{R2PC}}^{\mathbb{L},\Phi}$ (only suitable for static security). For each line of the form, "**on** c **from** P" for a command c and a party P, when the activation is given to the adversary the ideal functionality sends along $(\Phi(c), \mathsf{P})$.

We implement the ideal functionality in Figure 9. The inputs to the parties will be a garbling sequence. The commands are received one-by-one, to have a well defined sequence, but can be executed in parallel. We assume that at any point in time the input sequence received by a party is a prefix or suffix of the input sequence of the other parties, except that when a party receives a secret input by receiving input $(\texttt{Input}, t, i, x)$, then the other party receives $(\texttt{Input}, t, i, ?)$, to not leak the secret x, where we use ? to denote a special reserved input indicating that the real input has been removed. We also assume that the sequence of inputs given to any party is in $\mathbb{L}$. If not, the ideal functionality will simply stop operating. We only specify an ideal functionality for static security. To correctly handle adaptive security a party should sometimes be allowed to replace its input when becoming adaptively corrupted. Since we only prove static security, we chose to not add these complication to the specification.

The implementation will be based on the idea of a watchlist [17]. Alice and Bob will run many instances of a base protocol where Alice is the garbler and Bob is the evaluator. Alice will in each instance provide Bob with garbled functions, linking information, encoded inputs for Alice's inputs and encoded inputs for Bob's inputs, and decoding information. For all Bob's input bits, Alice computes encodings of both 0 and 1, and Bob uses an oblivious transfer to pick the encoding he wants. For a given input bit, the same oblivious transfer instance is used to choose the appropriate encodings in all the instances. This forces Bob to use the same input in all instances. Bob then does a garbled evaluation and decodes to get a plaintext output. Bob therefore gets one possible value of the output from each instance. If Alice cheats by sending incorrect garblings or using different inputs in different instances, the outputs might be different. We combat this by using a watchlist. For a random subset of the instances, Bob will learn all the randomness used by Alice to run the algorithms of the garbling scheme and Bob can therefore check whether Alice is sending the expected values in these instances. The instances inspected by Bob are called the *watchlist instances*. The other instances are called the *evaluation instances*. The watchlist is random and unknown to Alice. The number of instances and the size of the watchlist is set up such that except with negligible probability, either a majority of the evaluation instances are correct or Bob will detect that Alice cheated without leaking information about his input. Bob can therefore take the output value that appears the most often among the evaluation instances as his output. There are several issues with this general approach that must be handled.

1. We cannot allow Bob to learn the encoded inputs of Alice in watchlist instances, as Bob also knows the input encoding functions for the watchlist instances. This is handled by letting Alice send her random tape r_i for each instance i to Bob in an oblivious transfer, where the other message is a key that will be used by Alice to encrypt the encodings of her input. That way Bob can choose to *either* make instance i a watchlist instance, by choosing r_i, or learn the encoded inputs of Alice, but not both.
2. Alice might not send correct input encodings of her own inputs, in which case correctness is not guaranteed. This is not caught by the watchlist mechanism as Bob does not learn Alice's input encodings for the watchlist instances. To combat this attack, Alice must for all input bits of Alice, in all instances, commit to both the encoding of 0 and 1, in a random order, and send along with her input encodings an opening of one of the commitments. The randomness used to commit is picked from the random tape that Bob knows in the watchlist instances. That way Bob can check in the watchlist instances that the commitments were computed correctly, and hence the check in the evaluation position that the encoding sent by Alice opens one of the commitments will ensure that most evaluation instances were run with correct input encodings, except with negligible probability.
3. We have to ensure that Alice uses the same input for herself in all instances. For the same reason as item 2, this cannot be caught by the watchlist mechanism. Instead, it is done by revealing in all instances a privacy-preserving

message digest of Alice's input. Bob can then check that this digest is the same in all instances. For efficiency, the digest is computed using a two-universal hash function. This is a common trick by now, see [9,25,7]. However, all previous work used garbled circuits in a white box manner to make this trick work. We can do it by a black box use of reactive garbling, as follows. First Alice garbles the function f to be evaluated producing the garbling F where Alice is to provide some input component x. Then Alice garbles the function g which takes as input a mask m, an index c for a family h of two-universal hash functions and an input x for the hash function and which outputs x and $y = h_c(x) \oplus m$. Alice then randomly samples a mask m and then sends encodings of m and x to Bob as well as the output decoding function for y. Bob then samples an index c at random and makes it public. Then Alice sends the encoding of c to Bob. Alice then links the output component x of G into the input component x of F. This lets Bob compute y and an encoding X of the input x of f.

4. As usual Alice can mount a selective attack by for example offering Bob a correct encoding of 0 and an incorrect encoding of 1 in one of the OTs used for picking Bob's input. This will not be caught by the watchlist mechanism if Bob's input is 0. As usual this is combated by encoding Bob's input and instead using the encoding as input. The encoding is such that any s positions are uniformly random and independent of the input of Bob. Hence if Alice learns up to s bits of the encoding, it gives her no information on the input of Bob, and if she mounts more than s selective attacks, she will get caught except with probability 2^{-s}. This is again a known trick used in a white box manner in previous works, and again we use linking to generalize this technique to (reactive) garbling schemes. First, Alice will garble an identity function for which Bob will get an encoding of a randomly chosen input x' via OT. Then Bob selects a random hash function h from a two-universal family of hash functions such that $h(x') = x$ where x is Bob's real input. Bob sends h to Alice. Alice then garbles the hash function and links the output of the identity function to the input of the hash function and she links the output of the hash function to the encoded function which Bob is providing an input for.

With the above augmentations which solves obvious security problems, along with an augmentation described below, addressing a problem with simulation, the protocol is UC secure against a static adversary. We briefly sketch how to achieve simulation security.

Simulating corrupted Alice is easy. The simulator can cheat in the OTs used to set up the watchlist and learn both the randomness r_i and the input encodings of Alice in all the evaluation instances. The mechanisms described above ensure that in a majority of evaluation instances Alice correctly garbled and also used the same correct input encoding. Since the input encoding is projective, the input x of Alice can be computed from the input encoding function and her garbled input. By correctness of the garbling scheme, it follows that all correct evaluation instances would give the same output z consistent with x. Hence the simulator can use x as the input of Alice in the simulation.

As usual simulating corrupted Bob is more challenging. To get a feeling for the problem, assume that Alice has to send a garbled circuit F of the function f to be computed before Bob gives inputs. When Bob then gives input, the input y of Bob can be extracted in the simulation by cheating in the OTs and inspecting the choice bits used by Bob. The simulator then inputs y to the ideal functionality and gets back the output $z = f(x, y)$ that Bob is to learn. However, the simulator then in addition has to make F output z in the simulated execution of the protocol. This in general would require finding an input x' of Alice such that $z = f(x', y)$, which could be computationally hard. Previous papers have used white-box modifications of the garbled circuit or the output decoding function to facilitate enough cheating to make F hit z without having to compute x'. We show how to do it in a very simple and elegant way in a black-box manner from any reactive garbling scheme which can garble the exclusive-or function. In our protocol Alice will not send to Bob the decoding key for the encoded output Z. Instead, she garbles a masking function ($\psi(z, m) = z \oplus m$) and links the output of the function f to the first argument of the masking function. Then she produces an encoding M of the all-zero string for m and sends M to Bob along with the output decoding function for ψ. Bob can then compute and decode from Z and M the value $z \oplus 0 = z$. In the simulation, the simulator of corrupted Bob knows the watchlist and can hence behave honestly in the watchlist instances and use the freedom of m to make the output $z \oplus m$ hit the desired output from the ideal functionality in the evaluation positions. This will be indistinguishable from the real world because of the confidentiality property. Since this trick does not require modifying the garbled function, our protocol will only require a projective garbling scheme which is confidential. It will work for any side-information function. Earlier protocols required that the side-information be the topology of the circuit to hide the modification of the function f needed for simulation, or they needed to do white box modifications of the output decoding function to make the needed cheating occur as part of the output decoding.

5.1 Details of the Reactive 2PC Protocol

We now give more details on the protocol. The different instances will be indexed by $j \in I = \{1, \ldots, s\}$. The watchlist is given by $\boldsymbol{w} = (w_1, \ldots, w_s) \in \{0, 1\}^s$, where $w_j = 1$ iff j is a watchlist instance. In the protocol s instances are run in parallel. When a copy of a variable v is used in each instance, the copy used in instance j is denoted by v^j. In most cases the code for an instance does not depend on j explicitly but only on whether the instance is on the watchlist or the evaluation list, in which case we will write the code generically using the variable name v. The convention is that all s copies $v^1, \ldots, v^s$ are manipulated the same way, in single instruction multiple data program style. For instance, $w = 1$ will mean $w^j = 1$, such that $w = 1$ is true iff the instance is in the watchlist.

```
rule A.Initialize
// Sample watchlist key and an
   evaluation key
$\mathsf{wk}, \mathsf{ek} \xleftarrow{\$} \{0,1\}^k$
$\mathsf{OT.send}(\mathsf{ek}, \mathsf{wk})$
$\sigma \leftarrow ()$

rule A.Func
on $(\texttt{Func}, t, f)$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Func}, t, f)$

rule A.Garble
on $(\texttt{Garble}, t)$
await $\exists f$: $(\texttt{Func}, t, f) \in \sigma$
$(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(f, t; r)$
$E \leftarrow \mathsf{E}_{\mathsf{wk}}(r)$
send $F_t, E$ to B
$\sigma \leftarrow \sigma \,\|\, (\texttt{Garble}, t)$

rule A.Link
on $(\texttt{Link}, t_1, i_1, t, i_2)$
await $(\texttt{Garble}, t) \in \sigma$
await $(\texttt{Garble}, t_1) \in \sigma$
send $\mathsf{li}(o_{t_1,i_1}, e_{t,i_2})$ to B
$\sigma \leftarrow \sigma \,\|\, (\texttt{Link}, t_1, i_1, t, i_2)$
```

```
rule B.Initialize
// Learn either the watchlist
   key or the evaluation key
$w \xleftarrow{\$} \{0,1\}$
$k \leftarrow \mathsf{OT.choose}(w)$
$\sigma \leftarrow ()$

rule B.Func
on $(\texttt{Func}, t, f)$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Func}, t, f)$

rule B.Garble
on $(\texttt{Garble}, t)$
await $\exists f$: $(\texttt{Func}, t, f) \in \sigma$
on $F', E$ from A
if $w = 1$ then
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$
    $(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(f, t; r)$
    verify $F' = F_t$
$F_t \leftarrow F'$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Garble}, t)$

rule B.Link
on $(\texttt{Link}, t_1, i_1, t, i_2)$
await $(\texttt{Garble}, t) \in \sigma$
await $(\texttt{Garble}, t_1) \in \sigma$
on $\bar{L}$ from A
$\mathcal{L} \leftarrow \mathcal{L} \,\|\, (t_1, i_1, t, i_2, \bar{L})$
if $w = 1$ then verify
$\bar{L} = \mathsf{li}(o_{t_1,i_1}, e_{t,i_2})$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Link}, t_1, i_1, t, i_2)$
```

Fig. 10. Protocol (Initialize, Garble, Link)

We will use commitments and oblivious transfer within the protocol. We work in the OT hybrid model. We use $\mathsf{OT.send}(m_0, m_1)$ to mean that Alice sends two messages via the oblivious-transfer functionality and we use the notation $\mathsf{OT.choose}(b)$ to say that Bob chooses to receive m_b. We use a perfect binding and computationally hiding commitment scheme. If a public key is needed, it could be generated by Alice and sent to Bob in initialization. A commitment to a message m produced with randomness r is denoted by $\mathsf{com}(m; r)$, sending (m, r) constitutes an opening of the commitment.

