A State-Separating Proof for Yao's Garbling Scheme

Chris Brzuska
Aalto University, Finland
chris.brzuska@aalto.fi

Sabine Oechsner
University of Edinburgh, UK
s.oechsner@ed.ac.uk

***Abstract*—Secure multiparty computation enables mutually distrusting parties to compute a public function of their secret inputs. One of the main approaches for designing MPC protocols are garbled circuits whose core component is usually referred to as a *garbling scheme*. In this work, we revisit the security of Yao's garbling scheme and provide a modular security proof which composes the security of multiple *layer garblings* to prove security of the full *circuit garbling*. We perform our security proof in the style of state-separating proofs (*ASIACRYPT 2018*).**

I. Introduction

Secure multiparty computation (MPC) allows mutually distrusting parties $P_1, \dots, P_n$ to evaluate a public function f on their secret inputs $x_1, \dots, x_n$ and learn nothing but the result $y = f(x_1, .., x_n)$. In his seminal work [Yao86], Yao proposed a solution for the two-party case: Assume that f is represented as a Boolean circuit C. P_1 encodes or *garbles* the circuit C into $\tilde{C}$ and their own input x_1 into $\tilde{x_1}$, and sends both, $\tilde{C}$ and $\tilde{x_1}$, to P_2. The two parties then engage in a protocol to garble P_2's input x_2 into $\tilde{x_2}$. P_2 evaluates the garbled circuit $\tilde{C}$ on the garbled inputs $\tilde{x_1}$ and $\tilde{x_2}$, decodes the result, and sends it to P_1. The garbling of a circuit uses an encryption scheme, and the protocol is secure if the encryption scheme is indistinguishable under chosen plaintext attacks (IND-CPA) and the parties are semi-honest, i.e. follow the protocol description.

Despite Yao's garbled circuits becoming one of the main MPC design paradigms today, it took 20 years after its inception before the first security proof for Yao's original construction was published by Lindell and Pinkas [LP09]. Bellare, Hoang and Rogaway (BHR) [BHR12b] later abstracted Yao's garbled circuit construction to a general notion of garbling schemes. A garbling scheme allows one party to garble a circuit C and secret input x such that another party can evaluate the garbled circuit and learn the result $y = C(x)$ but nothing (else) about the input. BHR moreover proved security of the garbling scheme derived from Yao's garbled circuits construction, henceforth called *Yao's garbling scheme*, in the style of code-based game-playing [BR06].

Recently, Brzuska, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss (BDFKK) [BDF$^+$18] proposed *state-separating proofs* (SSP), a generalization of code-based game-playing that allows for more modularity in security proofs. SSPs propose to structure the pseudocode of cryptographic games into stateful code pieces (*packages*) that query each other via oracles.

A. Our contribution

In this work, we propose an easily verifiable version of the security proof of Yao's garbling scheme, including the soundness of the reductions. Our work can be seen as the next step towards understanding the security of Yao's garbling scheme which, inspired and empowered by SSPs, revisits existing proofs and refines their structure where appropriate. Security is proven with respect to BHR's garbling scheme syntax and security notion, expressed using SSPs. On a technical level, our proof is guided by the following observations:

1) Modular security proof: In a nutshell, existing security proofs of Yao's garbling scheme consist of two steps: Garbling scheme security is reduced to a proof-specific encryption scheme security notion, which is in a next step reduced to a standard assumption such as IND-CPA security.

In this work, we further break down the first step by separating the reduction to encryption scheme security from arguments about the structure of the circuit. For this purpose, we identify a new intermediate security notion that sits right in the middle between circuit garbling and encryption scheme security: the security of garbling a gate. (For our own convenience, we further assume that the circuit is layered and reason at the level of layers, i.e. sets of gates, instead of individual gates.)

2) Composable security notions: Following BHR, a garbling scheme is called *selectively secure* if the garbling of C and x can be simulated given only C and the circuit evaluation $y := C(x)$, but without knowledge of the secret input x. Just like a circuit can be described as composition of multiple circuit layers, we ask now if circuit *security* can be described as the composition of layer *security*. In the case of selective security, this is unclear. In fact just syntactically, not even the garbling scheme simulators can be composed: Consider a circuit $C := C_2 \circ C_1$ for two subcircuits C_1 and C_2 and input x to C, and assume that C_1 and C_2 can be garbled securely with simulators S_1 and S_2. If we want to construct a simulator S for C that is given only C and $y = C(x)$ from S_1 and S_2, we run into the problem that S needs to provide inputs for S_1. However, simulator S_1 for C_1 expects $y_1 = C_1(x)$ as input which neither S nor S_2 can provide.[1]

[1] One can, rather inelegantly, bypass this problem with a dummy value for y_1 and argue that the *joint* composition of the simulators does not actually depend on the value of y_1.

In the case of Yao's garbling scheme, however, we can refine the security notion and show that security under the modified notion implies selective security. The simulator in this new notion is only given C and a *garbling* of y rather than y itself. Going back to $C := C_2 \circ C_1$, simulator S_2 now simulates the garbling of input y_1 to circuit C_2, and conveniently, can provide S_1 with the required garbling of y_1. Not only can simulators now be composed with each other, but the security notion can be *self-composed*, meaning composing the security notions for garbling multiple individual circuit layers C_i yields a security notion for garbling circuit C.

3) Graph-based reductions: Finally, we write Yao's garbling scheme, our redefined syntax and security notion and *layer versions* thereof in the modular SSP style, that splits pseudo-code into multiple code packages which call one another. As a result, we define our reductions directly as a subset or rather *subgraph* of previously defined packages and use mere associativity of algorithm composition to prove the soundness of the reduction. Thus, our proof foregoes the need to explicitly define reductions and prove their soundness.

B. Outline

Section II introduces garbling schemes. State-separating proofs (SSPs) are introduced in Section III on the example of encryption scheme security, and Section IV formulates garbling schemes in terms of state-separated packages. In Section V, Yao's garbling scheme is introduced, and we state security and outline the security proof which is then presented in Sections VI, VII and VIII. Finally, Section IX discusses conceptual insights and compares with related work.

II. Garbling Schemes

A. Garbling schemes

Bellare, Hoang and Rogaway (BHR) [BHR12b] introduce the notion of a *garbling scheme* as an abstraction of the primitive underlying the garbled circuits approach.

Definition 1 (Garbling scheme [BHR12b])**.** *A* (circuit) garbling scheme *consists of* 5 *probabilistic, polynomial-time algorithms* $\mathsf{gs} = (gb, en, de, ev, gev)$ *for circuit garbling, input encoding and output decoding, circuit evaluation and garbled circuit evaluation, respectively.*

The circuit garbling algorithm gb outputs a garbled circuit $\tilde{C}$ as well as input encoding information e and output decoding information *dinf*. A garbling scheme is *input projective* if the circuit garbling gb generates input encoding information e that consists of two tokens per input bit, and input encoding en selects for each input bit the corresponding token. We assume the circuit evaluation algorithm ev to be fixed and write $C(x)$ instead of $ev(C, x)$, and sometimes omit ev from the description of a garbling scheme. $\Phi(C)$ is defined as the information the circuit garbling leaks about a circuit C (e.g. the circuit topology for Yao's garbling scheme). For simplicity, throughout this article, $\Phi(C)$ will be equal to C.

```
$\texttt{PRVSIM}^0_{\mathsf{gs},\Phi,\mathcal{S}}$

$\mathsf{GARBLE}(C, x)$
$(\tilde{C}, e, \mathrm{dinf}) \leftarrow gb(1^\lambda, C)$
$\tilde{x} \leftarrow en(e, x)$
$\mathbf{return}\ (\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

```
$\texttt{PRVSIM}^1_{\mathsf{gs},\Phi,\mathcal{S}}$

$\mathsf{GARBLE}(C, x)$
$y \leftarrow C(x)$
$(\tilde{C}, \tilde{x}, \mathrm{dinf}) \leftarrow \mathcal{S}(1^\lambda, y, \Phi(C))$
$\mathbf{return}\ (\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

Figure 1: Garbling scheme security games $\texttt{PRVSIM}^b_{\mathsf{gs},\Phi,\mathcal{S}}$.

Definition 2 (Garbling scheme correctness [BHR12b])**.** *Let* $\lambda \in \mathbb{N}$. *A garbling scheme* $\mathsf{gs} = (gb, en, de, gev)$ *is* perfectly correct *if for all circuits C and inputs x,*

$$\Pr_{(\tilde{C},e,dinf)\leftarrow\$ gb(1^\lambda,C)}\Big[C(x) = de(dinf, gev(\tilde{C}, en(e, x)))\Big] = 1$$

Garbling scheme gs *is* statistically correct *if the above equality holds with overwhelming probability in* λ.

BHR provide two equivalent security definitions for garbling schemes, an *indistinguishability-based* and a *simulation-based* definition. The latter follows the simulation paradigm: A real execution of the garbling scheme on real circuit C and input x is compared to a simulated (idealized) execution generated by an algorithm—the *simulator*. The simulator only has access to C and the result $y = C(x)$, but not to the input x itself, and thus, the ideal execution cannot leak more information about x than the output value y. If both executions are indistinguishable, then also the real execution does not leak more information about x than y. Formally, we capture the two executions via games $\texttt{PRVSIM}^b_{\mathsf{gs},\Phi,\mathcal{S}}$ for $b \in \{0, 1\}$ (Figure 1) and define security as indistinguishability between them.

Definition 3 (Garbling scheme security [BHR12b])**.** *Let* $\lambda \in \mathbb{N}$. *A garbling scheme* $\mathsf{gs} = (gb, en, de, gev)$ *is* secure wrt. leakage function Φ *if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that the distinguishing advantage*

$$\left|\Pr\left[1 \leftarrow\$\ \mathcal{A} \to \texttt{PRVSIM}^0_{\mathsf{gs},\Phi,\mathcal{S}}\right] - \Pr\left[1 \leftarrow\$\ \mathcal{A} \to \texttt{PRVSIM}^1_{\mathsf{gs},\Phi,\mathcal{S}}\right]\right|$$

of $\mathcal{A}$ interacting with $\texttt{PRVSIM}^0_{\mathsf{gs},\Phi,\mathcal{S}}$ *and* $\texttt{PRVSIM}^1_{\mathsf{gs},\Phi,\mathcal{S}}$ *is negligible in* λ.

The above security notion is referred to as *selective security* because the adversary needs to choose both circuit C and input x simultaneously. In turn, the stronger notion of *adaptive security* [BHR12a] allows the adversary to obtain a circuit garbling and only then adaptively choose the input x. Adaptive security is notoriously hard to achieve, see, e.g. [JW16] and references therein. This work focuses on the simulation-based notion of selective security presented in Def. 3.

B. Conventions

We encode the security parameter λ in unary 1^λ and omit it whenever it is clear from context. We make the simplifying assumption that each circuit is *layered*. A layered circuit is a circuit whose gates can be partitioned into layers $1, \ldots, d$ such that each wire connects gates in adjacent layers. Circuits such as AES are naturally layered, and transforming an arbitrary circuit of size s into a layered one results in at most a quadratic increase in size [Weg87]. All our results can be modified to non-layered circuits by formulating appropriate gate assumptions instead of layer assumptions. We focus in layered circuits due to their convenient visual representation. Namely, if dependencies between gates can be arbitrary, they can neither be drawn nor described in concise algebraic terms. Finding a suitable notation is an interesting open problem.