If we write $A(x; r)$ for a randomized algorithm, where r is not bound before, then it means that we make a random run of A on input x and that we use r in the following to denote the randomness used by A. If we send a set $\{x, y\}$, then it is sent as a vector with the bit strings x and y sorted lexicographically, such that all information extra to the elements is removed before sending. When rules are called, tags t are provided. It follows from the input sequences being legal

```
rule $\mathsf{A}.\textsc{Input}_\mathsf{A}$
on $(\texttt{Input}, t, i, x)$
await $(\texttt{Garble}, t) \in \sigma$
$\bar{t} \leftarrow 1\|(\texttt{Input}, t, i)\|0$
$\ell_1 \leftarrow \text{len}(f_t.A_i)$
$\ell_2 \leftarrow \text{len}(g_{\ell_1}.A_2)$
$\ell_3 \leftarrow \text{len}(g_{\ell_1}.A_3)$
$m \xleftarrow{\$} \{0,1\}^{\ell_2}$
// Garble auxiliary function $g$
$(G_{\bar{t}}, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(g_{\ell_1}, \bar{t}; r)$
// Watchlist encryption of garbled auxilary function's randomness
$E \leftarrow E_{\mathsf{wk}}(r)$
send $(G_{\bar{t}}, d_{\bar{t},2}, E)$ to B
for $u \in \{1, \ldots, \ell_1\}$ do
    $X_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 0)$
    $X_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 1)$
    $r_{u,0}, r_{u,1} \xleftarrow{\$} \{0,1\}^k$
    // Commit to tokens
    $S_{u,1} \leftarrow \{\mathsf{com}(X_{u,0}; r_{u,0}), \mathsf{com}(X_{u,1}; r_{u,1})\}$
    // Watchlist encryption of tokens
    $E_{u,1} \leftarrow \mathsf{E}_{\mathsf{wk}}((X_{u,0}, X_{u,1}))$
    // Watchlist encryption of commitment's randomness
    $E_{u,2} \leftarrow \mathsf{E}_{\mathsf{wk}}((r_{u,0}, r_{u,1}))$
    // Evaluation encryption of tokens for Alice's choice of input
    $E_{u,3} \leftarrow \mathsf{E}_{\mathsf{ek}}((X_{u,x_{i,u}}, r_{u,x_{i,u}}))$
    // Linking $G$ to $F_t$
    $L_u \leftarrow \mathsf{li}(o_{\bar{t},1,u}, e_{t,i,u})$
    send $(S_{u,1}, E_{u,1}, E_{u,2}, E_{u,3}, L_u)$ to B
for $u \in \{1, \ldots, \ell_2\}$ do
    $M_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 0)$
    $M_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 1)$
    $r'_{u,0}, r'_{u,1} \xleftarrow{\$} \{0,1\}^k$
    $S_{u,2} \leftarrow \{\mathsf{com}(M_{u,0}; r'_{u,0}), \mathsf{com}(M_{u,1}; r'_{u,1})\}$
    $E_{u,4} \leftarrow \mathsf{E}_{\mathsf{wk}}((M_{u,0}, M_{u,1}))$
    $E_{u,5} \leftarrow \mathsf{E}_{\mathsf{wk}}((r'_{u,0}, r'_{u,1}))$
    $E_{u,6} \leftarrow \mathsf{E}_{\mathsf{ek}}((M_{u,m_{i,u}}, r'_{u,m_u}))$
    send $(S_{u,2}, E_{u,4}, E_{u,5}, E_{u,6})$ to B
// Auxiliary input from Bob
on $c$ from B
// Encoding of auxiliary input
for $u \in \{1, \ldots, \ell_3\}$ do send $C_{u,c_u}$ to B
$\sigma \leftarrow \sigma \,\|(\texttt{Input}, t, i, \top)$
```

Fig. 11. $\textsc{Input}_\mathsf{A}$

that these tags are unique, except when referring to a legal previous occurrence. We further assume that all tags provided as inputs are of the form $0\|\{0,1\}^*$, which allows us to use tags of the form $1\|\{0,1\}^*$ for internal book keeping. Tags

```
rule B.$\textsc{Input}_{\mathsf{A}}$
on $(\texttt{Input}, t, i, ?)$
await $(\texttt{Garble}, t) \in \sigma$
$\bar{t} \leftarrow 1 \| (\texttt{Input}, t, i) \| 0$
$c \xleftarrow{\$} \{0,1\}^{\ell_3}$
on $G'_{\bar{t}}, d'_{\bar{t},2}, E$ from $\mathsf{A}$
for $u \in \{1,\ldots,\ell_1\}$ do on $(S_{u,1}, E_{u,1}, E_{u,2}, E_{u,3}, L_u)$ from $\mathsf{A}$
for $u \in \{1,\ldots,\ell_2\}$ do on $(S_{u,2}, E_{u,4}, E_{u,5}, E_{u,6})$ from $\mathsf{A}$
send $c$ to $\mathsf{A}$
for $u \in \{1,\ldots,\ell_3\}$ do on $C_{u,c_u}$ from $\mathsf{A}$
// Use watchlist key to verify correctness of garbling and commitments.
if $w = 1$ then
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$, $(G_{\bar{t}}, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(g_{\ell_1}, \bar{t}; r)$
    for $u \in \{1,\ldots,\ell_1\}$ do
        $X_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 0)$
        $X_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 1)$
    for $u \in \{1,\ldots,\ell_2\}$ do
        $M_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 0)$
        $M_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 1)$
    for $u \in \{1,\ldots,\ell_3\}$ do
        $C_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},3,u}, 0)$
        $C_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},3,u}, 1)$
    for $u \in \{1,\ldots,\ell_1\}$ do
        $(r_{u,0}, r_{u,1}) \leftarrow \mathsf{D}_{\mathsf{wk}}(E_{u,2})$
        verify $\mathsf{D}_{\mathsf{wk}}(E_{u,1}) = (X_{u,0}, X_{u,1})$
        verify $S_{u,1} = \{\mathsf{com}(X_{u,0}; r_{u,0}), \mathsf{com}(X_{u,1}; r_{u,1})\}$
        verify $L_u = \mathsf{li}(o_{\bar{t},1,u}, e_{t,i,u})$
    for $u \in \{1,\ldots,\ell_2\}$ do
        $(r'_{u,0}, r'_{u,1}) \leftarrow \mathsf{D}_{\mathsf{wk}}(E_{u,5})$
        verify $\mathsf{D}_{\mathsf{wk}}(E_{u,3}) = (M_{u,0}, M_{u,1})$
        verify $S_{u,2} = \{\mathsf{com}(M_{u,0}; r_{u,0}), \mathsf{com}(M_{u,1}, r_{u,1})\}$
    for $u \in \{1,\ldots,\ell_3\}$ do
        verify $C_{u,c_u} = \mathsf{En}(e_{\bar{t},3,u}, c_u)$
else
// Use evaluation key to extract tokens for Alice's choice of input
    for $u \in \{1,\ldots,\ell_1\}$ do
        $(X_{u,x_{i,u}}, r_{u,x_{i,u}}) \leftarrow \mathsf{D}_{\mathsf{ek}}(E_{u,3})$
    for $u \in \{1,\ldots,\ell_2\}$ do
        $(M_{u,x_{i,u}}, r'_{u,x_{i,u}}) \leftarrow \mathsf{D}_{\mathsf{ek}}(E_{u,6})$
    // Verify commitments of tokens for Alice's choice of input
    verify $\forall u \in \{1,\ldots,\ell_1\}$ $(\mathsf{com}(X_{u,x_{i,u}}; r_{u,x_{i,u}}) \in S_{u,1})$
    verify $\forall u \in \{1,\ldots,\ell_2\}$ $(\mathsf{com}(M_{u,m_u}; r'_{u,m_u}) \in S_{u,2})$
    $\bar{\mathcal{X}} \leftarrow \{(\bar{t}, 1, X_x), (\bar{t}, 2, M_m), (\bar{t}, 3, C_c)\}$
    $\bar{Y} \leftarrow \mathsf{Ev}(\{(\bar{t}, G_{\bar{t}})\}, \bar{\mathcal{X}})$
    $y_2 \leftarrow \mathsf{De}(d_2, \bar{Y}_2)$
    // Verify that auxiliary outputs are the same in each instance
    verify $\forall j, j'$ $(y_2^j = y_2^{j'})$
    $\mathcal{X} \leftarrow \mathcal{X} \| \bar{\mathcal{X}}$
    $\mathcal{F} \leftarrow \mathcal{F} \| (\bar{t}, G_{\bar{t}})$
    $\mathcal{L} \leftarrow \mathcal{L} \| (\bar{t}, 1, t, i, L)$
    $\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, \top)$
```

Fig. 12. $\textsc{Input}_{\mathsf{A}}$ (continued)

for internal use will be derived from the tags given as input and the name of the rule creating the new tag. For a garbling scheme $\mathcal{G}$, a commitment scheme com and an encryption scheme $\mathcal{E}$, we use $\pi_{\mathcal{G},\mathsf{com},\mathcal{E}}$ to denote protocol given by the set of rules in Figures 10 to 15. We add a few remarks to the figures.

```
rule A.$\textsc{Input}_{\mathsf{B}}$
on $(\texttt{Input}, t, i, ?)$
await $(\texttt{Garble}, t) \in \sigma$
$\ell \leftarrow \mathrm{len}(f_t.A_i)$
$\ell_1 \leftarrow \ell + 2s + 1$
$\bar{t} \leftarrow 1 \| (\texttt{Input}, t, i) \| 0$
$t' \leftarrow 1 \| (\texttt{Input}, t, i) \| 1$
// Garble the identity function
$(\mathsf{Id}_{\bar{t}}, e_{\bar{t}}, o_{\bar{t}}, d_{\bar{t}}) \leftarrow \mathsf{Gb}(\mathsf{id}_{\ell_1}, \bar{t}; r)$
// Send to Bob the garbled identity function and the watchlist
   encryption of its randomness to Bob
send $E \leftarrow \mathsf{E}_{\mathsf{wk}}(r), \mathsf{Id}_{\bar{t}}$ to B
for $u \in \{1, \ldots, \ell_1\}$ do
    $X_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},u}, 0)$
    $X_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},u}, 1)$
    // Oblivious Transfer of Bob's input tokens
    $\mathsf{OT.send}(\{X^j_{u,0}\}_{j \in \{1,\ldots,s\}}, \{X^j_{u,1}\}_{j \in \{1,\ldots,s\}})$
// Await universal hash function
on $h$ from B
// Garble universal hash function
$(H_{t'}, e_{t'}, o_{t'}, d_{t'}) \leftarrow \mathsf{Gb}(h, t'; r')$
// Send garbled hash function and the watchlist encryption of its
   randomness to Bob
send $H_{t'}, \mathsf{E}_{\mathsf{wk}}(r')$ to B
for $u \in \{1, \ldots, \ell_1\}$ do
    // Link $\mathsf{Id}_{\bar{t}}$ to $H_{t'}$
    send $\bar{L}_u \leftarrow \mathsf{li}(o_{\bar{t},u}, e_{t',u})$ to B
    // Link $H_{t'}$ to $F_t$
    send $L_u \leftarrow \mathsf{li}(o_{t',u}, e_{t,i,u})$ to B
$\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, \top)$
```

Fig. 13. $\textsc{Input}_{\mathsf{B}}$

In the INITIALIZE-rules Alice and Bob setup the watchlist. They use a (symmetric) encryption scheme $\mathcal{E} = (\mathsf{E}, \mathsf{D})$ with k-bit keys. For each instance j, Alice sends two keys via the oblivious transfer functionality, the watchlist key wk^j and the evaluation key ek^j. Alice will later encrypt and send the information Bob is to learn for watchlist (evaluation) instances with the key wk (ek). In the FUNC-rules they simply associate a function to a tag. In the GARBLE-rules Alice garbles the function and sends the garbling to Bob, she also sends an encryption using the watchlist key of the randomness used to produce this garbling. This allows Bob, for the watchlist positions to check that Alice produced a correct garbling and to store the result of garbling. This knowledge will be used in other rules. In the LINK-rules Alice sends linking information. Bob can for all watchlist positions check that the information is correct, since he knows the randomness used to garble. In the OUTPUT-rules Alice awaits that she has sent to Bob the encoded inputs and linkings to produce the encoded output associated to this rule. She produces a garbling of ψ. She will link the output to ψ and produce an encoding of the zero-string for the second component, she also sends an encryption of the randomness used to produce the garbling of ψ to Bob. Bob awaits that he has received the garbling, linking and encoding to produce the encoded output in question. For each instance of the watchlist, he uses the randomness to check that the linking was done correctly, that ψ was garbled correctly and that an encoding of an all zero-string was sent for the second component of ψ. He then evaluates each instance in the evaluation set and takes the majority value as his output.

In the $\texttt{Input}_{\mathsf{A}}$-rules Alice commits to both her input encodings and encrypts the openings of the commitments using the watchlist key. The opening of Alice's input encoding will be encrypted using the evaluation key. To verify Alice's input, we first pass Alice's input through an auxiliary function which combines the identity function with an additional verification function which forces Alice to use the same input in different instances. We then link the output of the identity function to the appropriate input. We denoted the auxiliary function by $g_l : A_1 \times A_2 \times A_3 \to B_1 \times B_2$ and $g_\ell(x, m, c) = (x, v_\ell(x, m, c))$ where $A_1 = A_2 = B_1 = \{0,1\}^\ell \cup \{\perp\}$ and $v_\ell : A_1 \times A_2 \times A_3 \to B_2$. Efficient such functions with the properties needed for the security of the protocol can be based on universal hash functions, see for instance [25,7].

In the $\texttt{Input}_{\mathsf{B}}$-rules Alice first garbles the identity function. Bob then randomly samples a value x' and gets an encoding of that value via oblivious transfer for the garbled identity function. Then Bob samples uniformly at random a function h from a two-universal family of hash functions such that $h(x') = x$ where x is the input of Bob. Alice will then garble the hash function. She will link the garbling of the identity function to the garbling of the hash function. She will then link the garbled hash function to the garbled function. We will denote by $\mathcal{H}_\ell$ a two-universal family of hash functions $h : \{0,1\}^{\ell+2s+1} \to \{0,1\}^\ell$. We use $\mathsf{id} : A \to A$ to denote the identify function on A.

In Appendix A, we prove the following theorem.

Theorem 2. *Let $\mathbb{L}$ be the set of all legal sequences and let Φ be a side-information function. Let $\mathcal{G}$ be a reactive garbling scheme. Let* com *be a commitment scheme and $\mathcal{E}$ an encryption scheme. If $\mathcal{G}$ is $\mathbb{L}$-correct and $(\mathbb{L}, \Phi)$-confidential and* com *is computationally hiding and perfect binding and $\mathcal{E}$ is IND-CPA secure, then $\pi_{\mathcal{G},\mathsf{com},\mathcal{E}}$ UC securely realizes $\mathcal{F}_{\mathrm{R2PC}}^{\mathbb{L},\Phi}$ in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model against a static, malicious adversary.*

References

1. M. Abdalla and R. D. Prisco, editors. *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*. Springer, 2014.
2. M. Bellare, V. T. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153. Springer, 2012.

```
rule B.INPUT_B
on $(\texttt{Input}, t, i, x)$
await $(\texttt{Garble}, t) \in \sigma$
$\bar{t} \leftarrow 1\|(\texttt{Input}, t, i)\|0$
$t' \leftarrow 1\|(\texttt{Input}, t, i)\|1$
// sample a random string $\bar{x}$
$\bar{x} \overset{\$}{\leftarrow} \{0,1\}^{\ell_1}$
// Sample a random universal hash function $h$ such that $h(\bar{x}) = x$
$h \overset{\$}{\leftarrow} \{\ \bar{h} \in \mathcal{H}_\ell \mid \bar{h}(\bar{x}) = x\ \}$
// Await a garbled identity function from Alice
on $E, \mathsf{Id}'_{\bar{t}}$ from A
// Obliviously learn tokens for $\bar{x}$
for $u \in \{1, \ldots, \ell_1\}$ do
    $\{\bar{X}^j_{u,\bar{x}_u}\}_{j \in \{1,\ldots,s\}} \leftarrow \mathsf{OT.choose}(\bar{x}_u)$
$\bar{X}_{\bar{t},\bar{x}} \leftarrow (\bar{X}_{1,\bar{x}_1}, \ldots, \bar{X}_{\ell_1,\bar{x}_{\ell_1}})$
if $w = 1$ then
    /* Verify garbled identity function and the correctness of
       received tokens using the watchlist encryption of the
       randomness                                                  */
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$
    $(\mathsf{Id}_{\bar{t}}, e_{\bar{t}}, o_{\bar{t}}, d_{\bar{t}}) \leftarrow \mathsf{Gb}(\mathsf{id}_{\ell_1}, \bar{t}; r)$
    verify $\mathsf{Id}_{\bar{t}} = \mathsf{Id}'_{\bar{t}}$
    verify $\forall u \in \{1, \ldots, \ell_1\}\ :\ \bar{X}_{\bar{t},u} = \mathsf{En}(e_{\bar{t},u}, \bar{x}_u)$
else
    $\mathcal{X} \leftarrow \mathcal{X}\,\|(\bar{t}, \bar{X}_{\bar{t},\bar{x}})$
send $h$ to A
on $H', E'$ from A
for $u \in \{1, \ldots, \ell_1\}$ do
    on $\bar{L}_u$ from A
    on $L_u$ from A
if $w = 1$ then
    /* Verify garbled hash function using the watchlist encryption
       of the randomness                                           */
    $r' \leftarrow \mathsf{D}_{\mathsf{wk}}(E')$
    $(H_{t'}, e_{t'}, o_{t'}, d_{t'}) \leftarrow \mathsf{Gb}(h, t'; r')$
    verify $H_{t'} = H'$
    // Verify linking information
    for $u \in \{1, \ldots, \ell_1\}$ do
        verify $\bar{L}_u = \mathsf{li}(o_{\bar{t},u}, e_{t',u})$
        verify $L_u = \mathsf{li}(o_{t',u}, e_{t,i,u})$
else
    $\mathcal{F} \leftarrow \mathcal{F}\,\|(\bar{t}, Id)$
    $\mathcal{F} \leftarrow \mathcal{F}\,\|(t', H)$
    $\mathcal{X} \leftarrow \mathcal{X}\,\|(\bar{t}, \bar{X}_{\bar{t},\bar{x}})$
    $\mathcal{L} \leftarrow \mathcal{L}\,\|(t', 1, t, i, L)$
    for $u \in \{1, \ldots, \ell_1\}$ do
        $\mathcal{L} \leftarrow \mathcal{L}\,\|(\bar{t}, u, t', u, \bar{L}_u)$
$\sigma \leftarrow \sigma\,\|(\texttt{Input}, t, i, \top)$
```

Fig. 14. INPUT$_{\mathsf{B}}$ (continued)

```
rule A.$\textsc{Output}$
on $(\texttt{Output}, t, i)$
await $(t, i) \in \mathsf{ready}(\sigma)$
$\bar{t} \leftarrow 1 \| (\texttt{Output}, t, i)$
// Garble $\psi$
$(\Psi, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(\psi, \bar{t}; r)$
$L \leftarrow \mathsf{li}(o_{t,i}, e_{\bar{t},1})$
$E \leftarrow \mathsf{E}_{\mathsf{wk}}(r)$
// Encode all zero-string
$X_{\bar{t},0} \leftarrow \mathsf{En}(e_{\bar{t},2}, 0)$
send $(L, E, \Psi, X_{\bar{t},0}, d_{\bar{t}})$ to B

rule B.$\textsc{Output}$
on $(\texttt{Output}, t, i)$
await $(t, i, \top) \in \mathsf{ready}(\sigma)$
$\bar{t} \leftarrow 1 \| (\texttt{Output}, t, i)$
on $(\bar{L}, \bar{E}, \bar{\Psi}, \bar{X}_{\bar{t},0}, \bar{d}_{\bar{t}})$ from A
if $w = 1$ then
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(\bar{E})$
    $(\Psi, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(\psi, \bar{t}; r)$
    $L \leftarrow \mathsf{li}(o_{t,i}, e_{\bar{t},1})$
    /* Verify:
    1) $\bar{\Psi}$ is the garbling of $\psi$
    2) Linking is correct
    3) Encoding of the all zero-string was sent
    4) Correct output decoding was sent                        */
    verify $\bar{L} = L \wedge \bar{\Psi} = \Psi$
    verify $\bar{X}_{\bar{t},0} = \mathsf{En}(e_{\bar{t},2}, 0) \wedge \bar{d}_{\bar{t},1} = d_{\bar{t},1}$
else
    $\mathcal{F} \leftarrow \mathcal{F} \| (\bar{t}, \Psi)$
    $\mathcal{X} \leftarrow \mathcal{X} \| (\bar{t}, 2, \bar{X}_{\bar{t},0})$
    $\mathcal{L} \leftarrow \mathcal{L} \| (t, i, \bar{t}, 1, \bar{L})$
    $\delta \leftarrow \delta \| (\bar{t}, 1, \bar{d}_{\bar{t},1})$
    await $\exists (\bar{t}, 1, Y_{t,1}) \in \mathsf{Ev}(\mathcal{F}, \mathcal{X}, \mathcal{L})$
    $y^j_{t,i} \leftarrow \mathsf{De}(\bar{d}_{\bar{t},1}, Y_{\bar{t},1})$
    // Apply majority decoding
    $y_{t,i} \leftarrow \mathsf{maj}(y^1_{t,i}, \ldots, y^1_{t,i})$
```

Fig. 15. Protocol ($\textsc{Output}$)

3. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.
4. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.
5. L. T. 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.
6. H. Carter, C. Lever, and P. Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. 2014.
7. T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen. Faster maliciously secure two-party computation using the GPU. In Abdalla and Prisco [1], pages 358–379.
8. T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, volume 7881, pages 537–556. Springer, 2013.
9. T. K. Frederiksen and J. B. Nielsen. Fast and maliciously secure two-party computation using the GPU. *IACR Cryptology ePrint Archive*, 2013:46, 2013.
10. J. A. Garay and R. Gennaro, editors. *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*. Springer, 2014.
11. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.
12. C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled RAM revisited. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2014.
13. C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology–CRYPTO 2010*, pages 155–172. Springer, 2010.
14. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *Advances in Cryptology–CRYPTO 2008*, pages 39–56. Springer, 2008.
15. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.
16. Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In Garay and Gennaro [10], pages 458–475.
17. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.
18. B. Kreuter, abhi shelat, and C. hao Shen. Billion-gate secure computation with malicious adversaries. Cryptology ePrint Archive, Report 2012/179, 2012. http://eprint.iacr.org/.
19. B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, pages 285–300, 2012.
20. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

21. Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
22. S. Lu and R. Ostrovsky. How to garble RAM programs. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 719–734. Springer, 2013.
23. B. Mood, D. Gupta, K. Butler, and J. Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 582–596. ACM, 2014.
24. J. B. Nielsen and C. Orlandi. Lego for two-party secure computation. In *Theory of Cryptography*, pages 368–386. Springer, 2009.
25. A. Shelat and C. Shen. Fast two-party secure computation with minimal assumptions. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 523–534, 2013.

A Proof of Theorem 2

In this section, we analyse the protocol. In the analysis, we use that the auxiliary function by $g_l : A_1 \times A_2 \times A_3 \to B_1 \times B_2$ and $g_\ell(x, m, c) = (x, v_\ell(x, m, c))$ where $A_1 = A_2 = B_1 = \{0, 1\}^\ell \cup \{\perp\}$ and $v_\ell : A_1 \times A_2 \times A_3 \to B_2$ has the following properties:

Statistical Binding $\forall x, y, m, m' \in A_1$
$x \neq x' \Rightarrow \mathsf{Pr}[v(x, m, c) = v(x', m', c') \mid c \xleftarrow{\$} A_3] \leq 2^{-s}$
Perfect Hiding $\forall x, x' \in A_1, c \in A_3, d \in B_2$
$\mathsf{Pr}[v(x, m, c) = d \mid m \xleftarrow{\$} A_2] - \mathsf{Pr}[v(x', m', c) = d \mid m' \xleftarrow{\$} A_2] = 0$

Efficient such functions can be based on universal hash functions, see for instance [25,7].

A.1 Legality

We will demonstrate that if the sequence input to the protocol is legal, then the sequence of garbling commands used by the protocol is legal too. We assume that we work with $\mathbb{L}$ being the set of all legal garbling commands.

Note that by assumption, the garbling sequence provided to both players is a legal garbling sequence. We have to show that the garbling sequence (σ) associated to the invocation of the rules results in a legal garbling sequence. The legality of the sequence is analysed casing on each required property.

Function uniqueness Since the sequence given to both players is legal and the tags are produced in a way that avoids collisions, the associated sequence will only have one command of the form $(\texttt{Func}, t, \cdot)$.
Garble uniqueness Since the sequence given to both players is legal and the tags are produced in a way that avoids collisions, the associated sequence will only have one command of the form $(\texttt{Garble}, t)$.
Linkage legality Since, the sequence given to both players is legal and that we ensure by construction that we only allow linking between components of the same length. In addition, linking rule only proceeds after the associated garbling have terminated. We can therefore deduce that linkage legality holds.
Input legality The same reasoning to show linkage legality also applies to input legality.
Garble legality Since the garble rule can only proceed after the $(\texttt{Func}, t, \cdot)$ has become part of the sequence, for all tags within the sequence, if $(\texttt{Garble}, t, \cdot)$ occurs in σ then it is preceded by $(\texttt{Func}, t, \cdot)$. In the case where we garble functions within the context of the protocol, we can see that each garbling uses a tag which is unique to its rule. We can therefore deduce that garble legality holds.
Output legality If $(\texttt{Output}, t, i)$ occurs in a sequence, it is preceded by $(\texttt{Garble}, t)$ because the output rule cannot proceed until $(\texttt{Garble}, t)$ is part of the sequence.
Input uniqueness This holds since the sequence is legal by assumption.

A.2 Corrupted Alice

We prove security for the case where Alice is corrupted. Since corrupted Alice is controlled by the environment, we will often use the term *environment* to mean *corrupted Alice.* The simulator will essentially run the protocol as an honest Bob would but with random inputs. The only additional thing that he will do is to extract the input of corrupted Alice. This he will be able to do because he will have access to all of the evaluation keys and all the watchlist keys.

The simulator in the initialization phase selects a random watchlist. For each element of the watchlist, the simulator aborts if the randomness committed to by the environment does not produce the encoded functions or the linking sent by the environment, as in the protocol.

For the output phase, for each element of the watchlist, if the garbling produced for ψ is incorrect, if an encoding of a non zero-string is sent or a different decoding is sent instead of the one produced during the garbling procedure with the specified randomness, the simulator aborts, as in the protocol.

For Alice's input phase, for the watchlist, the simulator checks that all the values sent by the environment are valid (relative to the specification of the protocol and the randomness sent by the environment), otherwise the simulator aborts. The simulator select a random input c for the validation function, checks that the output of the validation phase is consistent. If not it aborts. Otherwise, the simulator can compute the environment's input by using both the evaluation keys and the watchlist keys to see which encoded inputs did the environment open for evaluation indices. This is done by having the simulator take the majority input over all indices. The simulator will then forward the command $(\texttt{Input}, t, i, x)$ to the ideal functionality where x is the value the simulator extracted.

For Bob's input phase, the simulator selects a random x', h'. For the watchlist, the simulator checks that all the values sent by the environment and received by the OT for the choice of x' are valid otherwise abort. Otherwise, for the evaluation indices, run the protocol with random x' and h' and abort if given the chosen watchlist Bob would have aborted. Store the result and send $(\texttt{Input}, t, i, ?)$ to the ideal functionality.

The simulator is formally specified in Figures 16 to 18. We now proceed to analyse the simulator.