III. State-Separating Proofs and Encryption Scheme Security

Security games such as $\texttt{PRVSIM}^0_{\mathsf{gs},\Phi,\mathcal{S}}$ described in the previous section are not known to come with a natural way of composition such as Universal Composability [Can01], [Mau12]. However, Brzuska, Delignat-Lavaud, Fournet, Kohbrok, and Kohlweiss (BDFKK [BDF+18]) observe that by splitting a game into multiple parts while carefully preserving dependencies, one can indeed achieve compositionality and modularity. This section provides a brief overview over the key concepts of their proposal, *state-separating proofs* (SSPs), on the example of encryption scheme security.

A. Games

To understand SSPs, we first need to consider the notion of a game. Following Bellare and Rogaway [BR06], a game is a set of oracles operating on shared state.

Definition 4 (Game)**.** *A game* $\texttt{G}$ *consists of a set of oracles which operate on a shared state.*

To make this idea more concrete, let $se = (enc, dec)$ be a symmetric encryption scheme (with uniform key sampling as key generation), and consider the standard notion for encryption scheme security, indistinguishability under chosen plaintexts (IND-CPA). In the IND-CPA game, an adversary is given access to two oracles: A key sampling oracle SMP that initializes the key, and an encryption oracle ENC that takes two messages as input and returns an encryption of one of them, cf. Fig. 2 for the definition of game $\texttt{IND-CPA}(se)^b$ with $b \in \{0,1\}$. Both oracles are presented in pseudo-code notation. The notation $x \leftarrow y$ means that the value of variable y is stored in variable x, and $x \leftarrow\!\!\$ \ S$ means that x is sampled uniformly at random from S. Finally $x \leftarrow\!\!\$ \ \mathsf{algo}(a)$ means that the randomized algorithm algo is executed on argument a and the result is stored in variable x. Depending on b, the encryption oracle either returns an encryption of the left ($b = 0$) or the right ($b = 1$) message. The two oracles SMP and ENC of $\texttt{IND-CPA}(se)^b$ share state k, the encryption key. se is IND-CPA secure if any efficient adversary who interact with the oracles of the $\texttt{IND-CPA}(se)^b$ cannot determine b much better than with guessing probability.

```
IND-CPA$(se)^b$

SMP()
assert $k = \bot$
$k \leftarrow\!\!\$ \ \{0,1\}^\lambda$
return $k$

ENC$(m_0, m_1)$
assert $k \neq \bot$
assert $|m_0| = |m_1|$
$c \leftarrow\!\!\$ \ enc(k, m_b)$
return $c$
```

Figure 2: Games $\texttt{IND-CPA}(se)^b$.

B. Packages

The base object of SSPs are packages, a generalization of games which not only *provide* oracles, but can also *call* the oracles of other packages. The oracles are described in pseudocode. Importantly, a package's state cannot be accessed directly from the outside, but only through oracle calls.

Definition 5 (Package)**.** *A package* $\texttt{M}$ *provides a* <u>set of oracles</u> $[\rightarrow \texttt{M}]$ *which operate on a shared state and make calls to a set of oracles* $[\texttt{M} \rightarrow]$*, which we call the* <u>dependencies</u> *of* $\texttt{M}$*.*

The term *game* refers then to the special case of a package $\texttt{G}$ which has no dependencies, that is $[\texttt{G} \rightarrow] = \emptyset$. For example, the games $\texttt{IND-CPA}(se)^b$ have $[\texttt{IND-CPA}(se)^b \rightarrow] = \{\mathsf{SMP}, \mathsf{ENC}\}$ and $[\rightarrow \texttt{IND-CPA}(se)^b] = \emptyset$. The converse of games are *adversary* packages which do not provide any oracles and are thought of as the *main procedure*. An adversary outputs a single bit upon termination.

Definition 6 (Adversary)**.** *A package* $\texttt{A}$ *is an* adversary *if* $[\rightarrow \texttt{A}] = \emptyset$*.*

C. Composition

We can compose two packages $\texttt{M}$ and $\texttt{N}$ sequentially along matching oracle names and dependencies into a new package $\texttt{M} \rightarrow \texttt{N}$. Package composition is associative since the states of the individual packages are separated from one other. We represent the composition of packages by *call graphs*. Boxes represent packages and arrows labeled by oracle names represent oracles. We define security using the distinguishing *advantage* of an adversary composed with two games.

Definition 7 (Advantage)**.** *Let* $\texttt{G}^0$ *and* $\texttt{G}^1$ *be two games and let* $\mathcal{A}$ *be an adversary such that* $[\mathcal{A} \rightarrow] = [\rightarrow G_0] = [\rightarrow G_1]$*. Then the distinguishing* advantage *of* $\mathcal{A}$*, denoted* $\mathsf{Adv}(\mathcal{A}; \texttt{G}^0, \texttt{G}^1)$*, is defined as*

$$\Pr\big[1 \leftarrow\!\!\$ \ \mathcal{A} \rightarrow \texttt{G}^0\big] - \Pr\big[1 \leftarrow\!\!\$ \ \mathcal{A} \rightarrow \texttt{G}^1\big].$$

In order to define security, we additionally need to define polynomial runtime of a package.

Definition 8 (PPT runtime)**.** *Let* $\texttt{M}$ *be a package which has security parameter* λ*. If* $\texttt{M}$ *is a game, then we say that* $\texttt{M}$ *is probabilistic polynomial-time (PPT), if each of its oracles runs in time polynomial in* λ*, the number of queries which* $\texttt{M}$ *received and the length of the concatenation of the inputs of these queries, encoded in binary. If* $\texttt{M}$ *has dependencies, then we say that* $\texttt{M}$ *is PPT if for all PPT games* $\texttt{N}$*,* $\texttt{M} \rightarrow \texttt{N}$ *is a PPT game.*

By convention, all packages in a package composition depend on the *same* security parameter which we do not write explicitly. With definitions of advantage and runtime at hand, we can define IND-CPA security as follows.

Definition 9 (IND-CPA security)**.** *A symmetric encryption scheme* $se = (enc, dec)$ *is IND-CPA secure if for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}(\mathcal{A}; \texttt{IND-CPA}^0(se), \texttt{IND-CPA}^1(se))$ *is negligible.*

Note that by definition of PPT runtime, $\mathcal{A}$ can only make a polynomial number of queries to $\texttt{IND-CPA}^b(se)$, since else, the composition $\mathcal{A} \to \texttt{IND-CPA}^b(se)$ would not be PPT.

Two games $\texttt{G}^0$ and $\texttt{G}^1$ are *code equivalent*, denoted $\texttt{G}^0 \overset{\text{code}}{\equiv} \texttt{G}^1$, if for all adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}(\mathcal{A}; \texttt{G}^0, \texttt{G}^1)$ is 0. Arguing about code equivalence is useful when a code transformation in a proof does not affect input-output behaviour.

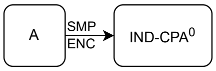


Figure 3: Real IND-CPA game execution.

Going back to our example, we can compose an IND-CPA adversary $\texttt{A}$, $[\texttt{A} \to] = \{\mathsf{SMP}, \mathsf{ENC}\}$, with the real IND-CPA game $\texttt{IND-CPA}(se)^0$. The result $\texttt{A} \to \texttt{IND-CPA}(se)^0$ (cf. Fig. 3) describes the real execution of the IND-CPA game. The adversary $\texttt{A}$ can interact with the game through calls to its oracles and eventually terminates by outputting a bit. We can also express the IND-CPA games as a composition of two packages, a $\texttt{KEY}$ package for key generation/storage and an $\texttt{ENCRYPT}^b$ package for encryption. We denote the resulting new modular games as $\texttt{mIND-CPA}(se)^b$ (cf. Fig. 4). The $\texttt{KEY}$ package (cf. Fig. 5) provides oracles SMP for key sampling and GET to retrieve a stored key. The oracle ENC of packages $\texttt{ENCRYPT}^b$ takes two messages m_0 and m_1, queries the key stored in $\texttt{KEY}$ and outputs an encryption of m_b. The $\texttt{KEY}$ and $\texttt{ENCRYPT}^b$ packages only share state via oracle calls: $\texttt{ENCRYPT}^b$ is stateless while $\texttt{KEY}$ has key k as state. We use assertions for error handling

$$\textbf{assert } \text{cond} := \textbf{if } \neg\text{cond } \textbf{then} \quad \textbf{return } \text{error symbol}$$

and assume that a system cannot be called anymore after an **assert** was violated. However, the adversary will still be allowed to produce an output.

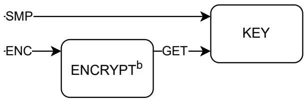


Figure 4: Modular IND-CPA games $\texttt{mIND-CPA}(se)^b$.

```
KEY
SMP()
  assert $k = \bot$
  $k \leftarrow\$ kgen$
  return ()

GET()
  assert $k \neq \bot$
  return $k$

ENCRYPT$^b$
ENC($m_0, m_1$)
  $k \leftarrow$ GET()
  assert $|m_0| = |m_1|$
  $c \leftarrow\$ enc(k, m_b)$
  return $c$
```

Figure 5: Oracles of $\texttt{ENCRYPT}^b$, $b \in \{0,1\}$, and $\texttt{KEY}$.

A package $\texttt{M}$ is not allowed to call its own oracles. Thus package call graphs are directed acyclic graphs. This restriction is in fact a functional style of oracles, i.e. after a caller $\texttt{M}$ calls a callee $\texttt{N}$, the package $\texttt{N}$ might make further oracle calls to other packages, but eventually returns control to $\texttt{M}$.

Notation. $\texttt{G(M)}$ denotes a composed package $\texttt{G}$ which is parametrized by package $\texttt{M}$, i.e. all of $\texttt{G}$ is fixed except for $\texttt{M}$. We write $\texttt{G}(\mathsf{algo})$ for a package (composition) which depends on an algorithm algo.

Examples for code equivalence. By inlining the GET oracle of $\texttt{KEY}$ into the ENC oracle of $\texttt{ENCRYPT}^b$ and comparing the resulting code, we can prove $\texttt{IND-CPA}(se)^0 \overset{\text{code}}{\equiv} \texttt{mIND-CPA}(se)^0$ and $\texttt{IND-CPA}(se)^1 \overset{\text{code}}{\equiv} \texttt{mIND-CPA}(se)^1$.

D. Reductions

We often bound the adversarial advantage between two games $\texttt{G}^0_{big}$, $\texttt{G}^1_{big}$ by the advantage of a related adversary between two smaller games $\texttt{G}^0_{sml}$, $\texttt{G}^1_{sml}$ which capture security of a primitive or a computational hardness assumption.