We call an instance j *correct* if all the garblings, linkings and output decoding functions sent by Alice were computed correctly from the randomness r obtained by decrypting under the evaluation key ek_j, and if in addition all the input encodings of Alice's input bits encrypted under the evaluation key were correct, i.e., a correct encoding of either 0 or 1, as computed by the input encoding functions which by definition were correctly generated for that instance. We use C to denote the set of correct instances. We use W to denote the set of watchlist instances. We use E to denote the set of evaluation instances.

For each correct instance j and each input component (t, i) Alice has a well-defined input, as computed in line 5 of Figure 18. We say that Alice has *consistent* *inputs* if it holds at all points at which Bob did not abort that the input of Alice in all instances $j \in C \cap E$ are the same.

The only difference between the simulation and the protocol is that in the simulation, Bob uses dummy (incorrect) inputs as opposed to his real inputs and outputs the value from the ideal functionality as opposed to the value he computes in the protocol. Also, there might be a difference in the probability that Bob aborts because of a failed check. The following three lemmas can be proven using standard techniques.

Lemma 2. *The probability that Bob aborts does not depend on the inputs of Bob, except for a probability mass negligible in s.*

Lemma 3. *It holds except with negligible probability in s that that Alice has consistent inputs.*

Lemma 4. *It holds that $|C \cap E| > |E \setminus C|$, except with negligible probability in s.*

We first argue that security follows from these lemmas and then sketch how to prove them. By the first of the lemmas it is sufficient to prove that the protocol and the simulation are indistinguishable up to the point where Bob aborts. Assume then that Bob does not abort. In that case the view the environment has of Bob is view is the outputs of Bob and the hash functions h sent to Alice. It is easy to prove that h does not depend on the inputs of Bob [21]. Hence we only need to argue that the outputs of Bob are indistinguishable in the two cases. By correctness, we know that all the instances $j \in C \cap E$ in the protocol would output values consistent with the inputs of Bob and the inputs extracted for Alice in those instances. This is because $j \in C \cap E$ implies that the garblings and linkings were done correctly and that Bob received the correct encodings for his inputs and Alice supplied correct encodings for her inputs. From the second lemma it then follows that all the instances $j \in C \cap E$ in the protocol would output values consistent with the inputs of Bob and the *consistent* inputs extracted for Alice in those instances. From $|C \cap E| > |E \setminus C|$ it then follows that it is the inputs of Alice in the instances $j \in C \cap E$ that becomes the values sent to the ideal functionality on behalf of corrupted Alice in the simulation. From $|C \cap E| > |E \setminus C|$ it also follows that it is the consistent outputs of the instances $j \in C \cap E$ that become the output that Bob uses in the protocol. From this it follows that the outputs are the same in the protocol and the simulation.

The proof of Lemma 2 follows from the properties of the hash function h used using a by now standard argument, see for instance [21]. All other abort probabilities are clearly independent of the inputs of Bob. The proof of Lemma 2 follows from the properties of the auxiliary function, specifically the statistical binding. Namely, if the environment uses inconsistent inputs in otherwise correct evaluation instances, then that instance will correctly evaluate the auxiliary function (by correctness of the garbling scheme) and will therefore provide Bob with different check values except with negligible probability, by the statistical binding property. See for instance [25,7] for the details. The proof of Lemma 4 follows from the fact that if an instance $j \notin C$ becomes a watchlist instance, then Bob will abort except with negligible probability, as detailed below. Hence by setting s large enough we can ensure that if Bob does not abort, then less than half of the evaluation positions are correct except with probability $2^{-O(s)}$. To see that if an incorrect position becomes a watchlist position, then Bob aborts, except with negligible probability, observe that if a garbling, linking or output decoding function is incorrect, then clearly Bob detects. Assume then that Alice encrypted an incorrect input encoding under the evaluation key. Recall that this input encoding is encrypted together with an opening of one of the two commitments to the input encodings of 0 and 1. From this and the perfect binding of the commitment scheme, it follows that when Bob checks the encodings inside the commitments (he can do this on watchlist positions), he will see the same incorrect encoding, and abort.

A.3 Corrupted Bob

We now consider the case where Bob is corrupt. We use environment to designate the entity controlling the corrupted Bob. The idea of the simulator is that it will follow the description of an honest Alice except in four places. In the input rule for Bob, the simulator will also extract the environment's input without deviating from the behaviour of a real world Alice. It just inspects the choice bits of the OTs simulated by the simulator. In the input rule for Alice, it will follow the behaviour of Alice with an all-zero string. Only in the output phase will the simulate differ behaviourally from a real world Alice. In the output phase, the simulator will learn the output from the ideal functionality and will then in the simulated protocol send a correction string (instead) of the all-zero string, to make it the value from the ideal functionality. During the function, garbling and linking phases, the simulator will follow the behaviour specified in the protocol. We now describe the deviations from the real protocol in a little more detail.

During Alice's input phase, the simulator will select value 0^ℓ (ℓ is the size of the input component in question) for Alice and follow the description of the protocol with this value. It will send $(\texttt{Input}, t, i, 0^\ell)$ to the ideal functionality.

During Bob's input phase, the simulator follows the description of the protocol. The simulator will extract the environment's input x via the choice of Bob to the oblivious transfer and the hash functions. It then send $(\texttt{Input}, t, i, x)$ to the ideal functionality.

During the output phase, the simulator will deviate from the protocol. It will call the functionality with the output rule and receive an output y. The simulator also computes the output that Bob receives in the simulated protocol, call it y'. Note the most likely $y' \neq y$ as in the simulated protocol, Alice was run with the dummy inputs 0^ℓ which are most likely different from her real inputs. The simulator will then compute $\Delta \leftarrow y \oplus y'$. For each watchlist index, he will send encodings of 0^ℓ where ℓ denotes the length of the output component in question. For the evaluation indices, he will send encodings of Δ. As a result,

```
rule INITIALIZE
on $(\texttt{Send}, \mathsf{ek}, \mathsf{wk})$ from Env
$w \xleftarrow{\$} \{0,1\}$
// Simulator learns both
   evaluation and watchlist key
store $\mathsf{ek}, \mathsf{wk}$
$\sigma \leftarrow \{\}$

rule GARBLE
on $(\texttt{Garble}, t)$ from Env
await $\exists f$: $(\texttt{Func}, t, f) \in \sigma$
on $F', E$ from Env
if $w = 1$ then
    $r_t \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$
    $(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(f, t; r)$
    verify $F' = F_t$
send $(\texttt{Garble}, t)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Garble}, t, \texttt{done})$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Garble}, t, \texttt{done})$ to Env
$\sigma \leftarrow \sigma \,\|\, (\texttt{Garble}, t)$

rule OUTPUT
on $(\texttt{Output}, t, i)$ from Env
await $(t, i, \top) \in \mathsf{ready}(\sigma)$
on $(\bar{L}, \bar{E}, \bar{\Psi}, \bar{X}_{\bar{t},0}, \bar{d}_{\bar{t}})$ from Env
if $w = 1$ then
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(\bar{E})$
    $(\Psi, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(\psi, \bar{t}; r)$
    verify $\bar{\Psi} = \Psi$
    verify $\bar{X}_{\bar{t},0} = \mathsf{En}(e_{\bar{t},2}, 0)$
    verify $\bar{d}_{\bar{t}} = d_{\bar{t}}$
    verify $\bar{L} = \mathsf{li}(o_{t,i}, e_{\bar{t},1})$
send $(\texttt{Output}, t, i)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Output}, t, i, \texttt{done})$ from Env
send $(\texttt{Output}, t, i, \texttt{done})$ to Env

rule FUNC
on $(\texttt{Func}, t, f)$ from Env
send $(\texttt{Func}, t, f)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Func}, t, \texttt{done})$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Func}, t, \texttt{done})$ to Env
$\sigma \leftarrow \sigma \,\|\, (\texttt{Func}, t, f)$

rule LINK
on $(\texttt{Link}, t_1, i_1, t, i_2)$ from Env
await $(\texttt{Garble}, t_1) \in \sigma$
await $(\texttt{Garble}, t) \in \sigma$
on $L$ from Env
$\mathcal{L} \leftarrow \mathcal{L} \,\|\, (t_1, i_1, t, i_2, L)$
if $w = 1$ then
    verify $L = Li(o_{t_1,i_1}, e_{t,i_2})$
send $(\texttt{Link}, t_1, i_1, t, i_2)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Link}, t_1, i_1, t, i_2, \texttt{done})$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Link}, t_1, i_1, t, i_2, \texttt{done})$ to Env
$\sigma \leftarrow \sigma \,\|\, (\texttt{Link}, t_1, i_1, t, i_2)$
```

Fig. 16. Alice Corrupt: INITIALIZE, FUNC, GARBLE, LINK, OUTPUT

the simulated evaluation will compute the same output as the one given by the functionality.

A.4 Indistinguishability

We will show that if the environment can distinguish between the real and ideal world then it can break the confidentiality property of a reactive garbling scheme or the computational hiding of the commitment scheme or the IND-CPA security of the encryption scheme.

Notice that aside from INPUT$_{\mathsf{A}}$, INPUT$_{\mathsf{B}}$ and the OUTPUT phase, the simulator follows exactly the steps that an honest Alice would. In those steps, for the

```
rule $\textsc{Input}_\mathsf{B}$
on $(\texttt{Input}, t, i, ?)$ from $\mathsf{Env}$
await $(\texttt{Garble}, t) \in \sigma$
send $(t, i, ?)$ to $\mathcal{F}_{\text{R2PC}}$
$\ell \leftarrow \text{len}(f_t.A_i)$
$\ell_1 \leftarrow 2(\ell + s)$
$\bar{t} \leftarrow 1\|(\texttt{Input}, t, i)\|0$
$t' \leftarrow 1\|(\texttt{Input}, t, i)\|1$
on $E, \mathsf{Id}'$ from $\mathsf{Env}$
if $w = 1$ then
    $r \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$
    $(\mathsf{Id}_{\bar{t}}, e_{\bar{t}}, o_{\bar{t}}, d_{\bar{t}}) \leftarrow \mathsf{Gb}(\mathsf{id}_{\ell_1}, \bar{t}; r)$
    verify $\mathsf{Id}' = \mathsf{Id}_{\bar{t}}$
for $u \in \{1, \ldots, \ell_1\}$ do
    on $(\texttt{Send}, \bar{X}_{\bar{t},u,0}, \bar{X}_{\bar{t},u,1})$ from $\mathsf{Env}$
$\bar{x} \xleftarrow{\$} \{0,1\}^{\ell_1}$
$h \xleftarrow{\$} H$                                   // Random two-universal hash function
$\bar{X}_{\bar{t},\bar{x}} \leftarrow (\bar{X}_{1,\bar{x}_1}, \ldots, \bar{X}_{l,\bar{x}_l})$
if $w = 1$ then
    verify $\forall u, \bar{X}_{\bar{t},u,x_u} = \mathsf{En}(e_{\bar{t},u}, x_u)$
send $h$ to $\mathsf{Env}$
on $E', H'$ from $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_1\}$ do
    on $\bar{L}_u$ from $\mathsf{Env}$
    on $L_u$ from $\mathsf{Env}$
if $w = 1$ then
    $r' \leftarrow \mathsf{D}_{\mathsf{wk}}(E')$
    $(H_{t'}, e_{t'}, o_{t'}, d_{t'}) \leftarrow \mathsf{Gb}(h, t'; r')$
    verify $H_{t'} = H'$
    for $u \in \{1, \ldots, \ell_1\}$ do
        verify $\bar{L}_u = \mathsf{li}(o_{\bar{t},u}, e_{t',u})$
        verify $L_u = \mathsf{li}(o_{t',u}, e_{t,i,u})$
send $(\texttt{Input}, t, i, ?)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Input}, t, i, done)$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Input}, t, i, done)$ to $\mathsf{Env}$
$\sigma \leftarrow \sigma\,\|(\texttt{Input}, t, i, \top)$
```