Lemma 1 (Perfect reduction lemma)**.** *Let $\texttt{G}^0_{big}$, $\texttt{G}^1_{big}$ and $\texttt{G}^0_{sml}$, $\texttt{G}^1_{sml}$ be two game pairs with $[\to \texttt{G}^0_{big}] = [\to \texttt{G}^1_{big}]$ and $[\to \texttt{G}^0_{sml}] = [\to \texttt{G}^1_{sml}]$. If we can define a reduction $\mathcal{R}$ with $[\to \mathcal{R}] = [\to \texttt{G}^0_{big}]$ and $[\mathcal{R} \to] = [\to \texttt{G}^0_{sml}]$ such that*

$$\texttt{G}^0_{big} \overset{\text{code}}{\equiv} \mathcal{R} \to \texttt{G}^0_{sml} \text{ and } \texttt{G}^1_{big} \overset{\text{code}}{\equiv} \mathcal{R} \to \texttt{G}^1_{sml}, \tag{1}$$

then for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}; \texttt{G}^0_{big}, \texttt{G}^1_{big}) = \mathsf{Adv}(\mathcal{B}; \texttt{G}^0_{sml}, \texttt{G}^1_{sml}) \tag{2}$$

where $\mathcal{B} := \mathcal{A} \to \mathcal{R}$. We call $\mathcal{R}$ a perfect reduction.

Proof. Using associativity of package composition, we obtain:

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}; \texttt{G}^0_{big}, \texttt{G}^1_{big}) \\
&= \Pr\big[1 \leftarrow\$ \mathcal{A} \to \texttt{G}^0_{big}\big] - \Pr\big[1 \leftarrow\$ \mathcal{A} \to \texttt{G}^1_{big}\big] \\
&= \Pr\big[1 \leftarrow\$ \mathcal{A} \to (\mathcal{R} \to \texttt{G}^0_{sml})\big] - \Pr\big[1 \leftarrow\$ \mathcal{A} \to (\mathcal{R} \to \texttt{G}^1_{sml})\big] \\
&= \Pr\big[1 \leftarrow\$ (\mathcal{A} \to \mathcal{R}) \to \texttt{G}^0_{sml}\big] - \Pr\big[1 \leftarrow\$ (\mathcal{A} \to \mathcal{R}) \to \texttt{G}^1_{sml}\big] \\
&= \mathsf{Adv}(\mathcal{A} \to \mathcal{R}; \texttt{G}^0_{sml}, \texttt{G}^1_{sml}) = \mathsf{Adv}(\mathcal{B}; \texttt{G}^0_{sml}, \texttt{G}^1_{sml})
\end{aligned}$$

□

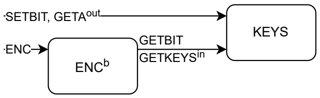


Figure 6: Games $\texttt{2CPA}(se)^b$.

The SSP style is particularly useful for finding and expressing perfect reductions. As an example, consider the modified encryption scheme security notion $\texttt{2CPA}(se)^b$ with two keys in Figure 6. The game consists of packages $\texttt{KEYS}$ and $\texttt{ENC}^b$ shown in Fig. 7, where sets and maps are denoted by capital letters: S, $T(x)$. In this IND-CPA variant with two encryption keys, an adversary chooses one of the keys to be corrupt, while security of encryptions under the honest key is still guaranteed. Package $\texttt{KEYS}$ stores two keys $Z(0)$ and $Z(1)$. $\texttt{KEYS}$ provides oracles SETBIT for choosing which key to corrupt and GETBIT for retrieving this information, $\mathsf{GETA}^{\text{out}}$ for retrieving the adversary key and sampling both keys if they do not exist yet and $\mathsf{GETKEYS}^{\text{in}}$ that returns both keys if they exist. Package $\texttt{ENC}^b$ provides an encryption oracle that encrypts one of two messages. Importantly, oracle ENC retrieves both keys and computes which to use as encryption key. If the key is corrupted, then message m_0 will always be encrypted, else m_b is encrypted depending on the package parameter b. $\texttt{ENC}^b$ retrieves the keys from $\texttt{KEYS}$ via $\mathsf{GETKEYS}^{\text{in}}$ and the bit of the corrupt key via GETBIT. Looking ahead, $\texttt{KEYS}$ will be used in Section IV for the two keys associated with a circuit wire in projective garbling schemes.

Using Lemma 1, we can now reduce an adversary $\mathcal{A}$'s distinguishing advantage for games $\texttt{2CPA}(se)^b$ to the IND-CPA security of the encryption scheme se.

```
Oracle of ENC^b

ENC(d, m_0, m_1)
  $Z^{in} \leftarrow \mathsf{GETKEYS}^{in}()$
  $z^{in} \leftarrow \mathsf{GETBIT}()$
  assert $|m_0| = |m_1|$
  if $z^{in} \neq d$ then
    $c \leftarrow\$ \; enc(Z^{in}(d), m_b)$
  if $z^{in} = d$ then
    $c \leftarrow\$ \; enc(Z^{in}(d), m_0)$
  return $c$
```

```
Oracles of KEYS

SETBIT(z)
  assert $z = \bot$
  $z \leftarrow z$
  return ()

GETKEYS^in()
  assert $z \neq \bot$
  assert aflag
      ∨ bflag
  return $Z$

GETBIT()
  assert $z \neq \bot$
  return $z$

GETA^out()
  aflag ← 1
  if $Z = \bot$ then
    $Z(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z(1) \leftarrow\$ \{0,1\}^\lambda$
  return $Z(z)$
```

Figure 7: Oracles of double key packages KEYS and encryption packages $\texttt{ENC}^0$, $\texttt{ENC}^1$. Further oracles of KEYS will be introduced in Fig. 18. The flag bflag can be ignored for the current section.

Lemma 2. *Let se be a symmetric encryption scheme. For reduction $\mathcal{R}_{cpa} :=$ RED (cf. Fig. 8), it holds that for any PPT adversary $\mathcal{A}$,*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \texttt{2CPA}^0(se), \texttt{2CPA}^1(se))\\ &\leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{cpa}; \texttt{IND-CPA}^0(se), \texttt{IND-CPA}^1(se)).\end{aligned}$$

```
RED

SETBIT(z)
  assert $z = \bot$
  $z \leftarrow z$
  return ()

GETA^out()
  assert $z \neq \bot$
  aflag ← 1
  if $k_a = \bot$ then
    $k_a \leftarrow\$ \{0,1\}^\lambda$
    SMP()
  return $k_a$

ENC(d, m_0, m_1)
  assert $k_a \neq \bot$
  assert $z \neq \bot$
  assert $|m_0| = |m_1|$
  if $z \neq d$ then
    $c \leftarrow \mathsf{ENC}(m_0, m_1)$
  if $z = d$ then
    $c \leftarrow\$ \; enc(k_a, m_0)$
  return $c$
```

Figure 8: Oracles of reduction package RED.

Proof. RED samples and stores the corrupt key k_a and answers all queries related to it, while queries regarding the honest key are forwarded to the IND-CPA game. We prove

$$\texttt{2CPA}^b(se) \overset{\text{code}}{\equiv} \texttt{RED} \to \texttt{IND-CPA}^b(se) \text{ for } b \in \{0,1\}. \quad (3)$$

Lemma 2 now follows from Lemma 1 by observing that with $\mathsf{G}^b_{big} := \texttt{2CPA}^b(se)$ and $\mathsf{G}^b_{sml} := \texttt{IND-CPA}^b(se)$, (3) corresponds to (1). It thus remains to prove (3). On a high-level, the state of each game $\texttt{2CPA}^b(se)$ consists of a bit z and two key $Z(0)$, $Z(1)$, all of which are stored in KEYS. In $\texttt{RED} \to \texttt{IND-CPA}^b(se)$ (shown in Fig. 9), on the other hand, the same state is split between RED and $\texttt{IND-CPA}^b(se)$: RED stores z and k_a acting as $Z(z)$, while $\texttt{IND-CPA}^b(se)$'s state is k (i.e. $Z(1-z)$). Moreover the stateless oracle ENC behaves identically in both games since encryption under the corrupt key always yields an encryption of m_0. We defer the formal inlining argument of (3) to Appendix A. □

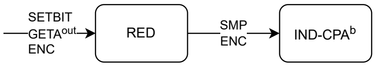


Figure 9: Games $\texttt{RED} \to \texttt{IND-CPA}^b(se)$.

E. Multi-instance assumptions

It is often convenient to consider multiple independent instances of an assumption at the same time. In this case, we add indices to package names and oracles to distinguish the instances. The BDFKK multi-instance lemma shows that single-instance security of a game implies multi-instance security of the same game. We here reproduce the lemma for $\texttt{2CPA}^b(se)$ to obtain its multi-instance version $\texttt{2CPA}^b_{1..n}(se)$.

Lemma 3. *([BDF+18, Appendix B, Lemma 38]) There exists a PPT reduction $\mathcal{R}_{se}$ such that for all PPT $\mathcal{A}$, we have that*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \texttt{2CPA}^0_{1..n}(se), \texttt{2CPA}^1_{1..n}(se))\\ &\leq n \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{se}; \texttt{2CPA}^0(se), \texttt{2CPA}^1(se)),\end{aligned}$$

where $\texttt{2CPA}^b_{1..n}(se)$ are n parallel copies of $\texttt{2CPA}^b(se)$, disambiguated by index i.

Definition 10 (2CPA security)**.** *A symmetric encryption scheme se is 2-key IND-CPA-secure or 2CPA-secure if for all PPT adversaries $\mathcal{A}$, the advantage*

$$\mathsf{Adv}(\mathcal{A}, \texttt{2CPA}^0_{1..n}(se), \texttt{2CPA}^1_{1..n}(se))$$

is negligible, where $\texttt{2CPA}^b_{1..n}(se)$ are n parallel copies of $\texttt{2CPA}^b(se)$, disambiguated by index i.

The following corollary combines the results of this section.

Corollary 1. *Let $\mathcal{R}_{2cpa} := \mathcal{R}_{se} \to \mathcal{R}_{cpa}$. For all PPT $\mathcal{A}$, we have that*

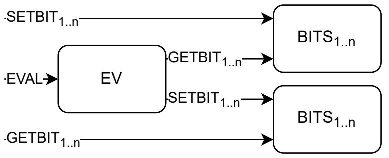
$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \texttt{2CPA}^0_{1..n}(se), \texttt{2CPA}^1_{1..n}(se))\\ &\leq n \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{2cpa}; \texttt{IND-CPA}^0(se), \texttt{IND-CPA}^1(se)).\end{aligned}$$

IV. State-Separated Garbling Schemes

We now apply the SSP approach (Section III) to garbling schemes and revisit their syntax, correctness and security.

A. Syntax and Correctness

Traditionally (including the SSP literature), cryptographic constructions are viewed as a tuple of algorithms, or alternatively Turing machines. Security and correctness are then described as games which invoke the different algorithms. In turn, in this work, we define the syntax of a garbling scheme as a tuple of *packages* (Definition 5). Recall from Def. 1 that BHR define a garbling scheme as tuple of algorithms (gb, en, gev, de, ev), where gb garbles a circuit C, en garbles an input x, gev evaluates a garbled circuit on a garbled input, de provides the output of the garbled circuit using decoding information obtained from the garbled evaluation (Def. 2) and ev is a simple circuit evaluation algorithm.



(a) Circuit evaluation game CEV with packages EV and BITS.

```
EV
EVAL($C$)
for $j = 1..n$ do
  $z_{0,j} \leftarrow$ GETBIT$_j$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $z_{i,j} \leftarrow op(z_{i\text{-}1,\ell}, z_{i\text{-}1,r})$
for $j = 1..d$ do
  SETBIT$_j(z_{d,j})$
return ()
```

```
BITS$_j$
SETBIT$_j(z)$
assert $z_j = \bot$
$z_j \leftarrow z$
return ()

GETBIT$_j$
assert $z_j \neq \bot$
return $z_j$
```

(b) Code of packages EV and BITS$_j$.

Figure 10: Graph and code of circuit evaluation package CEV.

1) Circuit evaluation: We start with an SSP version of circuit evaluation, game CEV. The EVAL oracle of game CEV takes a layered circuit of depth d and width n and corresponding to algorithm ev. The game provides oracles SETBIT for setting input bits, EVAL for evaluating the circuit, and GETBIT to obtain the result. To evaluate a circuit C on input x of length n, an adversary can query SETBIT n times with the individual bits of x, then EVAL with input C, and then can obtain the result by querying GETBIT n times.

Taking advantage of the fact that the composition of packages is again a package, we define CEV as composition of three packages: two BITS packages that model the input bits and output bits, respectively, and a circuit evaluation package EV which performs the actual computation. BITS is a simple package for storing bits (Fig. 10b). Package EV on the other hand is stateless and provides an oracle that evaluates an input circuit C on the bits stored in the top BITS package, and stores the result in bottom BITS package.

Definition 11 (Circuit evaluation)**.** *Circuit evaluation* CEV *for layered Boolean circuits of width n and depth d is defined in Fig. 10 and has*

$[\rightarrow \texttt{CEV}]$: SETBIT$_{1..n}$, EVAL, GETBIT$_{1..n}$ *and* $[\texttt{CEV} \rightarrow] = \emptyset$.

2) Garbling scheme syntax: Remember that BHR define a garbling scheme as tuple of algorithms gb, en, de, gev, ev. We capture the syntax of a garbling scheme in SSP style as a tuple of seven packages GB, EN, EKEYS, DE, DINF, GEV and EV, corresponding to the original algorithms plus shared state between them where convenient. The call graph in Fig. 12 composes the packages such that the adversary can use them meaningfully as follows: Garbling a circuit and input, then evaluating the garbled circuit on the garbled input and decoding the result.

Each package captures the algorithm of the same name, except for the two packages we added to capture shared state between algorithms: EKEYS which model input encoding information e as pairs of keys and DINF for output decoding information d. We consider only input projective garbling schemes, and thus, EN and EKEYS are fixed (cf. Fig. 11). Hence a garbling scheme is defined solely by providing the package tuple (GB, DE, DINF, GEV).

```
EKEYS$_j$
SETKEYS$_j(Z)$
assert $Z_j = \bot$
$Z_j \leftarrow Z$
return ()

GETKEYS$_j$
assert $Z_j \neq \bot$
return $Z_j$
```

```
EN$_j$
SETBIT$_j(z)$
assert $z_j = \bot$
$z_j \leftarrow z$
return ().

GETA$_j$
assert $z_j \neq \bot$
$Z \leftarrow$ GETKEYS$^{\text{in}}_j$
return $Z(z_j)$
```

Figure 11: Code of EKEYS$_j$ and EN$_j$.

3) Correctness: A garbling scheme (GB, DE, DINF, GEV) is correct if the game GCORR(GB, DE, DINF, GEV) (Fig. 12) behaves as CEV. Due to decryption ambiguities, a negligible statistical gap might exist between the two.

Definition 12 (Garbling Scheme)**.** *A packages tuple* gs = (GB, DE, DINF, GEV) *is a garbling scheme if the games* GCORR(GB, DE, DINF, GEV) *(Fig. 12) and* CEV *(Fig. 10) are statistically indistinguishable, i.e.,* Adv($\mathcal{A}$; GCORR(GB, DE, DINF, GEV), CEV) *is negligible for any adversary $\mathcal{A}$, and*

$[\rightarrow \texttt{GB}]$: GBL $[\texttt{GB} \rightarrow]$: SETKEYS$_{1..n}$, SETDINF,
$[\rightarrow \texttt{DE}]$: SETA$_{1..n}$, GETBIT $[\texttt{DE} \rightarrow]$: GETDINF,
$[\rightarrow \texttt{DINF}]$: SETDINF, GETDINF $[\texttt{DINF} \rightarrow]$: $\emptyset$,
$[\rightarrow \texttt{GEV}]$: EVAL $[\texttt{GEV} \rightarrow]$: GETA$^{out}_{1..n}$, SETA$_{1..n}$, GBL.

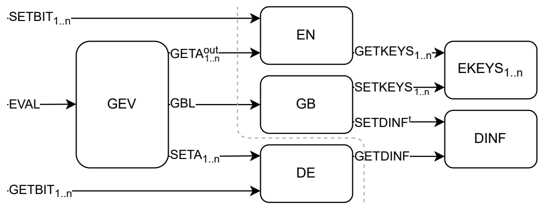


Figure 12: Real garbling scheme correctness game GCORR(GB, DE, DINF, GEV).

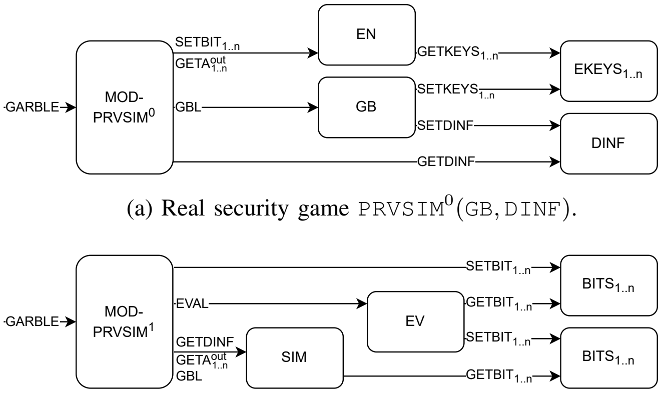
B. Security

We now encode the garbling scheme security games PRVSIM$^b_{\mathsf{gs},\Phi,\mathcal{S}}$ from Section II-A using packages, with fixed leakage $\Phi(C) = C$. The package MOD-PRVSIM^b (Fig. 13) models the core of the game: It provides the expected interface GARBLE to the adversary and calls the garbling scheme's oracles in the intended order.