Fig. 17. Alice corrupt: $\textsc{Input}_\mathsf{B}$

watchlist positions, the distribution of what the environment sees is the same in both worlds. Since the watchlist in both the ideal and real world have the same distribution, the watchlist cannot help the environment distinguish between the real and ideal world. We thus only need to look at what changes for the non-watchlist positions. In $\textsc{Input}_\mathsf{A}$, the simulator uses the encoding of the all-zero string. In $\textsc{Input}_\mathsf{B}$, he extracts the environments choice of input via the choice of hash function and the input to the oblivious oblivious transfer. In the output phase, it uses Δ to correct for an output from the ideal evaluation different from the one in the simulated execution.

```
rule INPUT_A
on $(\texttt{Input}, t, i, z)$ from Env
await $(\texttt{Garble}, t) \in \sigma$
send $(\texttt{Input}, t, i, z)$ to $\mathcal{F}_{\mathsf{R2PC}}$
$\bar{t} \leftarrow 1 \| (\texttt{Input}, t, i) \| 0$
$\ell_1 \leftarrow \mathrm{len}(f_t.A_i)$
$\ell_2 \leftarrow \mathrm{len}(g_\ell.A_2)$
$\ell_3 \leftarrow \mathrm{len}(g_\ell.A_3)$
on $E$ from Env
on $(G', d'_2)$ from Env
for $u \in \{1, \dots, \ell_1\}$ do  on $(S_{u,1}, E_{u,1}, E_{u,2}, E_{u,3}, L_u)$ from Env
for $u \in \{1, \dots, \ell_2\}$ do  on $(S_{u,2}, E_{u,4}, E_{u,5}, E_{u,6})$ from Env
$c \xleftarrow{\$} \{0,1\}^{\ell_3}$
send $c$ to Env
for $u \in \{1, \dots, \ell_3\}$ do  on $C_{u,c_u}$ from Env
$r \leftarrow \mathsf{D}_{\mathsf{wk}}(E)$
$(G_{\bar{t}}, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(g_{\ell_1}, \bar{t}; r)$
for $u \in \{1, \dots, \ell_1\}$ do  $X_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 0)$, $X_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 1)$
for $u \in \{1, \dots, \ell_2\}$ do  $M_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 0)$, $M_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 1)$
if $w = 1$ then
    verify $(G', d'_2) = (G_{\bar{t}}, d_{\bar{t},2})$
    for $u \in \{1, \dots, \ell_1\}$ do
        $(r_{u,0}, r_{u,1}) \leftarrow \mathsf{D}_{\mathsf{wk}}(E_{u,2})$
        verify $\mathsf{D}_{\mathsf{wk}}(E_{u,1}) = (X_{u,0}, X_{u,1})$
        verify $S_{u,1} = \{\mathsf{com}(X_{u,0}, r_{u,0}), \mathsf{com}(X_{u,1}, r_{u,1})\}$
        verify $L_u = \mathsf{li}(o_{\bar{t},1,u}, e_{t,i,u})$
    for $u \in \{1, \dots, \ell_2\}$ do
        $(r'_{u,0}, r'_{u,1}) \leftarrow \mathsf{D}_{\mathsf{wk}}(E_{u,5})$
        verify $\mathsf{D}_{\mathsf{wk}}(E_{u,3}) = (M_{u,0}, M_{u,1})$
        verify $S_{u,2} = \{\mathsf{com}(M_{u,0}, r_{u,0}), \mathsf{com}(M_{u,1}, r_{u,1})\}$
    for $u \in \{1, \dots, \ell_3\}$ do
        verify $C_{u,c_u} = \mathsf{com}(\mathsf{En}(e_{\bar{t},3,u}, c_u), r''_{u,c_u})$
else
    for $u \in \{1, \dots, \ell_1\}$ do  $(\bar{X}_u, r_u) \leftarrow \mathsf{D}_{\mathsf{ek}}(E_{u,3})$
    for $u \in \{1, \dots, \ell_2\}$ do  $(\bar{M}_u, r'_u) \leftarrow \mathsf{D}_{\mathsf{ek}}(E_{u,6})$
    verify $\forall u \in \{1, \dots, \ell_1\}$, $\mathsf{com}(\bar{X}_u, r_u) \in S_{u,1}$
    verify $\forall u \in \{1, \dots, \ell_2\}$, $\mathsf{com}(\bar{M}_u, r_u) \in S_{u,2}$
    $\bar{\mathcal{X}} \leftarrow \{(\bar{t}, 1, X_x), (\bar{t}, 2, M_m), (\bar{t}, 3, C_c)\}$, $\bar{Y} \leftarrow \mathsf{Ev}(\{(\bar{t}, G_{\bar{t}})\}, \bar{\mathcal{X}})$, $y_2 \leftarrow \mathsf{De}(d_2, \bar{Y}_2)$
    verify $\forall j, j'$, $y_2^{(j)} = y_2^{(j')}$
    // Use garbling with encrypted randomness to learn Alice's input
    for $u \in \{1, \dots, \ell_1\}$ do
        for $j \in \{1, \dots, s\}$ do
            if $\bar{X}_u^j = X_{u,0}^j$ then $x_u^j \leftarrow 0$
            else $x_u^j \leftarrow 1$
        $x_u \leftarrow \mathsf{maj}(x_u^1, \dots, x_u^s)$                    // Apply majority decoding
    $x \leftarrow x_1 \dots x_{\ell_1}$
    send $(\texttt{Input}, t, i, x)$ to $\mathcal{F}_{\mathsf{R2PC}}$
    on $(\texttt{Input}, t, i, \texttt{done})$ from $\mathcal{F}_{\mathsf{R2PC}}$
    send $(\texttt{Input}, t, i, \texttt{done})$ to Env
    $\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, \top)$
```

Fig. 18. Alice corrupt: INPUT_A

We argue indistinguishability using five hybrids.

In the first hybrid the simulator will in all evaluation positions change one of the committed values to a dummy value. It will correctly commit to the input encoding that it is going to send to B, but is going to commit to a dummy value in the other commitment, say the all-zero string. Since this commitment is never opened, the difference is indistinguishable by the computational hiding of the

```
rule INITIALIZE
$\mathsf{wk}, \mathsf{ek} \xleftarrow{\$} \{0,1\}^k$
// Simulator learns if
   evaluation instance or
   watchlist instance
on $(\texttt{Transfer}, w)$ from Env
if $w$ then $z \leftarrow \mathsf{wk}$
else $z \leftarrow \mathsf{ek}$
send $(\texttt{Transfered}, w, z)$ to Env

rule LINK
on $(\texttt{Link}, t_1, i_1, t, i_2)$ from Env
await $(\texttt{Garble}, t) \in \sigma$
await $(\texttt{Garble}, t_1) \in \sigma$
send $\mathsf{li}(o_{t_1,i_1}, e_{t,i_2})$ to Env
send $(\texttt{Link}, t_1, i_1, t, i_2)$ to $\mathcal{F}_{\text{R2PC}}$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Link}, t_1, i_1, t, i_2)$

rule FUNC
on $(\texttt{Func}, t, f)$ from Env
send $(\texttt{Func}, t, f)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Func}, t, f, \texttt{done})$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Func}, t, f, \texttt{done})$ to $\mathcal{F}_{\text{R2PC}}$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Func}, t, f)$

rule GARBLE
on $(\texttt{Garble}, t)$ from Env
await $(\texttt{Func}, t, f) \in \sigma$
$(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(f, t; r)$
send $F_t, \mathsf{E}_{\mathsf{wk}}(r)$ to Env
send $(\texttt{Garble}, t)$ to $\mathcal{F}_{\text{R2PC}}$
$\sigma \leftarrow \sigma \,\|\, (\texttt{Garble}, t)$

rule OUTPUT
on $(\texttt{Output}, t, i)$ from Env
await $(t, i, \top) \in \mathsf{access}(\mathsf{eval})(\sigma)$
send $(\texttt{Output}, t, i)$ to $\mathcal{F}_{\text{R2PC}}$
on $(t, i, y_{t,i})$ from $\mathcal{F}_{\text{R2PC}}$
$(t, i, y'_{t,i}) \in \mathsf{eval}(\sigma)$
// The xor between the output produced by
   the ideal functionality and the output
   for the sequence produced by the
   simulator
$\Delta \leftarrow y_{t,i} \oplus y'_{t,i}$
$(\Psi, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(\psi, \bar{t}; r)$
$L \leftarrow \mathsf{li}(o_{t,i}, e_{\bar{t},i})$
$E \leftarrow \mathsf{E}_{\mathsf{wk}}(r)$
if $w = 1$ then
    // For a watchlist instance, encode the
       all-zero string
    $X_{\bar{t},0} \leftarrow \mathsf{En}(e_t, t, i + f.n, 0)$
else
    // For an evaluation instance, encode
       the xor value
    $\bar{X}_{t,0} \leftarrow \mathsf{En}(e_t, t, i + f.n, \Delta)$
send $(L, E, \Psi, X_{\bar{t},0}, d_{\bar{t}})$ to Env
```

Fig. 19. Bob corrupt: INITIALIZE, FUNC, GARBLE, LINK, OUTPUT

commitment scheme. In the watchlist positions it still commits correctly to both input tokens.

In the second hybrid, we let the simulator replace encryptions under the watchlist keys with encryption of dummy values under the watchlist key in all evaluation instances. Since the environment does no know the watchlist key

```
rule $\textsc{Input}_{\mathsf{A}}$
on $(\texttt{Input}, t, i, ?)$ from $\mathsf{Env}$
await $(\texttt{Garble}, t) \in \sigma$
$\bar{t} \leftarrow 1 \| (\texttt{Input}, t, i) \| 0$
$\ell_1 \leftarrow \mathrm{len}(f_t.A_i)$
$\ell_2 \leftarrow \mathrm{len}(g_\ell.A_2)$
$\ell_3 \leftarrow \mathrm{len}(g_\ell.A_3)$
$m \xleftarrow{\$} \{0,1\}^{\ell_2}$
$x' \leftarrow 0^{\mathrm{len}(f_t.A_i)}$ // Input the all-zero string to the sequence
$r \in \{0,1\}^k$,
$(G_{\bar{t}}, e_{\bar{t}}, d_{\bar{t}}, o_{\bar{t}}) \leftarrow \mathsf{Gb}(g_{\ell_1}, \bar{t}; r)$
for $u \in \{1, \ldots, \ell_1\}$ do $X_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 0)$, $X_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},1,u}, 1)$, $r_{u,0}, r_{u,1} \xleftarrow{\$} \{0,1\}^k$
for $u \in \{1, \ldots, \ell_2\}$ do $M_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 0)$, $M_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},2,u}, 1)$, $r'_{u,0}, r'_{u,1} \xleftarrow{\$} \{0,1\}^k$
for $u \in \{1, \ldots, \ell_3\}$ do $C_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},3,u}, 0)$, $C_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},3,u}, 1)$, $r''_{u,0}, r''_{u,1} \xleftarrow{\$} \{0,1\}^k$
send $E \leftarrow \mathsf{E}_{\mathsf{wk}}(r)$ to $\mathsf{Env}$
send $(G'_{\bar{t}}t, d'_{\bar{t},2}) \leftarrow (G_{\bar{t}}, d_{\bar{t},2})$ to $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_1\}$ do
    $S_{u,1} \leftarrow \{\mathsf{com}(X_{u,0}, r_{u,0}), \mathsf{com}(X_{u,1}, r_{u,1})\}$
    $E_{u,1} \leftarrow \mathsf{E}_{\mathsf{wk}}((X_{u,0}, X_{u,1}))$, $E_{u,2} \leftarrow \mathsf{E}_{\mathsf{wk}}((r_{u,0}, r_{u,1}))$
    $E_{u,3} \leftarrow \mathsf{E}_{\mathsf{ek}}((X_{u,x_{i,u}}, r_{u,x_{i,u}}))$
    $L_u \leftarrow \mathsf{li}(o_{\bar{t},1,u}, e_{t,i,u})$
    send $S_{u,1}, E_{u,1}, E_{u,2}, E_{u,3}, L_u$ to $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_2\}$ do
    $S_{u,2} \leftarrow \{\mathsf{com}(M_{u,0}, r'_{u,0}), \mathsf{com}(M_{u,1}, r'_{u,1})\}$
    $E_{u,4} \leftarrow \mathsf{E}_{\mathsf{wk}}((M_{u,0}, M_{u,1}))$, $E_{u,5} \leftarrow \mathsf{E}_{\mathsf{wk}}((r'_{u,0}, r'_{i,1}))$
    $E_{u,6} \leftarrow \mathsf{E}_{\mathsf{ek}}((M_{u,m_{i,u}}, r'_{u,m_u}))$
    send $S_{u,2}, E_{u,4}, E_{u,5}, E_{u,6}, L_u$ to $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_3\}$ do
    $(\mathcal{C}_{u,0}, \mathcal{C}_{u,1}) \leftarrow (\mathsf{com}(C_{u,0}; r''_{u,0}), \mathsf{com}(C_{u,1}; r''_{u,1}))$
    send $\mathcal{C}_{u,0}, \mathcal{C}_{u,1}$ to $\mathsf{Env}$
on $c$ from $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_3\}$ do
    send $(C_{u,c_u}, r''_{u,c_u})$ to $\mathsf{Env}$
send $(\texttt{Input}, t, i, ?)$ to $\mathcal{F}_{\text{R2PC}}$
on $(\texttt{Input}, t, i, \texttt{done})$ from $\mathcal{F}_{\text{R2PC}}$
send $(\texttt{Input}, t, i, \texttt{done})$ to $\mathsf{Env}$
$\sigma \leftarrow \sigma \| (\texttt{Input}, t, i, x')$
```

Fig. 20. Bob corrupt: $\textsc{Input}_A$

in the evaluation instances, this change is indistinguishable by the IND-CPA security of the encryption scheme. Similarly, it will encrypt dummy values under the evaluation keys in the watchlist positions.