Upon a query with input C and x from the adversary, GARBLE stores x in another package via SETBIT queries and then obtains the circuit garbling, input encoding and output decoding. In the real game $\texttt{PRVSIM}^0(\texttt{GB}, \texttt{DINF})$, the garbling is performed by the garbling scheme (Fig. 14a). The ideal game $\texttt{PRVSIM}^1(\texttt{SIM})$ is parametrized by a simulator $\texttt{SIM}$ that computes the garbling instead, given access to the output of circuit evaluation but not the input x itself (Fig. 14b). Security then demands the existence of an efficient simulator $\texttt{SIM}$ such that the real and ideal game are indistinguishable for every efficient adversary.

```
$\texttt{MOD-PRVSIM}^b$
$\mathsf{GARBLE}(C, x)$
for $j = 1..n$ do
    $\mathsf{SETBIT}_j(x_j)$
if $b = 1$ then
    $\mathsf{EVAL}(C)$
$\tilde{C} \leftarrow \mathsf{GBL}(C)$
dinf $\leftarrow$ $\mathsf{GETDINF}$
for $j = 1..n$ do
    $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\text{out}}_j$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

Figure 13: Code of $\texttt{MOD-PRVSIM}^b$.



(a) Real security game $\texttt{PRVSIM}^0(\texttt{GB}, \texttt{DINF})$.

(b) Ideal security game $\texttt{PRVSIM}^1(\texttt{SIM})$.

Figure 14: Games $\texttt{PRVSIM}^0(\texttt{GB}, \texttt{DINF})$ and $\texttt{PRVSIM}^1(\texttt{SIM})$.

Definition 13 (Garbling scheme security)**.** *Let* $\mathsf{gs} = (\texttt{GB}, \texttt{DE}, \texttt{DINF}, \texttt{GEV})$ *be a garbling scheme.* gs *is secure if there exists a PPT simulator* $\texttt{SIM}$ *such that for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}(\mathcal{A}; \texttt{PRVSIM}^0(\texttt{GB}, \texttt{DINF}), \texttt{PRVSIM}^1(\texttt{SIM}))$$

is negligible. See Fig. 14 for the definitions of games $\texttt{PRVSIM}^0(\texttt{GB}, \texttt{DINF})$ *and* $\texttt{PRVSIM}^1(\texttt{SIM})$*.*

V. Yao's Garbling Scheme

After introducing a garbling scheme notion, we now turn to Yao's garbling scheme as concrete example. We present an informal overview, state security and provide a proof overview.

A. Overview

Yao's construction uses an IND-CPA secure symmetric encryption scheme $(kgen, enc, dec)$ where $kgen$ selects uniformly random bitstrings as keys[2].

1) Circuit garbling: To garble a circuit C of depth d with n inputs and width d, Yao's garbling scheme starts by choosing two uniformly random bitstrings per gate storing them as $Z_{i,j}(0)$ and $Z_{i,j}(1)$, respectively. Here, 0 and 1 is a bit associated with the key, $0 \leq i \leq d$ describes the depth of the gate and $0 \leq j \leq n$ describes the index of the gate within a layer. W.l.o.g. and for our convenience, we assume that circuits are layered and have constant width. Now, for a gate $g_{i,j}$ with operation $op_{i,j}$, denote by $Z_{i-1,\ell}$, $Z_{i-1,r}$ the indices of the gates which compute the left and right input to $g_{i,j}$. Now, Yao's garbling scheme computes four ciphertexts encrypting the output wire keys under the input wire keys according to $op_{i,j}$ as follows:

$$\begin{aligned}
c_0 &= enc_{Z_{i-1,r}(0)}(enc_{Z_{i-1,\ell}(0)}(Z_{i,j}(op_{i,j}(0,0)))),\\
c_1 &= enc_{Z_{i-1,r}(1)}(enc_{Z_{i-1,\ell}(0)}(Z_{i,j}(op_{i,j}(0,1)))),\\
c_2 &= enc_{Z_{i-1,r}(0)}(enc_{Z_{i-1,\ell}(1)}(Z_{i,j}(op_{i,j}(1,0)))),\\
c_3 &= enc_{Z_{i-1,r}(1)}(enc_{Z_{i-1,\ell}(1)}(Z_{i,j}(op_{i,j}(1,1)))).
\end{aligned}$$

The garbled gate $\tilde{g}_{i,j}$ consists of the ciphertexts $c_0, \ldots, c_3$ arranged so that the computation order is hidden, and the garbled circuit $\tilde{C}$ consists of the $d \cdot n$ garbled gates $(\tilde{g}_{i,j})_{1 \leq i \leq d, 1 \leq j \leq n}$ and the output decoding information $Z_{d,1}$,..,$Z_{d,n}$.

2) Input encoding: For each bit x_i of input x, Yao's garbling scheme returns the corresponding input wire key on the ith input wire, i.e., the input encoding information is $Z_{0,1}(x_1)$,..,$Z_{0,n}(x_n)$ for input $x = x_1||..||x_n$.

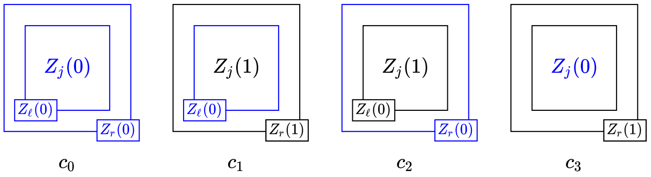
3) Circuit evaluation: Given garbled circuit $\tilde{C}$ and encoded input $\tilde{x}$, the garbled circuit is evaluated as follows: For each gate $\tilde{g}_{i,j}$, let $k_{i-1,\ell}$ and $k_{i-1,r}$ be the wire keys corresponding to left and right input wire (either obtained from $\tilde{x}$ or a previous gate evaluation). Then attempt to decrypt each of the four gate ciphertexts with $k_{i-1,\ell}$ and $k_{i-1,r}$. If the circuit was garbled correctly, exactly one will decrypt to the desired output wire key $k_{i,j}$ without error, except with negligible probability.

4) Output decoding: For each key $k_{d,j}$, $1 \leq j \leq n$, return $y_{d,j}$ such that $Z_{d,j}(y_{d,j}) = k_{d,j}$.

5) Security: To prove security, we will attach different semantics to the wire keys. Real garbling uses 0/1 semantics, i.e. each key is mapped to a bit. Intuitively, garbling scheme security holds because an adversary will only ever learn *one* key per wire, referred to as the *active*. If we manage to switch completely from 0/1 key semantics to active/inactive, we can simulate garbling without knowledge of the input. The reason why the adversary learns one key per wire is as follows. For each input wire to a gate, the adversary only knows one of the two wire keys. Thus for each gate garbling, they can decrypt exactly one out of the four ciphertexts. I.e., given two active keys for the input wires, the adversary (only) learns the active

[2] The encryption scheme is assumed to satisfy further properties to achieve correctness, which we omit here due to our focus on garbling scheme security. In a nutshell, an adversary cannot generate a valid ciphertext from the ciphertext space under a random but unknown key, and the encryption scheme returns a special error symbol $\perp$ if decryption fails due to the use of an incorrect decryption key. See [LP09, Definition 2] for details.

key for the output wire of a circuit. To see this, let us consider the xor operation as an example, and let us say that for the left input wire, the 0-key is active (known to the adversary) and for the right input wire, also the 0-key is active. The 4 ciphertexts can be illustrated as follows:



The adversary only knows the blue key and thus can only recover the blue key. This observation generalizes to arbitrary operations, since there are four ways[3] to combine left active/inactive and right active/inactive key so that the adversary always only learns one ciphertext—there is only one ciphertext which can be represented by two nested blue squares. The adversary always learns the active output key, because if b_ℓ and b_r are the active bits, then we encrypt $Z_j(op(b_\ell, b_r)$ under $Z(b_\ell)$ and $Z(b_r)$—which is the active key. Applying this argument recursively yields the desired security statement.

B. Security

Following Section IV-A, the traditional version of Yao's garbling scheme can be defined as package tuple

$$\mathsf{gs}_{\text{tdyao}} = (\mathtt{GB}_{\text{tdyao}}, \mathtt{DE}_{\text{tdyao}}, \mathtt{DINF}_{\text{tdyao}}, \mathtt{GEV}_{\text{tdyao}}).$$

Since the behaviour of garbled evaluation package $\mathtt{GEV}_{\text{tdyao}}$ and output decoding package $\mathtt{DE}_{\text{tdyao}}$ are conceptually induced by the behaviour of the garbling package $\mathtt{GB}_{\text{tdyao}}$, and since the security definition only depends on $\mathtt{GB}_{\text{tdyao}}$ and $\mathtt{DINF}_{\text{tdyao}}$, we omit the description of the former. $\mathtt{GB}_{\text{tdyao}}$ has a single oracle GBL, and $\mathtt{DINF}_{\text{tdyao}}$ is similar to $\mathtt{EKEYS}$. Both are shown in Fig. 15. To garble a circuit C, oracle GBL first performs some checks on the inputs, then samples keys for all wires, parses the circuit layer by layer, garbles each gate, and eventually returns the garbled circuit $\tilde{C}$. We assume that sets are encoded by ordering their elements in lexicographic ordering to hide the order of ciphertexts comprising a garbled gate. The remainder of this paper proves the security of the Yao's garbling scheme as defined in Section IV-B:

Theorem 1 (Security of Yao's garbling scheme)**.** *Let $\mathcal{A}$ be a PPT adversary, let d be a polynomial upper bound on the depth of the circuit which $\mathcal{A}$ chooses, let n denote the width of the circuit and let se denote the symmetric encryption scheme used within gs_{tdyao} (Fig. 15). Then, there exists a PPT simulator $\mathtt{SIM}_{tdyao}$ and reduction $\mathcal{R}$ such that*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{PRVSIM}^0(\mathtt{GB}_{tdyao}, \mathtt{DINF}_{tdyao}), \mathtt{PRVSIM}^1(\mathtt{SIM}_{tdyao}))\\ &\leq dn \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}; \mathtt{IND\text{-}CPA}^0(se), \mathtt{IND\text{-}CPA}^1(se)). \quad (4)\end{aligned}$$

Thus if se is IND-CPA secure, then gs_{tdyao} is secure.

Oracle of $\mathtt{GB}_{\text{tdyao}}$

```
GBL(C)
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $i = 0..d$ do
  for $j = 1..n$ do
    $Z_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    for $(b_\ell, b_r) \in \{0,1\}^2$ do
      $b_j \leftarrow op(b_\ell, b_r)$
      $k_j \leftarrow Z_{i,j}(b_j)$
      $c_{\mathsf{in}} \leftarrow\$ enc(Z_{i,\ell}(b_\ell), k_j)$
      $c \leftarrow\$ enc(Z_{i,r}(b_r), c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}[i,j] \leftarrow \tilde{g}_j$
for $j = 1..n$ do
  $\mathsf{SETKEYS}_j(Z_{0,j})$
$\mathsf{SETDINF}(Z_{d,1}, \ldots, Z_{d,n})$
return $\tilde{C}$
```

Oracles of $\mathtt{DINF}_{\text{tdyao},j}$

```
SETDINF(dinf)
dinf ← dinf
return ()
```

```
GETDINF
return dinf
```

Oracles of $\mathtt{SIM}_{\text{tdyao}}$

```
GBL(C)
for $j = 1..n$ do
  $S_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $S_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $S_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $S_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^\lambda$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
return $\tilde{C}$
```

```
GETDINF
for $j = 1..n$ do
  $z_{d,j} \leftarrow \mathsf{GETBIT}_j$
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  $\mathsf{dinf}[j] \leftarrow Z_{d,j}$
return dinf
```

```
GETA_j
return $S_{0,j}(0)$
```

Figure 15: Code of $\mathtt{GB}_{\text{tdyao}}$ (top left), $\mathtt{DINF}_{\text{tdyao}}$ (lower left) and $\mathtt{SIM}_{\text{tdyao}}$ (right).

Looking ahead, the constructed simulator $\mathtt{SIM}_{\text{tdyao}}$ (shown for completeness in Fig. 15) samples wire keys in $S_{i,j}$ with active/inactive semantics and thus independently of any real input. To garble a gate, $\mathtt{SIM}_{\text{tdyao}}$ encrypts the active output wire key $S_{i,j}(0)$ under both active input wire keys $S_{i\text{-}1,\ell}(0)$, $S_{i\text{-}1,r}$, and 0^λ under all other input wire key combinations.

C. Proof outline

We use symmetric encryption security once for each gate in the circuit (cf. Section V-A5). Afterwards, the proof connects the *gate* garbling arguments and turns them into an argument about *circuit* garbling, and relate them to the security statement. The proof introduces an alternative representation of Yao's garbling scheme, annotated by index *yao*. Technically, the proof proceeds as follows.

[3] active/active, active/inactive, inactive/active, inactive/inactive

```
Oracle of $\mathtt{MODGB}_i$

$\mathsf{GBL}_i(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op})$
assert $\tilde{C} = \bot$
assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
for $j = 1..n$ do
    $(\ell, r, op) \leftarrow$
    $(\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{C}_j \leftarrow \mathsf{GBLG}(\ell, r, op, j)$
$\tilde{C} \leftarrow \tilde{C}_{1..n}$
return $\tilde{C}$
```

```
Oracle of $\mathtt{GATE}$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{C}_j \leftarrow \bot$
$Z^{\text{out}}_j \leftarrow \mathsf{GETKEYS}^{\text{out}}_j$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
    $b_j \leftarrow op(b_\ell, b_r)$
    $k^0_j \leftarrow Z^{\text{out}}_j(b_j)$
    $c^0_{\mathsf{in}} \leftarrow \mathsf{ENC}_\ell(b_\ell, k^0_j, 0^\lambda)$
    $c^1_{\mathsf{in}} \leftarrow \mathsf{ENC}_\ell(b_\ell, 0^\lambda, 0^\lambda)$
    $c \leftarrow \mathsf{ENC}_r(b_r, c^0_{\mathsf{in}}, c^1_{\mathsf{in}})$
    $\tilde{C}_j \leftarrow \tilde{C}_j \cup \{c\}$
return $\tilde{C}_j$
```

```
Oracle of $\mathtt{SIM}_{\mathsf{gate}}$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
$\mathsf{EVAL}_j(\ell, r, op)$
$S^{\text{out}}_j(0) \leftarrow \mathsf{GETA}^{\text{out}}_j$
$S^{\text{in}}_r(0) \leftarrow \mathsf{GETA}^{\text{in}}_r$
$S^{\text{in}}_r(1) \leftarrow \mathsf{GETINA}^{\text{in}}_r$
$S^{\text{in}}_\ell(0) \leftarrow \mathsf{GETA}^{\text{in}}_\ell$
$S^{\text{in}}_\ell(1) \leftarrow \mathsf{GETINA}^{\text{in}}_\ell$
for $(d_\ell, d_r) \in \{0,1\}^2$ :
    $k^{\text{in}}_\ell \leftarrow S^{\text{in}}_\ell(d_\ell)$
    $k^{\text{in}}_r \leftarrow S^{\text{in}}_r(d_r)$
    if $d_\ell = d_r = 0$ :
        $k^{\text{out}}_j \leftarrow S^{\text{out}}_j(0)$
    else $k^{\text{out}}_j \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\!\!\$\ enc(k^{\text{in}}_r, k^{\text{out}}_j)$
    $c \leftarrow\!\!\$\ enc(k^{\text{in}}_\ell, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$
```

Figure 16: Code of $\mathtt{MODGB}_i$, $\mathtt{GATE}$ and $\mathtt{SIM}_{\mathsf{gate}}$.

Encryption scheme security (Section III): We introduced a 2-key multi-instance version of IND-CPA security and reduced it to single-instance IND-CPA (Corollary 1).

Layer garbling security (Section VI): We define security of Yao's *layer garbling* and and reduce it to our 2-key multi-instance IND-CPA security notion (Lemma 4).

Circuit garbling security (Section VII): We show that the layer security notion *self-composes* and, via a hybrid argument, implies security of the circuit garbling (Lemma 5). We here use a *modular* security notion for circuit garbling which allows the adversary to garble the input *before* garbling the circuit (Fig. 21).

Standard garbling scheme security (Section VIII): We show that composable circuit security for Yao's garbling scheme implies $\mathtt{PRVSIM}$-security of $\mathtt{GB}_{\text{tdyao}}$.

VI. Layer Security

Consider a symmetric encryption scheme *se* with 2-key IND-CPA security as defined in Section III. In this section, we extend the encryption scheme to layer garbling for Yao's garbling scheme. We then define layer garbling security and reduce it to 2-key IND-CPA security.

A. Yao's layer garbling package $\mathtt{GB}^0_{yao,i}$

Remember that in order to garble a circuit layer, we need to garble each gate, using encryption where the keys and message for each ciphertext depend on the gate description. The layer garbling package $\mathtt{GB}^0_{\text{yao},i}$ reflects this structure: Each $\mathtt{GB}^0_{\text{yao},i}$ is composed of the packages $\mathtt{MODGB}_i$, $\mathtt{GATE}$ and $\mathtt{ENC}^0_{1\ldots n}$ (cf. Fig. 16 and 17). $\mathtt{GATE}$ garbles a gate and makes (simple) encryption queries to $\mathtt{ENC}^0_{1\ldots n}$. $\mathtt{MODGB}_i$ modularizes the garbling of a layer and queries oracle GBLG of $\mathtt{GATE}$ for each gate in the layer.

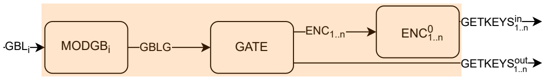


Figure 17: Layer garbling package $\mathtt{GB}^0_{\text{yao},i}$

Definition 14 (Yao's Layer Garbling)**.** *Let $i \in \mathbb{N}$. We define the circuit layer garbling package $\mathtt{GB}^0_{yao,i}$ as*

$$\mathtt{GB}^0_{yao,i} := \mathtt{MODGB}_i \to \mathtt{GATE} \to \mathtt{ENC}^0_{1\ldots n}$$

where Fig. 16 defines $\mathtt{MODGB}_i$ and $\mathtt{GATE}$, Fig. 7 defines $\mathtt{ENC}^0_{1\ldots n}$ and $\mathtt{KEYS}^0_{1\ldots n}$, and Fig. 17 composes them.

B. Layer Garbling Security Definition

We define layer security as indistinguishability between two games. The ideal game is parametrized by an (existentially quantified) simulator $\mathtt{GB}^1_{\text{yao},i}$ and the real game uses layer garbling package $\mathtt{GB}^0_{\text{yao},i}$ which specifies a layer garbling scheme for layer i. To be able to define the ideal game, we extend package $\mathtt{KEYS}$ (cf. Fig. 7) with further oracles and provide a layer evaluation package $\mathtt{LEV}$ as shown in Fig. 18.

```
$\mathtt{LEV}_j$

$\mathsf{EVAL}_j(\ell, r, op)$
$z_\ell \leftarrow \mathsf{GETBIT}_\ell$
$z_r \leftarrow \mathsf{GETBIT}_r$
$z_j \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
$\mathsf{SETBIT}_j(z_j)$
```

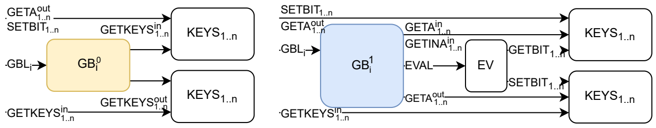
```
$\mathtt{KEYS}$

$\mathsf{GETKEYS}^{\text{out}}$
bflag $\leftarrow 1$
if $Z = \bot$ then
    $Z(0) \leftarrow\!\!\$\ \{0,1\}^\lambda$
    $Z(1) \leftarrow\!\!\$\ \{0,1\}^\lambda$
return $Z$