Notice that now the values sent in the watchlist positions can be computed without knowing the input of Alice, as only the input tokens encrypted under the evaluation key depend on the input of Alice. Furthermore, the garbled values sent in the evaluation instances corresponds to values that the simulator could

```
rule $\textsc{Input}_\mathrm{B}$
on $(\mathtt{Input}, t, i, ?)$ from $\mathsf{Env}$
await $(\mathtt{Garble}, t) \in \sigma$
$\ell \leftarrow \mathrm{len}(f_t.A_i)$
$\ell_1 \leftarrow 2(\ell + s)$
$\bar{t} \leftarrow \mathsf{next}()$
$t' \leftarrow \mathsf{next}()$
$(\mathsf{Id}_{\bar{t}}, e_{\bar{t}}, o_{\bar{t}}, d_{\bar{t}}) \leftarrow \mathsf{Gb}(\mathsf{id}_\ell, \bar{t}; r)$
for $u \in \{1, \ldots, \ell_1\}$ do
    $\bar{X}_{u,0} \leftarrow \mathsf{En}(e_{\bar{t},u}, 0)$
    $\bar{X}_{u,1} \leftarrow \mathsf{En}(e_{\bar{t},u}, 1)$
send $E \leftarrow \mathsf{E}_{\mathsf{wk}}(r), \mathsf{Id}_{\bar{t}}$ to $\mathsf{Env}$
// Learn $\bar{x}$ from Bob's input to OT.
for $u \in \{1, \ldots, \ell_1\}$ do
    on $(\mathtt{Transfer}, \bar{x}_u)$ from $\mathsf{Env}$
    send $(\mathtt{Transfered}, \bar{x}_u, \bar{X}_{u,\bar{x}_u})$ to $\mathsf{Env}$
    store $\bar{x}_u$
$\bar{x} \leftarrow \bar{x}_1, \ldots, \bar{x}_{\ell_1}$
on $h$ from $\mathsf{Env}$
$(H_{t'}, e_{t'}, o_{t'}, d_{t'}) \leftarrow \mathsf{Gb}(h, t'; r')$
send $\mathsf{E}_{\mathsf{wk}}(r')$ to $\mathsf{Env}$
send $H_{t'}$ to $\mathsf{Env}$
for $u \in \{1, \ldots, \ell_1\}$ do
    send $\bar{L}_u \leftarrow \mathsf{li}(o_{\bar{t},u}, e_{t',u})$ to $\mathsf{Env}$
    send $L_u \leftarrow \mathsf{li}(o_{t',u}, e_{t,i,u})$ to $\mathsf{Env}$
// Learn Bob's input from $\bar{x}$ and from the hash function
$x \leftarrow h(\bar{x})$
send $(\mathtt{Input}, t, i, x)$ to $\mathcal{F}_{\mathrm{R2PC}}$
on $(\mathtt{Input}, t, i, \mathtt{done})$ from $\mathcal{F}_{\mathrm{R2PC}}$
send $(\mathtt{Input}, t, i, \mathtt{done})$ to $\mathsf{Env}$
$\sigma \leftarrow \sigma \,\|\, (\mathtt{Input}, t, i, x)$
```

Fig. 21. Bob corrupt: $\textsc{Input}_B$

get when being an adversary in the game in Figure 6, namely garbled inputs, where it gets/needs only one token for each wire, garbled functions and output decoding functions. We can therefore now change the simulation such that the simulator uses the real inputs of Alice (it can learn these by inspecting the ideal functionality (a simulator might not do this, but we may do this as part of the proof as we are just defining a hybrid distribution)) and $\Delta = 0^\ell$. When garbling the input of Alice it asks to get an input encoding of 0^ℓ or the real input of Alice and then commits to the received tokens and all-zero values for the tokens it does not know, and when it is to give output it asks to get a garbling of either $\Delta = y \oplus y'$ or $\Delta = 0^\ell$. If one is concisely using either the left value or the right value, these inputs give the same outputs. Therefore the change to using the real inputs goes unnoticed by the environment by confidentiality of the garbling scheme. This also uses the perfect hiding of the auxiliary function.

After the change to using the real inputs and the all-zero string in output, we can then reverse the two first changes. First we start encrypting correct values under all evaluation and watchlist keys, and then we start committing to all the correct values again. This goes unnoticed by IND-CPA security and computational hiding.

After this series of changes, the hybrid we arrived at is identical to the real protocol, so apply transitivity of indistinguishability.

B Analysis

In this section, we demonstrate the security of our reactive garbling scheme in the random oracle model. The first key idea of this proof is that the values produced by the simulator will only depend on the leakage of one of the sequences. If the sequence don't have the same leakage then the adversary loses anyways. As a result, either the adversary loses the game or the views produced are indistinguishable.

The second key idea is that we can explain the values produced as an instance of either sequence by programming the random oracle. Therefore, if we show that the adversary cannot distinguish between the simulation, and a real garbler, we can show that the adversary has negligible advantage in the confidentiality game.

The basic methodology of the simulator is as follows: when the adversary sends a GARBLE command, the simulator will produce random tokens and random tables that only depend on the leakage of the function. For each non-output wire, we will fix a token that the sender will receive and one token that will remain hidden from the sender. The same thing will be applied to outputs except for the second least significant bit which will be defined after the garbling.

As each output becomes defined, the second least significant of encoded output and the decoding will be chosen by the simulator to reflect this value. If they are not the same then the adversary loses the game. There is a subtle issue that need to be considered, the adversary might first request a decoding string for an output before the output is ready or he may ask for tokens or linking which defines an output before the decoding string is given. In the first case, we will select a random decoding and then fix the second least significant so that decoding produces the correct output and in the second case, we will instead select the second least significant randomly and then select the decoding so the correct output is produced.

To show that the values shown to the adversary could be used to explain the sequence, we simply program the oracle to match one of the two sequences.

Here is a short description of what the simulator does. PROGRAM() is used to ensure that the chosen token would be produced by an adversary who evaluates the sequence correctly. On GARBLE(t) return a garbling with the right structure and number of garbled tables, where each entry in each table is chosen uniformly at random. Additionally for each non-output wire select two uniformly random tokens without assigning semantics. The random oracle will only at evaluation time be programmed to be consistent with the tables and the tokens. For the output wire, do the same but with the second least significant to bit be determined. On INPUT(t, i, x), simply return the token that the simulator intended to reveal. On LINK(t_1, i_1, t_2, i_2), simply return the values that are meant to be produced.

Often we will use σ_0, in the simulation. This is to show that the simulator does not really care which bit was chosen by the game, he only needs the leakage of the function.

We use the following notation to simplify the description of the simulation. We denote the value of the wire i of f_t for the sequence σ_b as $\mathsf{w}(t, i, b)$. We use the notation $G_{t,i,b}$ to denote the gate associated to the function f_t for the sequence σ_b.

B.1 Proof

We now prove Theorem 1

First, we will prove that either the adversary sends a command which loses him the game or that the view of the adversary when interacting with the simulator when $b = 0$ is statistically indistinguishable from the view of the adversary when interacting with the simulator and the game picks $b = 1$. Secondly, we need to show that given the linking, the garbled tables, the encoding and decodings that are produced, we can explain for any choice of $b \in \{0, 1\}$ these values as a garbling of σ_b by programming the random oracle. We also need to prove that the explanation is indistinguishable from a real garbling of the sequence σ_b.

We first prove that the view of the adversary when $b = 0$ and $b = 1$ are statistically indistinguishable unless the adversary loses the game. We prove that by showing that for any legal sequences σ_0 and σ_1 for which $\Phi(\sigma_0) = \Phi(\sigma_1)$ we can produce the view of the adversary of a garbling of σ_b without using the value b. This clearly proves the statement.

The following are constructed without taking into account the bit that was selected. On garble command, the simulator produces two token for each input, the first token is always the token that the simulator will provide to the adversary if asked to encode an input and the other will always remain hidden. One of the tokens will always stay hidden by the input uniqueness condition.

For the non-output wires and non-input wires, the simulator chooses randomly a token that the adversary would compute using inputs, output and linking and a token that would remain hidden. It is the same thing for the output wires except that the second to last bit is left undefined until later. Since linking does not depend on this bit, linking does not depend on b.

Now the second to last bit of an output wire and the decoding are always randomly constructed such that their xor is the output produced by the first sequence. The adversary could select two sequences which produce different output and where a decoding is provided but then the adversary would lose the game. Since both sequences need to produce the same output for any output which is ready and where the decoding is provided and that the function associated to tag t in σ_0 and σ_1 have to be similar. Therefore, the values produced when $b = 0$ is the same when $b = 1$. Therefore, the adversary cannot distinguish between the view produced with $b = 0$ and the view produced with $b = 1$ without losing the game.

Next, we prove that for any $b \in \{0, 1\}$, the values produced can be explained as a garbling of the sequence σ_b.

Notice that only one element of each garbled table has been given a pre-image, namely the one that would be decoded by the adversary. This of course, leaves the simulator the freedom to program the pre-images of the other elements in a garbled table. The other thing to note is that except for the encoded outputs for which both of the following conditions hold: 1) they are ready and 2) the associated decoding has been given, none of the other tokens have been given a semantic meaning yet.

The simulator in the Explain procedure, for each token that the adversary would be able to produce assigns the correct semantic by programming the oracle for that token (the one that the sequence would give). Now there might be inputs that are still undefined and as such the value of gates which depend on that input would still be undefined. In this case, the simulator just choose all four pre-images and can just assign semantics as he chooses. The result is that the garbling with the chosen tokens and the programming is a valid garbling of the sequence σ_b.

We also need to prove that the explanation is indistinguishable from a real garbling of the sequence σ_b. This is trivial, since there is a one-to-one correspondence between what could be produced by a garbler and a random oracle and a simulator programming the random oracle.

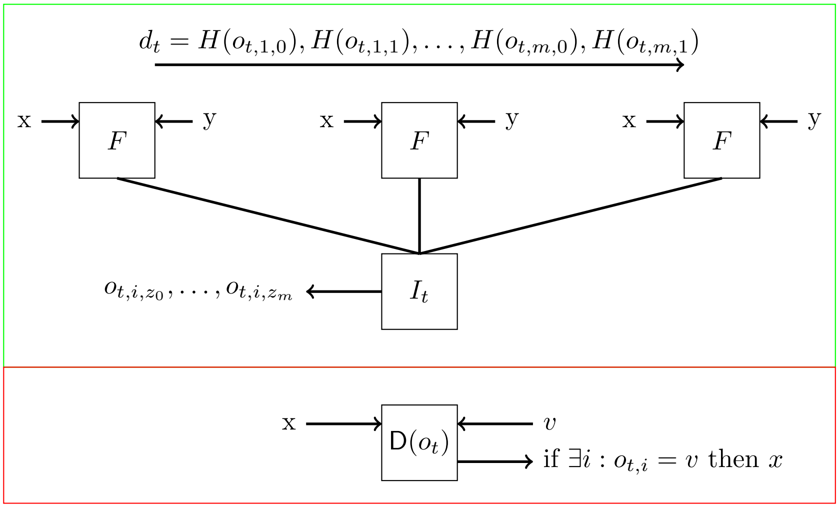
We thus have that the views produced are indistinguishable or the adversary loses. The values produced by the simulator can be explained as a garbling of each sequence and is indistinguishable from a sequence generated by a real garbler. As a result, we can see that the adversary's advantage is negligible and that the protocol is confidential.

C Forge and Lose

In the work of [20], it was shown how to apply cut-and-choose with garbled circuits. We will show that if a reactive garbling scheme fulfills two conditions then we can apply Lindell's technique. The first requirement is that the output encoding is projective. The second condition requires that if an output encoding is not linked then it is safe to reveal the output encoding. We provide a brief overview of the protocol. We will avoid discussing enforcing input consistency, preventing selective failure attacks since these can be derived from the previous protocol. The cut-and-choose occurs after the evaluation.

First, we will note that the scheme of Lindell is a garbling scheme with projective output tokens. On garbling a function, the algorithm select random output tokens for each output. The decoding string is the table containing all the hashes of the output tokens. To decode an encoded output, look for the hash which matches this encoded output.

The protocol overview is as follows. First, Alice garbles the identity function once, she also garbles the function f a total of s times, she also links each garbling of f to the garbling of the identity function. Next, Alice and Bob will first run the evaluation of the garbled function using their respective input x, y. Next Alice and Bob will run the cheater detection phase. If Bob has for any i both $o_{t,i,0}, o_{t,i,1}$ then he can extract Alice's input. Now, he gets an output either from the evaluation or he can compute it using Alice's input. Note that after evaluation, Alice and Bob will execute Cut-and-Choose to verify that Alice acted honestly. This entails that Alice reveal the output encoding of the identity function.

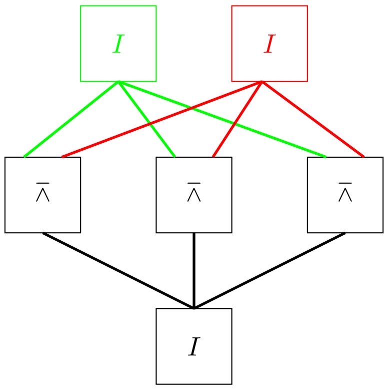

$d_t = H(o_{t,1,0}), H(o_{t,1,1}), \ldots, H(o_{t,m,0}), H(o_{t,m,1})$
x
y
F
I_t
$o_{t,i,z_0}, \ldots, o_{t,i,z_m}$
x
$\mathsf{D}(o_t)$
v
if $\exists i : o_{t,i} = v$ then x

D Minilego

In the case of minilego, we can define minilego soldering of gates as a reactive garbling scheme. Minilego uses the free-xor technique. Each gate is a function. Prior to garbling a fixed parameter Δ is randomly selected.

To garble a gate, each input zero token $e_{t,i,o}$ is randomly selected and $e_{t,i,1} \leftarrow e_{t,i,0} \oplus \Delta$. The output zero token $o_{t,1,o}$ is randomly selected and $o_{t,1,1} \leftarrow o_{t,1,0} \oplus \Delta$. To link a gate associate with label t to a gate with label $\bar{t}$ and input $i \in \{0, 1\}$, we set $L_{t,1,\bar{t},i} \leftarrow o_{t,1,o} \oplus e_{\bar{t},i,0}$. Therefore, we have that $L_{t,1,\bar{t},i} \oplus o_{t,i,b} = e_{\bar{t},i,b}$.

A bucket is simply produced by garbling two input identity gate and an output identity gate and then linking each gate in the bucket with the identity functions.


I
I
$\overline{\wedge}$
$\overline{\wedge}$
$\overline{\wedge}$
I

```
proc $\textsc{Garble}(t)$
$(n, m, q, A, B) \leftarrow \phi(\sigma_0.f_t)$
for $i \in \{1, \ldots, n+q-m\}$ do
    $c_{t,i} \xleftarrow{\$} \{0,1\}$
    // $V_{t,i,0}$ is the token that the adversary will receive for
       internal wires.
    $V_{t,i,0} \leftarrow \{0,1\}^{k-1} \parallel c_{t,i}$
    $V_{t,i,1} \leftarrow \{0,1\}^{k-1} \parallel 1 - c_{t,i}$
for $i \in \{n+q-m+1, \ldots, n+q\}$ do
    /* $K_{t,i,0}$ is the root of the encoded output that the adversary
       will learn. $c_{t,i}$ will be the type of the encoded output that
       the adversary will learn. The mask of the encoded output will
       be chosen later.                                               */
    $K_{t,i,0} \xleftarrow{\$} \{0,1\}^{k-2}$
    $K_{t,i,1} \xleftarrow{\$} \{0,1\}^{k-2}$
    $c_{t,i} \xleftarrow{\$} \{0,1\}$
for $(i, u, v) \in \{1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ do
    // Randomly sample table.
    $P[i, u, v] \xleftarrow{\$} \{0,1\}^k$
$F_t \leftarrow (n, m, q, A, B, P)$
$\sigma_0 \leftarrow \sigma_0 \parallel (\texttt{Garble}, t)$
$\sigma_1 \leftarrow \sigma_1 \parallel (\texttt{Garble}, t)$
return $F_t$

proc $\textsc{Input}(t, i, x_0, x_1)$
$\bar{\sigma}_0 \leftarrow \sigma_0 \parallel (\texttt{Input}, t, i, x_0)$
$S \leftarrow \mathsf{ready}(\bar{\sigma}_0) \setminus \mathsf{ready}(\sigma_0)$
for $(t, i) \in S$ do
    $\mathsf{update}(t, i, \bar{\sigma}_0)$
$\sigma_0 \leftarrow \sigma_0 \parallel (\texttt{Input}, t, i, x_0)$
$\sigma_1 \leftarrow \sigma_1 \parallel (\texttt{Input}, t, i, x_1)$
return $V_{t,i,0}$

proc $\mathsf{update}(t, i, \bar{\sigma}_0)$
/* If output is ready, then the output
   decoding has already been produced. In
   this case, the simulator must choose a
   mask such that the correct output is
   produced. Otherwise, the mask is chosen
   randomly.                                */
if $(\texttt{Output}, t, i) \in \sigma_0$ then
    for $(t, i, y_{t,i}) \in \mathsf{eval}(\bar{\sigma}_0)$ do
        $r_{t,i} \leftarrow y_{t,i} \oplus d_{t,i}$
else
    $r_{t,i} \xleftarrow{\$} \{0,1\}$
$Y_{t,i,y_{t,i}} \leftarrow K_{t,i,0} \parallel r_{t,i} \parallel c_{t,i}$
$Y_{t,i,1-y_{t,i}} \leftarrow K_{t,i,1} \parallel 1 - r_{t,i} \parallel 1 - c_{t,i}$
```

Fig. 22. Reactive garbling Simulation

```
proc $\textsc{Link}(t_1, i_1, t_2, i_2)$
$\bar{\sigma}_0 \leftarrow \sigma_0 \| \textsc{Link}(t_1, i_1, t_2, i_2)$
$S \leftarrow \mathsf{ready}(\bar{\sigma}_0) \setminus \mathsf{ready}(\sigma_0)$
for $(t, i) \in S$ do
    $\mathsf{update}(t, i, \bar{\sigma}_0)$
$\sigma_0 \leftarrow \sigma_0 \,\|\, \textsc{Link}(t_1, i_1, t_2, i_2)$
$\sigma_1 \leftarrow \sigma_1 \,\|\, \textsc{Link}(t_1, i_1, t_2, i_2)$
$U_0 \leftarrow \mathsf{H}(C \,\|\, k_{t,i,r_i}) \oplus V_{t,i,c_{t,i}}$
$U_1 \leftarrow \mathsf{H}(C \,\|\, k_{t,i,1-r_i}) \oplus V_{t,i,1-c_{t,i}}$
$L_{t_1,i_1,t_2,i_2} \leftarrow (U_0, U_1)$
return $L_{t_1,i_1,t_2,i_2}$

proc $\textsc{Output}(t, i)$
$\sigma_0 \leftarrow \sigma_0 \,\|(\texttt{Output}, t, i)$
$\sigma_1 \leftarrow \sigma_1 \,\|(\texttt{Output}, t, i)$
/* If output is not ready then sample a
   random $d_{t,i} \xleftarrow{\$} \{0,1\}$. Otherwise, the mask
   has already been chosen, in this case
   $d_{t,i}$ is chosen so that the correct output
   is produced.                              */
if $(t, i) \notin \mathsf{ready}(\sigma_0)$ then
    $d_{t,i} \xleftarrow{\$} \{0,1\}$
else
    $\exists (t, i, y_{t,i}) \in \mathsf{eval}(\sigma_0)$
    $d_{t,i} \leftarrow r_{t,i} \oplus y_{t,i}$
return $d_{t,i}$

proc $\textsc{Program}()$
/* Program the random oracle so that the
   chosen encodings are produced.            */
for $t \in \mathrm{Tags}(\sigma_0)$ do
    $(n, m, q, A, B, P) \leftarrow \sigma_0.f_t$
    for $i \in \{1, \ldots, n+q-m\}$ do
        if $\mathsf{w}(t, i, \sigma_0) \neq \perp$ then
            $a \leftarrow A(i), b \leftarrow B(i)$
            $A \leftarrow \mathsf{root}(V_{t,a,0})$
            $B \leftarrow \mathsf{root}(V_{t,b,0})$
            $\mathfrak{a} \leftarrow \mathsf{lsb}(V_{t,a,0})$
            $\mathfrak{b} \leftarrow \mathsf{lsb}(V_{t,b,0})$
            $T \leftarrow i \,\|\, \mathfrak{a} \,\|\, \mathfrak{b}$
            $\mathsf{H}(T \,\|\, A \,\|\, B) \leftarrow P[i, \mathfrak{a}, \mathfrak{b}] \oplus V_{t,i,0}$
    for $i \in \{1, \ldots, m\}$ do
        if $(y_{t,i} \in \mathsf{eval}(\sigma_0))$ then
            $a \leftarrow A(n+q-m+i), b \leftarrow$
            $B(n+q-m+1)$ $A \leftarrow \mathsf{root}(V_{t,a,0})$
            $B \leftarrow \mathsf{root}(V_{t,b,0})$
            $\mathfrak{a} \leftarrow \mathsf{lsb}(V_{t,a,0})$
            $\mathfrak{b} \leftarrow \mathsf{lsb}(V_{t,b,0})$
            $T \leftarrow i \,\|\, \mathfrak{a} \,\|\, \mathfrak{b}$
            $\mathsf{H}(T \,\|\, A \,\|\, B) \leftarrow P[i, \mathfrak{a}, \mathfrak{b}] \oplus Y_{t,i,y_{t,i}}$
```

Fig. 23. Program

```
proc $\textsc{Explain}()$
for $\{t \mid (\mathit{Garble}, t) \in \sigma_0\}$ do
    $(n, m, q, A, B, P) \leftarrow \sigma_b.f_t$
    for $i \in \{1, \ldots, n+q-m\}$ do
        if $\mathsf{w}(t, i, \sigma_b) \neq \perp$ then
            $\varDelta \leftarrow \mathsf{w}(t, i, \sigma_b)$
            $X_{t,i,\varDelta} \leftarrow V_{t,i,0}$
            $X_{t,i,1-\varDelta} \leftarrow V_{t,i,1}$
        else
            $X_{t,i,0} \leftarrow V_{t,i,0}$
            $X_{t,i,1} \leftarrow V_{t,i,1}$
    for $i \in \{1, \ldots, m\}$ do
        if $d_{t,i} \neq \perp \wedge r_{t,i} = \perp$ then
            $r_{t,i} \leftarrow d_{t,i}$
            $\textsc{Upd}(t, i)$
        if $r_{t,i} \neq \perp \wedge d_{t,i} = \perp$ then
            $d_{t,i} \leftarrow r_{t,i}$
            $\textsc{Upd}(t, i)$
    for $i \in \{1, \ldots, n+q\}$ do
        $a \leftarrow A(i), b \leftarrow B(i)$
        for $(u, v) \in \{0,1\} \times \{0,1\}$ do
            $A \leftarrow \mathsf{root}(X_{t,a,u}), \mathfrak{a} \leftarrow \mathsf{lsb}(X_{t,a,u})$
            $B \leftarrow \mathsf{root}(X_{t,b,v}), \mathfrak{b} \leftarrow \mathsf{lsb}(X_{t,b,v})$
            $T \leftarrow g \parallel \mathfrak{a} \parallel \mathfrak{b}$
            $\mathsf{H}(T \parallel A \parallel B) \leftarrow P[i, \mathfrak{a}, \mathfrak{b}] \oplus X_{t,i,G_{t,i,b}(i,u,v)}$

proc $\textsc{Upd}(t, i)$
$\varDelta \xleftarrow{\$} \{0,1\}$
$Y_{t,i,0} \leftarrow K_{t,i,\varDelta} \parallel r_{t,i} \parallel c_{t,i}$
$Y_{t,i,1} \leftarrow K_{t,i,1-\varDelta} \parallel 1 - r_{t,i} \parallel 1 - c_{t,i}$
$X_{t,n-q+m+i,1} \leftarrow Y_{t,i,0}$
$X_{t,n-q+m+i,0} \leftarrow Y_{t,i,1}$
```

Fig. 24. Correctness

```
proc $\mathsf{Gb}(\Delta, f_t, t)$
if $f_t = I$ then
    return $\mathsf{GBID}(t)$
$e_{t,1,0}, e_{t,2,0} \xleftarrow{\$} \{0,1\}^k$
$e_{t,1,1} \leftarrow e_{t,1,0} \oplus \Delta$
$e_{t,2,1} \leftarrow e_{t,2,0} \oplus \Delta$
$a_0, a_1 \xleftarrow{\$} \{0,1\}^k$
$d_t \leftarrow H(o_{t,1,0}), H(o_{t,1,1})$
if $f_t = \mathtt{Xor}$ then
    $F_t = \mathtt{Xor}$
    $o_{t,1,0} \leftarrow e_{t,1,0} \oplus e_{t,2,0}$
else
    $o_{t,1,0} \xleftarrow{\$} \{0,1\}^k$
    $a \leftarrow \mathsf{lsb}(e_{t,1,0})$
    $b \leftarrow \mathsf{lsb}(e_{t,2,0})$
    for $v, q \in \{0,1\}^2$ do
        $c \leftarrow a \oplus v$
        $d \leftarrow b \oplus q$
        $z \leftarrow H(t \| e_{t,1,c} \| e_{t,2,d})$
        $P_t[v,q] \leftarrow z \oplus o_{t,i,f_t(c,d)}$
    $F_t \leftarrow P_t$
$o_{t,1,1} \leftarrow o_{t,1,0} \oplus \Delta$
$o_t \leftarrow (o_{t,1,0}, o_{t,1,1})$
$e_t \leftarrow (e_{t,1,0}, e_{t,1,1}, e_{t,2,0}, e_{t,2,1})$
return $(F_t, o_t, e_t, d_t)$

proc $\mathsf{En}(t, i, x)$
$X_{t,i} \leftarrow e_{t,i,x}$
return $X_{t,i}$

proc $\mathsf{li}(t_1, i_1, t_2, i_2, o_{t_1,i_1}, e_{t_2,i_2})$
$L_{t_1,i_1,t_2,i_2} \leftarrow o_{t_1,i_1} \oplus e_{t_2,i_2}$
return $L_{t_1,i_1,t_2,i_2}$

proc $\mathsf{Li}(L_{t_1,i_1,t_2,i_2}, Y_{t_1,i_1})$
return $L_{t_1,i_1,t_2,i_2} \oplus Y_{t_1,i_1}$

proc $\mathsf{EvXor}(X_{t,1}, X_{t,2})$
return $X_{t,1} \oplus X_{t,2}$

proc $\mathsf{EvIdentity}(F_t, X_{t,1})$
$(I, \delta_t) \leftarrow F_t$
return $X_{t,1} \oplus \delta_t$
```

```
proc $\mathsf{GBID}(t)$
$e_{t,1,0} \xleftarrow{\$} \{0,1\}^k$
$e_{t,1,1} = e_{t,1,0} \oplus \Delta$
$\delta_t \xleftarrow{\$} \{0,1\}^k$
$o_{t,1,0} \leftarrow e_{t,1,0} \oplus \delta_t$
$o_{t,1,1} \leftarrow e_{t,1,1} \oplus \delta_t$
return $((I, \delta_t), e_t, o_t)$

proc $\mathsf{EvI}(F_t, \mathbb{X})$
if $F_t = (I, \delta_t)$ then
    $\mathsf{EvIdentity}(F_t, \mathbb{X}_1)$
else if $F_t = \mathtt{Xor}$ then
    $\mathsf{EvXor}(\mathbb{X}_1, \mathbb{X}_2)$
else
    $\mathsf{EvF}(F_t, \mathbb{X}_1, \mathbb{X}_2)$

proc $\mathsf{EvF}(F_t, X_{t,1}, X_{t,2})$
$a \leftarrow \mathsf{lsb}(X_{t,1})$
$b \leftarrow \mathsf{lsb}(X_{t,2})$
return $H(t \| X_{t,1} \| X_{t,2}) \oplus F_t[a,b]$

proc $\mathsf{De}(Y_{t,i}, d_{t,i})$
if $H(Y_{t,i}) = d_{t,i,0}$ then
    return 0
if $H(Y_{t,i}) = d_{t,i,1}$ then
    return 1
return $\mathtt{error}$
```