$\mathsf{GETA}^{\text{in}}$
assert aflag
return $Z(z)$

$\mathsf{GETINA}^{\text{in}}$
assert aflag
return $Z(1-z)$
```

Figure 18: Oracles of $\mathtt{LEV}$, and additional oracles of $\mathtt{KEYS}$.

Definition 15 (Layer Security)**.** *Let $i \in \mathbb{N}$. Layer garbling package $\mathtt{GB}^0_{yao,i}$ is secure if there exists a PPT layer simulator $\mathtt{GB}^1_{yao,i}$ such that for all PPT adversaries $\mathcal{B}$,*

$$\mathsf{Adv}(\mathcal{B}; \mathtt{LSEC}^0(\mathtt{GB}^0_{yao,i}), \mathtt{LSEC}^1(\mathtt{GB}^1_{yao,i}))$$

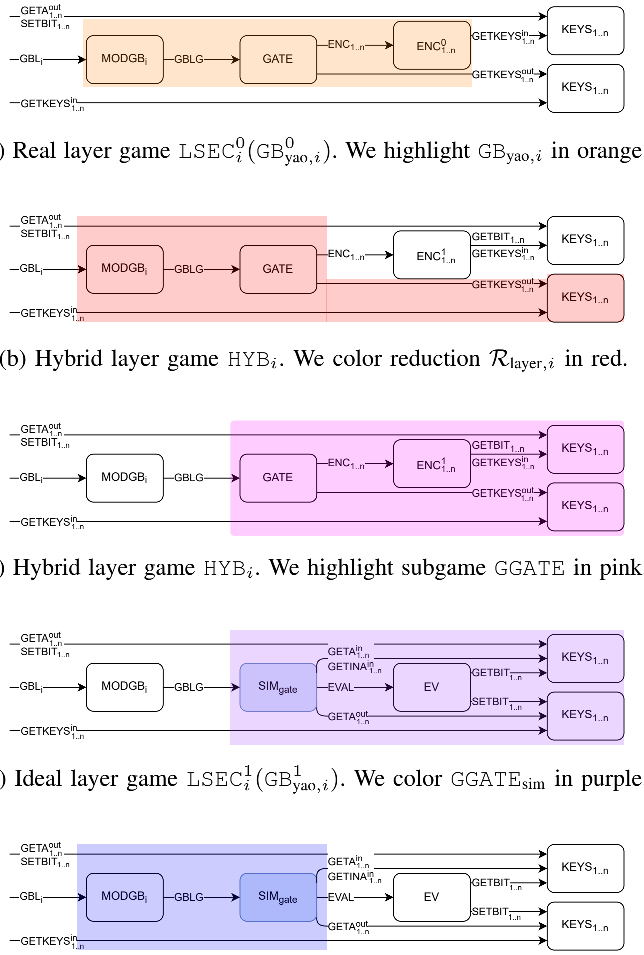
is negligible, where Fig. 19a defines $\mathtt{LSEC}^0(\mathtt{GB}^0_{yao,i})$ and Fig. 19b defines $\mathtt{LSEC}^1(\mathtt{GB}^1_{yao,i})$.



(a) Real layer sec. game $\mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao,i}})$.

(b) Ideal layer security game $\mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao,i}})$.

Figure 19: Layer security games.


$\text{GETA}^{\text{out}}_{1..n}$
$\text{SETBIT}_{1..n}$
GBL_i
MODGB_i
GBLG
GATE
$\text{ENC}_{1..n}$
$\text{ENC}^0_{1..n}$
$\text{GETKEYS}^{\text{in}}_{1..n}$
$\text{GETKEYS}^{\text{out}}_{1..n}$
$\text{KEYS}_{1..n}$
$\text{KEYS}_{1..n}$
$\text{GETKEYS}^{\text{in}}_{1..n}$
$\text{ENC}^1_{1..n}$
$\text{GETBIT}_{1..n}$
SIM_{gate}
$\text{GETA}^{\text{in}}_{1..n}$
$\text{GETINA}^{\text{in}}_{1..n}$
EVAL
EV
$\text{GETA}^{\text{out}}_{1..n}$

(a) Real layer game $\texttt{LSEC}^0_i(\texttt{GB}^0_{\text{yao},i})$. We highlight $\texttt{GB}_{\text{yao},i}$ in orange.

(b) Hybrid layer game $\texttt{HYB}_i$. We color reduction $\mathcal{R}_{\text{layer},i}$ in red.

(c) Hybrid layer game $\texttt{HYB}_i$. We highlight subgame $\texttt{GGATE}$ in pink.

(d) Ideal layer game $\texttt{LSEC}^1_i(\texttt{GB}^1_{\text{yao},i})$. We color $\texttt{GGATE}_{\text{sim}}$ in purple.

(e) Ideal layer game $\texttt{LSEC}^0_i(\texttt{GB}^1_{\text{yao},i})$. We mark sim. $\texttt{GB}^1_{\text{yao},i}$ in blue.

Figure 20: Layer security games and hybrids for Lemma 4.

The games $\texttt{LSEC}^0(\texttt{GB}^0_{\text{yao},i})$ and $\texttt{LSEC}^1(\texttt{GB}^1_{\text{yao},i})$ can be seen as layer version of the selective security games $\texttt{PRVSIM}^0(\texttt{GB}_{\text{tdyao}}, \texttt{DINF}_{\text{tdyao}})$ and $\texttt{PRVSIM}^1(\texttt{SIM}_{\text{tdyao}})$, modified for our composition goals:

- The adversary inputs a *single circuit layer* to the game,
- the adversary's query to the game is split into SETBIT, GBL, GETA^{out} and GETDINF queries,
- input/output keys and bits are stored in $\texttt{KEYS}$ packages,
- the simulator gets the input keys via $\text{GETINA}^{\text{in}}$ and GETA^{in} queries from the top $\texttt{KEYS}$ package and the *active* output key via GETA^{out} from the lower $\texttt{KEYS}$ package rather than sampling them itself.

Interestingly, the layer garbling security game even allows the adversary to query GETA before GBL and thus obtain the input garbling *before* choosing the circuit. This feature as well as the aforementioned splitting of queries will be useful for (self-)composability to which we turn in Section VII.

C. Security Reduction to 2-key IND-CPA security

We now prove that security of $\texttt{GB}^0_{\text{yao,i}}$ reduces to 2-key IND-CPA security of the underlying encryption scheme se.

Lemma 4 (Layer Security)**.** *Let $i \in \mathbb{N}$. Let $\mathcal{R}_{layer,i}$ be the reduction defined in Figure 20b, $\texttt{GB}_{yao,i}$ as defined in Fig. 17 and $\texttt{GB}^1_{yao,i}$ as defined in Fig. 20e and 16. Then for all PPT adversaries $\mathcal{A}$,*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \texttt{LSEC}^0(\texttt{GB}_{yao,i}), \texttt{LSEC}^1(\texttt{GB}^1_{yao,i}))\\ =&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{layer}; \texttt{2CPA}^0_{1..n}(se), \texttt{2CPA}^1_{1..n}(se)).\end{aligned}$$

Gate garbling simulator $\texttt{SIM}_{\textsf{gate}}$ (Fig. 16) works as follows: Instead of garbling a gate based on the 0/1 semantics of wire keys like the real $\texttt{GB}^0_{\text{yao,i}}$, it uses their inactive/active semantics. The simulator first retrieves all relevant wire keys except for the inactive output wire key. One ciphertext, the one containing the active output key, is computed honestly using the left and right active input keys. The remaining ciphertexts are generated by encrypting the all-zero key. Simulator $\texttt{GB}^1_{\text{yao,i}} := \texttt{MODGB}_i \to \texttt{SIM}_{\textsf{gate}}$ extends this to layer garbling.

Proof of Lemma 4. Let $\mathcal{A}$ be an adversary. We wish to apply the perfect reduction lemma (Lemma 1). For this purpose, we prove two claims:

Claim 1 (Real Code Equivalence)**.** *$\forall 1 \le i \le d$, it holds that*

$$\texttt{LSEC}^0(\texttt{GB}^0_{yao,i}) \overset{code}{\equiv} \mathcal{R}^i_{layer} \to \texttt{2CPA}^0_{1..n}(se),$$

where $\mathcal{R}^i_{layer}$ is defined in Figure 20b.

Claim 1 follows by definition of $\texttt{GB}^0_{\text{yao},i}$ and observing that Fig. 20a and Fig. 20b indeed present the same package composition, merely with different highlighting.

Claim 2 (Ideal Code Equivalence)**.** *$\forall 1 \le i \le d$, it holds that*

$$\texttt{LSEC}^1(\texttt{GB}^1_{yao,i}) \overset{code}{\equiv} \mathcal{R}^i_{layer} \to \texttt{2CPA}^1_{1..n}(se),$$

where $\mathcal{R}^i_{layer}$ is defined in Figure 20b.

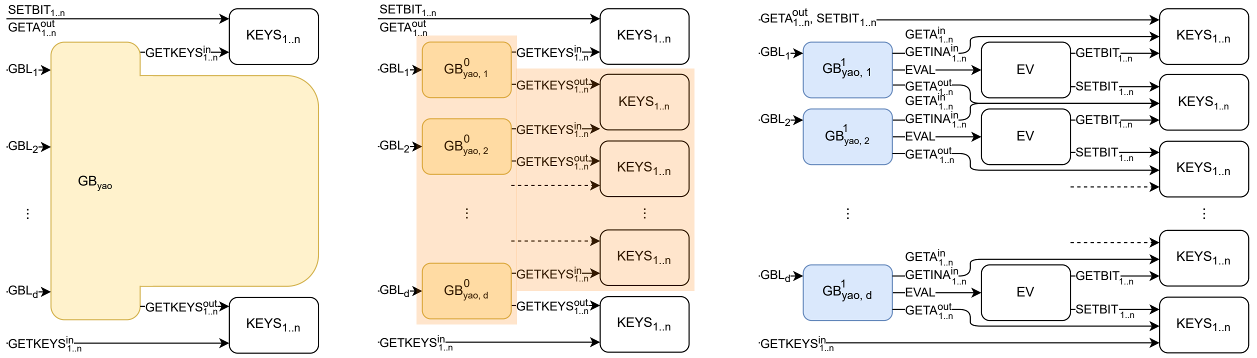
Claim 2 will be proved in a moment. Applying the perfect reduction lemma with Claims 1 and 2 concludes our proof. □

Claim 2 is the technical heart of the proof in which the semantics of keys used to garble a gate is switched: From 0/1 to active/inactive semantics, the latter being independent of the input to the layer and hence a simulation.

Proof of Claim 2. We need to show code equivalence of $\mathcal{R}^i_{\text{layer}} \to \texttt{IND-CPA}^1_{1..n}(se)$ and $\texttt{LSEC}^1(\texttt{GB}^1_{\text{yao},i})$. In a first step, we define real and ideal gate garbling subgames $\texttt{GGATE}$ and $\texttt{GGATE}_{\text{sim}}$ (Fig. 20c and Fig. 20d). If we can show that

$$\texttt{GGATE} \overset{\text{code}}{\equiv} \texttt{GGATE}_{\text{sim}}, \tag{5}$$

then the layer garbling games in Fig. 20b and Fig. 20e are functionally equivalent and we obtain Claim 2. The proof of Equation 5 is an inlining argument that switches which wire key semantics used to garble the layer from 0/1 to active/inactive semantics, see Appendix B for details. In a nutshell, the argument first inlines all packages, then changes the garbling of a gate from encrypting under all combinations of input wire keys using 0/1 semantics to the equivalent computation using active/inactive semantics, and then factors out the relevant packages again. □



(a) (b) (c)

Figure 21: Fig. 21a-21b display game $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$ and package $\mathtt{GB}_{\text{yao}}$ (orange) in different representations. Fig. 21c defines $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$ where $\mathtt{SIM}_{\text{yao}}$ is the parallel composition of the $\mathtt{GB}^1_{\text{yao},i}$ packages (blue).

VII. Circuit Security

A. *Yao's circuit garbling package* $\mathtt{GB}_{yao}$

Circuit garbling extends layer garbling by composing layer garbling packages and share state through $\mathtt{KEYS}$:

Definition 16 (Yao's Layer Garbling)**.** *Let $d \in \mathbb{N}$. We define the circuit garbling package $\mathtt{GB}^0_{yao}$ as the composition of layer garbling packages $\mathtt{GB}^0_{yao,1}, \ldots, \mathtt{GB}^0_{yao,d}$ with $\mathtt{KEYS}$ packages as shown in Fig. 21a-21b.*

B. *Circuit garbling security*

Analogous to layer garbling security, we define circuit garbling security as indistinguishability of two games: Real game $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$ in Fig. 21a-21b can be seen as composition of multiple real layer security games $\mathtt{LSEC}^0(\mathtt{GB}_{\text{yao},i})$ that overlap in their $\mathtt{KEYS}$ packages. Similarly, the composition of multiple ideal layer security games $\mathtt{LSEC}^1(\mathtt{SIM}_{\text{yao},i})$ in Fig. 21c defines the ideal game $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$.

Lemma 5 (Circuit Security)**.** *Let d be a polynomial upper bound on the depth of the circuit which $\mathcal{A}$ chooses. Then, for each $1 \leq i \leq d$, there exists a PPT reduction $\mathcal{R}^i_{circ}$ such that for all PPT adversaries $\mathcal{A}$,*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{SEC}^0(\mathtt{GB}_{yao}), \mathtt{SEC}^1(\mathtt{SIM}_{yao}))\\ &\leq \sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{circ}; \mathtt{LSEC}^0(\mathtt{GB}^0_{yao,i}), \mathtt{LSEC}^1(\mathtt{SIM}_{yao,i})),\end{aligned}$$

for $\mathtt{SEC}^0(\mathtt{GB}_{yao})$ and $\mathtt{SEC}^1(\mathtt{SIM}_{yao})$ in Fig. 21a and 21c.

Proof of Lemma 5. We reduce circuit garbling security to layer garbling security via a hybrid argument over the d layers of the circuit. The game-hopping argument starts with hybrid 0, which is $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$, and gradually makes modifications until reaching hybrid d, which is $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$. The first step rewrites $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$ and $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$ as

$$\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}}) \overset{\text{code}}{\equiv} \mathcal{R}^1_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},1}) \tag{6}$$

$$\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}}) \overset{\text{code}}{\equiv} \mathcal{R}^d_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},d}), \tag{7}$$

where Fig. 22b and Fig. 22e define $\mathcal{R}^1_{\text{circ}}$ and $\mathcal{R}^d_{\text{circ}}$, respectively. Both equivalences hold by associativity of package composition (cf. Fig. 22b and Fig. 22e). The definition of $\mathcal{R}^1_{\text{circ}}$ and $\mathcal{R}^d_{\text{circ}}$ can be generalized to obtain $\mathcal{R}^i_{\text{circ}}$ for all $i \in \{1,..,d\}$ as in Fig. 22c. Given $\mathcal{R}^i_{\text{circ}}$, we can now define the hybrid games between the 0-th hybrid $\mathcal{R}^1_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},1})$ and the d-th hybrid $\mathcal{R}^d_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},d})$. Namely, for $i \in \{1,..,d\text{-}1\}$, we define the i-th hybrid in the following two equivalent ways:

$$\mathcal{R}^i_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},i}) \overset{\text{code}}{\equiv} \mathcal{R}^{i+1}_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},i+1}) \tag{8}$$

Fig. 22c and 22d show that the two games are indeed equivalent. With Equations 6, 7 at hand, we can prove Lemma 5:

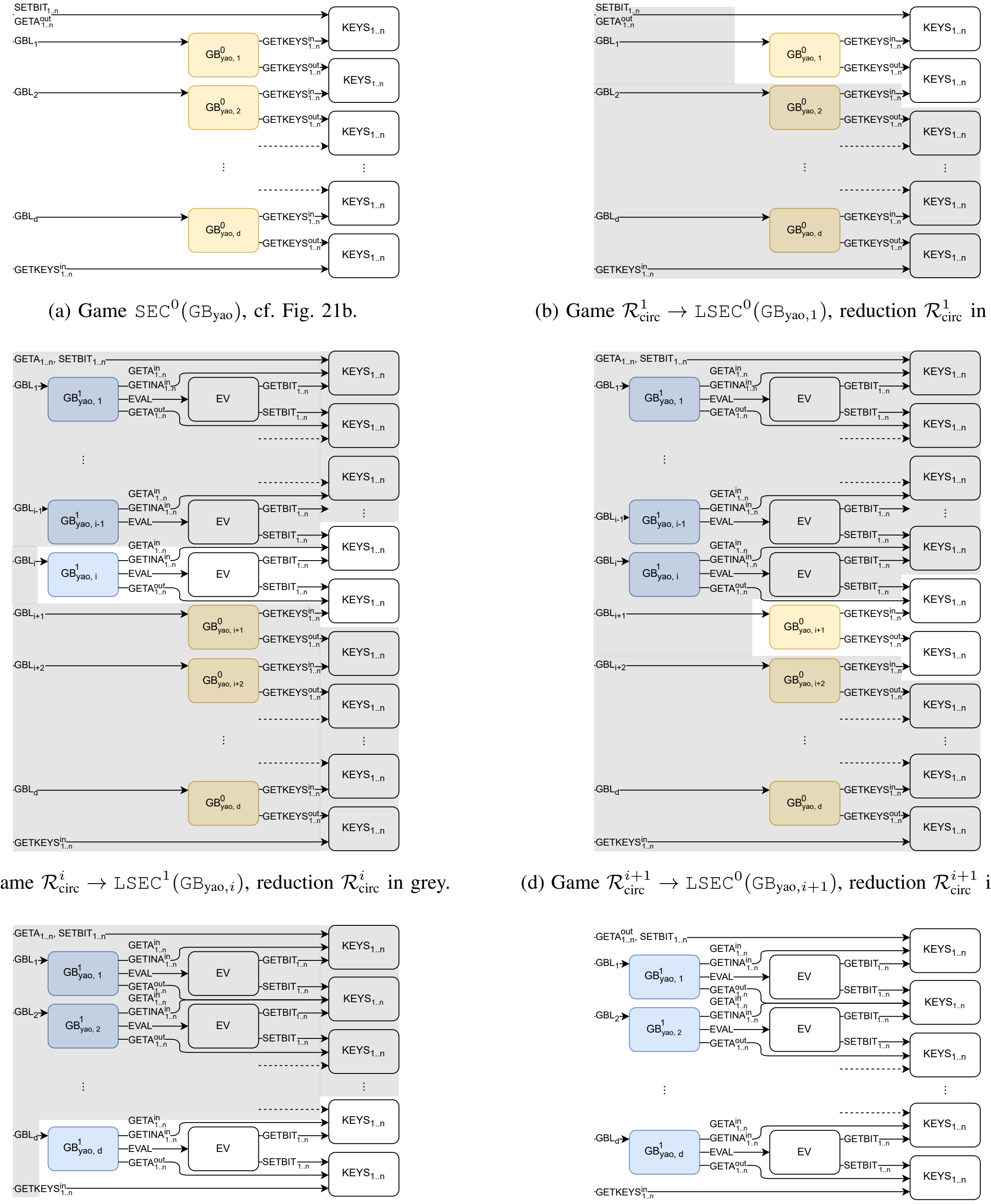
$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}}), \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}}))\\ =\ &\Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})\big] - \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})\big]\\ \overset{(6),(7)}{=}\ &\Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^1_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},1})\big]\\ &- \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^d_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},d})\big]\end{aligned}$$

Applying a telescopic sum and (8) then yields

$$\begin{aligned}&\Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^1_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},1})\big]\\ &+ \Big(\sum_{i=1}^{d-1} - \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^i_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},i})\big]\\ &\qquad + \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^{i+1}_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},i+1})\big]\Big)\\ &- \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^d_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},d})\big]\\ =\ &\sum_{i=1}^{d} \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^i_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},i})\big]\\ &- \sum_{i=1}^{d} \Pr\big[1 \leftarrow\$\ \mathcal{A} \to \mathcal{R}^i_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}^1_{\text{yao},i})\big]\\ =\ &\sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{\text{circ}}; \mathtt{LSEC}^0(\mathtt{GB}^0_{\text{yao},i}), \mathtt{LSEC}^1(\mathtt{SIM}_{\text{yao},i})).\end{aligned}$$

□

Combining Lemma 5 (Circuit security), Lemma 4 (Layer security) and Corollary 1 (2CPA), we obtain the following useful corollary.



(a) Game $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$, cf. Fig. 21b.

(b) Game $\mathcal{R}^1_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}_{\text{yao},1})$, reduction $\mathcal{R}^1_{\text{circ}}$ in grey.

(c) Game $\mathcal{R}^i_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}_{\text{yao},i})$, reduction $\mathcal{R}^i_{\text{circ}}$ in grey.

(d) Game $\mathcal{R}^{i+1}_{\text{circ}} \to \mathtt{LSEC}^0(\mathtt{GB}_{\text{yao},i+1})$, reduction $\mathcal{R}^{i+1}_{\text{circ}}$ in grey.

(e) Game $\mathcal{R}^d_{\text{circ}} \to \mathtt{LSEC}^1(\mathtt{GB}_{\text{yao},d})$, reduction $\mathcal{R}^d_{\text{circ}}$ in grey.

(f) Game $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao},d})$, cf. Fig. 21c.

Figure 22: Reductions for the hybrid argument for Lemma 5.

Corollary 2. *Let $\mathcal{R}_{hyb}$ be the reduction that samples $i \leftarrow\!\!\$ \{1,..,d\}$ and then executes $\mathcal{R}^i_{circ} \to \mathcal{R}_{layer,i} \to \mathcal{R}_{2cpa}$. Then, for all PPT adversaries $\mathcal{A}$*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{SEC}^0(\mathtt{GB}_{yao}), \mathtt{SEC}^1(\mathtt{SIM}_{yao}))\\ &\leq n \cdot d \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{hyb}; \mathtt{IND\text{-}CPA}^0(se), \mathtt{IND\text{-}CPA}^1(se))\end{aligned}$$

Proof. Let $\mathcal{A}$ be an adversary. Denote $\mathcal{R}^i_{\text{hyb}} := \mathcal{R}^i_{\text{circ}} \to \mathcal{R}_{\text{layer},i} \to \mathcal{R}_{\text{2cpa}}$ and note that the probability that $\mathcal{R}_{\text{hyb}} = \mathcal{R}^i_{\text{hyb}}$ is $\frac{1}{d}$, and hence,

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}}), \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}}))\\ \overset{Lem.\ 5}{\leq}& \sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{\text{circ}}; \mathtt{LSEC}^0(\mathtt{GB}_{\text{yao,i}}), \mathtt{LSEC}^1(\mathtt{SIM}_{\text{yao,i}}))\\ \overset{Lem.\ 4}{\leq}& \sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{\text{circ}} \to \mathcal{R}_{\text{layer},i}; \mathtt{2CPA}^b_{1..n}(se))\\ \overset{Cor.\ 1}{\leq}& n \cdot \sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{\text{hyb}}; \mathtt{IND\text{-}CPA}^b(se))\\ =& n \cdot d \cdot \sum_{i=1}^{d} \frac{1}{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^i_{\text{hyb}}; \mathtt{IND\text{-}CPA}^b(se))\\ =& n \cdot d \sum_{i=1}^{d} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\text{hyb}}; \mathtt{IND\text{-}CPA}^b(se))\end{aligned}$$

□

VIII. Alignment with $\mathtt{PRVSIM}^b$

To conclude our proof of Theorem 1, it remains to reduce selective security of Yao's garbling scheme to its circuit garbling security that we established in the previous section. Towards this goal, we define a reduction package MOD in Fig. 23 that provides a GARBLE oracle and queries the oracles of the circuit security games, and apply the perfect reduction lemma (Lemma 1) one last time.