Fig. 25. Minilego

E Simulation based security

Definition 7 (Confidentiality (simulation)). *For a legal sequence class $\mathbb{L}$ relative to side-information function Φ and a reactive garbling scheme $\mathcal{G}$, we say that $\mathcal{G}$ is* $(\mathbb{L}, \Phi)$-sim-confidential *if for all PPT $\mathcal{A}$ it holds that $\mathbf{Adv}^{\text{adp.sim.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k)$ is negligible, where $\mathbf{Adv}^{\text{adp.sim.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k) = \Pr[\mathbf{Game}^{\text{adp.sim.con}}_{\mathcal{G},\mathbb{L}',\Phi,\mathcal{A}}(1^k) = \top] - \frac{1}{2}$ and $\mathbf{Game}^{\text{adp.sim.con}}_{\mathcal{G},\mathbb{L}',\Phi}$ is given in Figure 26.*

```
proc Initialize()
$b \xleftarrow{\$} \{0,1\}$
$\sigma \leftarrow \emptyset$

proc Output($t, i$)
$\sigma \leftarrow \sigma \,\|\, (\texttt{Output}, t, i)$
if $b$=0 then
    return $(S \circ \Phi)(\sigma)$
return $d_{t,i}$

proc Link($t_1, i_1, t_2, i_2$)
$\sigma \leftarrow \sigma \,\|\, \texttt{Link}, t_1, i_1, t_2, i_2)$
if $b$=0 then
    return $(S \circ \Phi)(\sigma)$
return $\mathsf{li}(t_1, i_1, t_2, i_2, o_{t_1,i_1}, e_{t_2,i_2})$

proc Finalize($b'$)
if $b = b' \wedge \sigma \in \mathbb{L} \wedge \mathsf{eval}(\sigma) \neq \texttt{Error}$ then
    return $\top$
else return $\bot$

proc Func($f, t$)
$f_t \leftarrow f$
$\sigma \leftarrow (\texttt{Func}, f, t)$
$(S \circ \Phi)(\sigma)$

proc Input($t, i, x$)
$\sigma \leftarrow \sigma \,\|\, (\texttt{Input}, t, i, x)$
if $b$=0 then
    return $(S \circ \Phi)(\sigma)$
return $\mathsf{En}(e_t, t, i, x)$

proc Garble($t$)
$\sigma \leftarrow \sigma \,\|\, (\texttt{Garble}, t)$
if $b$=0 then
    return $(S \circ \Phi)(\sigma)$
$(F_t, e_t, o_t, d_t) \leftarrow \mathsf{Gb}(\mathsf{sps}, f_t, t)$
return $F_t$
```

Fig. 26. The game $\mathbf{Game}^{\text{adp.sim.con},\mathbb{L},\Phi}_{\mathcal{G}}(1^k)$ defining *adaptive simulation confidentiality.* In Finalize, we check that $\sigma \in \mathbb{L}$ and the adversary loses if this is not the case. We can therefore by monotonicity assume that the game returns $\bot$ as soon as it happens that $\sigma \notin \mathbb{L}$. The notation $(S \circ \Phi)(\sigma)$ is used to mean that the game sends $\Phi(\sigma)$ to the simulator. We also use the conventions used in $\mathbf{Game}^{\text{adp.sim.con},\mathbb{L},\Phi}_{\mathcal{G}}(1^k)$.

F Simulation Proof

Theorem 3. *Let $\mathbb{L}$ be the set of all legal garbling sequence, let Φ denote the circuit topology of a function. Then RGS is $(\mathbb{L}, \Phi)$-simulation confidential in the random oracle model.*

On garble command, the simulator produces two token for each input, the first token is always the token that the simulator will provide to the adversary

```
proc $\textsc{Link}(t_1, i_1, t_2, i_2)$
$\bar{\sigma} \leftarrow \sigma \| \textsc{Link}(t_1, i_1, t_2, i_2)$
$S \leftarrow \mathsf{ready}(\bar{\sigma}) \setminus \mathsf{ready}(\sigma)$
for $(t, i) \in S$ do
    $\mathsf{update}(t, i, \bar{\sigma})$
$\sigma \leftarrow \sigma \,\|\, \textsc{Link}(t_1, i_1, t_2, i_2)$
$U_0 \leftarrow \mathsf{H}(C \,\|\, k_{t,i,r_i}) \oplus V_{t,i,c_{t,i}}$
$U_1 \leftarrow \mathsf{H}(C \,\|\, k_{t,i,1-r_i}) \oplus V_{t,i,1-c_{t,i}}$
$L_{t_1,i_1,t_2,i_2} \leftarrow (U_0, U_1)$
$\sigma \leftarrow \sigma \,\|(\texttt{Link}, t_1, i_1, t_2, i_2))$
$\textsc{Program}()$
return $L_{t_1,i_1,t_2,i_2}$
```

```
proc $\textsc{Garble}(t)$
$(n, m, q, A, B, ?) \leftarrow \Phi(\sigma.f_t)$
for $i \in \{1, \dots, n + q - m\}$ do
    $c_{t,i} \xleftarrow{\$} \{0, 1\}$
    $V_{t,i,0} \leftarrow \{0, 1\}^{k-1} \,\|\, c_{t,i}$
    $V_{t,i,1} \leftarrow \{0, 1\}^{k-1} \,\|\, 1 - c_{t,i}$
for $i \in \{n + q - m + 1, \dots, n + q\}$ do
    $K_{t,i,0} \xleftarrow{\$} \{0, 1\}^{k-2}$
    $K_{t,i,1} \xleftarrow{\$} \{0, 1\}^{k-2}$
    $c_{t,i} \xleftarrow{\$} \{0, 1\}$
for $(i, v, q) \in \{1, \dots, n + q\} \times \{0, 1\}^2$
do
    $P[i, v, q] \xleftarrow{\$} \{0, 1\}^k$
$F_t \leftarrow (n, m, q, A, B, P)$
$\sigma \leftarrow \sigma \,\|(\texttt{Garble}, t))$
$\textsc{Program}()$
return $F_t$
```

```
proc $\textsc{Program}()$
for $t \in \mathrm{Tags}(\sigma)$ do
    $(n, m, q, A, B, ?) \leftarrow \Phi(\sigma.f_t)$
    for $i \in \{1, \dots, n + q - m\}$ do
        if $\mathsf{ready}(\mathsf{w}(t, i, \sigma))$ then
            $a \leftarrow A(i), b \leftarrow B(i)$
            $A \leftarrow \mathsf{root}(V_{t,a,0})$
            $B \leftarrow \mathsf{root}(V_{t,b,0})$
            $\mathfrak{a} \leftarrow \mathsf{lsb}(V_{t,a,0})$
            $\mathfrak{b} \leftarrow \mathsf{lsb}(V_{t,b,0})$
            $T \leftarrow i \,\|\, \mathfrak{a} \,\|\, \mathfrak{b}$
            $\mathsf{H}(T \,\|\, A \,\|\, B) \leftarrow P[i, \mathfrak{a}, \mathfrak{b}] \oplus V_{t,i,0}$
```

```
proc $\textsc{Input}(t, i, ?)$
$\bar{\sigma} \leftarrow \sigma \|(\texttt{Input}, t, i, ?)$
$S \leftarrow \mathsf{ready}(\bar{\sigma}) \setminus \mathsf{ready}(\sigma)$
for $(t, i) \in S$ do
    $\mathsf{update}(t, i, \bar{\sigma})$
return $V_{t,i,0}$
```

```
proc $\textsc{Output}(t, i)$
if $(t, i) \notin \mathsf{ready}(\sigma)$ then
    $d_{t,i} \xleftarrow{\$} \{0, 1\}$
else
    for $(t, i, y_{t,i}) \in \mathsf{eval}(\sigma)$ do
        $d_{t,i} \leftarrow r_{t,i} \oplus y_{t,i}$
$\sigma \leftarrow \sigma \,\|(\texttt{Output}, t, i)$
$\textsc{Program}()$
return $d_{t,i}$
```

```
proc $\mathsf{update}(t, i, \bar{\sigma})$
if $(\texttt{Output}, t, i) \in \sigma$ then
    for $(\texttt{Output}, t, i, y_{t,i}) \in \Phi(\bar{\sigma})$ do
        $r_{t,i} \leftarrow y_{t,i} \oplus d_{t,i}$
else
    $r_{t,i} \xleftarrow{\$} \{0, 1\}$
$Y_{t,i,y_{t,i}} \leftarrow K_{t,i,0} \,\|\, r_{t,i} \,\|\, c_{t,i}$
$Y_{t,i,1-y_{t,i}} \leftarrow K_{t,i,1} \,\|\, 1 - r_{t,i} \,\|\, 1 - c_{t,i}$
```

Fig. 27. Reactive garbling simulation

if asked to encode an input and the other will always remain hidden. One of the tokens will always stay hidden by the input uniqueness condition.

For the non-output wires and non-input wires, the simulator chooses randomly a token that the adversary would compute using inputs, output and linking and a token that would remain hidden. It is the same thing for the output wires except that the second to last bit is left undefined until later. Since linking does not depend on this bit, linking does not depend on the particular sequence.

Now the second to last bit of an output wire and the decoding are always randomly constructed such that their xor is the output produced by the leakage of the sequence. Since any sequences with the same side-information produces the same output for any output which is ready and where the decoding is provided and that the function associated to tag t has to to have the same side-information. Therefore, the values produced do not depend on the sequence.

Next, we prove that for any sequence with the given side-information, the values produced are the same.

Notice that only one element of each garbled table has been given a pre-image, namely the one that that would be decoded by the adversary. This of course, leaves the other pre-images of the other elements undefined for the garbled table.

The other thing to note is that except for the encoded outputs for which both of the following conditions hold: 1) they are ready and 2) the associated decoding has been given, none of the other tokens have been given a semantic meaning yet.

The simulator in the Explain procedure, for each token that the adversary would be able to produce assigns the correct semantic by programming the oracle for that token (the one that the sequence would give). Now there might be inputs that are still undefined and as such the value of gates which depend on that input would still be undefined. In this case, the simulator just choose all four pre-images and can just assign semantics later. The result is that the garbling with the chosen tokens and the programming looks like a valid garbling of any sequence with the given side-information.

We also need to prove that the explanation is indistinguishable from a real garbling of any sequence with the given side-information. This holds since the values produced are either random or only depend on the side-information of the sequence.

We thus have that the views produced are indistinguishable. As a result, we can see that the adversary's advantage is negligible and that the protocol is simulation-confidential.

Theorem 4. *Let $\mathbb{L}$ be the set of all legal garbling sequence, let Φ denote a side-information function. If a RGS is $(\mathbb{L}, \Phi)$-simulation confidential then it is $(\mathbb{L}, \Phi)$-indistinguishable confidential.*

Our proof will consist of an initial view and three hybrids. The initial view consists of the adversary interacting with the indistinguishability game that initially sampled $b = 0$. The first hybrid consists of a game that on receipt of sequences σ_0, σ_1, if $\Phi(\sigma_0) \neq \Phi(\sigma_1)$ outputs $\perp$, otherwise it simply sends $\Phi(\sigma_0)$ to the simulator and forwards the response to the adversary. The third game is the same as the second game except that it sends $\Phi(\sigma_1)$ to the simulator. The final hybrid, consists of the adversary interacting with the indistinguishability game that initially sampled $b = 1$.

Since by assumption, the garbling scheme is $(\mathbb{L}, \Phi)$-simulation confidential, it must be that the initial view and the first hybrid are indistinguishable. This also holds for the second and third hybrid. Since by assumption, $\varPhi(\sigma_0) = \varPhi(\sigma_1)$ and that the game only forwards responses from the simulator, the first and second hybrids are indistinguishable. Therefore by transitivity of indistinguishability, we have that the initial view and the final hybrid are indistinguishable and thus the scheme is $(\mathbb{L}, \varPhi)$-indistinguishable confidential.