```
Oracle of MOD
GARBLE$(C, x)$
assert $\tilde{C} = \bot$
assert depth$(C) = d$
for $j = 1..n$ do
    SETBIT$_j(x_j)$
    $\tilde{x}[j] \leftarrow$ GETA$^{\text{out}}_j$
for $i = 1..d$ do
    $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
    $\tilde{C}[i] \leftarrow$ GBL$_i(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op})$
for $j = 1..n$ do
    dinf$[j] \leftarrow$ GETKEYS$^{\text{in}}_j$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

Figure 23: Code of MOD.

Proof of Theorem 1. Consider simulator $\mathtt{SIM}_{\text{tdyao}}$ which we have already seen in Fig. 15. To be able to apply Lemma 1 with reduction MOD, we need to show the following two claims:

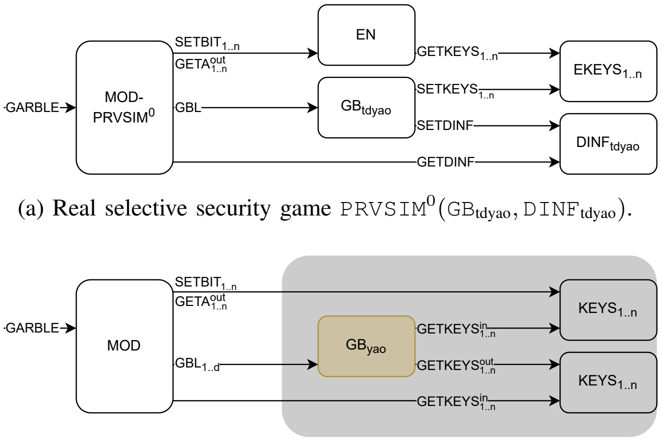
Claim 3 (Real game equivalence)**.**

$$\mathtt{PRVSIM}^0(\mathtt{GB}_{tdyao}, \mathtt{DINF}_{tdyao}) \overset{code}{\equiv} \mathtt{MOD} \to \mathtt{SEC}^0(\mathtt{GB}_{yao})$$

Fig. 24 shows the two games. The claim follows directly after inlining all packages, see Appendix C for details.

Claim 4 (Ideal game equivalence)**.**

$$\mathtt{PRVSIM}^1(\mathtt{SIM}_{tdyao}) \overset{code}{\equiv} \mathtt{MOD} \to \mathtt{SEC}^1(\mathtt{SIM}_{yao})$$



(a) Real selective security game $\mathtt{PRVSIM}^0(\mathtt{GB}_{\text{tdyao}}, \mathtt{DINF}_{\text{tdyao}})$.

(b) $\mathtt{MOD} \to \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$, the game $\mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$ is marked in grey.

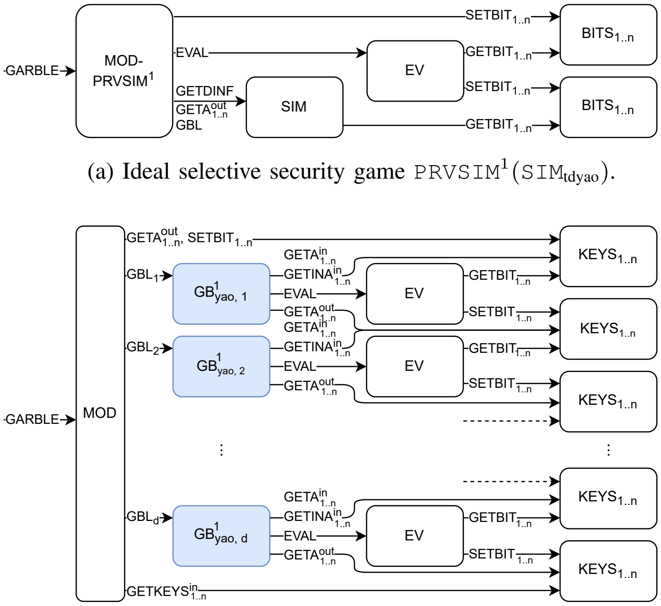
Figure 24: Real security games $\mathtt{PRVSIM}^0(\mathtt{GB}_{\text{tdyao}}, \mathtt{DINF}_{\text{tdyao}})$ and $\mathtt{MOD} \to \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}})$.

The two games are shown in Fig. 25. The claim follows directly after inlining all packages. For details, see Appendix D.

Applying Lemma 1 with Claims 3 and 4 as well as Corollary 2 guarantees now the existence of PPT reductions $\mathcal{B} := \mathcal{A} \to \mathtt{MOD}$ and $\mathcal{R}$ such that

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{PRVSIM}^0(\mathtt{GB}_{\text{tdyao}}, \mathtt{DINF}_{\text{tdyao}}), \mathtt{PRVSIM}^1(\mathtt{SIM}_{\text{tdyao}}))\\ \overset{(3),(4)}{=}\ & \mathsf{Adv}(\mathcal{B}; \mathtt{SEC}^0(\mathtt{GB}_{\text{yao}}), \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}}))\\ \leq\ & d \cdot n \cdot \mathsf{Adv}(\mathcal{B} \to \mathcal{R}; \mathtt{IND\text{-}CPA}^0(se), \mathtt{IND\text{-}CPA}^1(se))\end{aligned}$$

which concludes our proof. □



(a) Ideal selective security game $\mathtt{PRVSIM}^1(\mathtt{SIM}_{\text{tdyao}})$.

(b) $\mathtt{MOD} \to \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$, game $\mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$ is marked in grey.

Figure 25: Ideal security games $\mathtt{PRVSIM}^1(\mathtt{SIM}_{\text{tdyao}})$ and $\mathtt{MOD} \to \mathtt{SEC}^1(\mathtt{SIM}_{\text{yao}})$

IX. Discussion

We now revisit our new security proof for Yao's garbling scheme to discuss our insights and compare our proof to existings proofs in more detail.

A. Definitions and proof style

We reduce BHR security of Yao's garbling scheme to IND-CPA security of the underlying encryption scheme. The main focus of this work is the alternative representation of Yao's garbling *scheme* in the style of state-separating proofs (SSPs) as well as an alternative representation of the *security* of Yao's garbling scheme, also in the style of SSPs. In SSP style, both the scheme and the security notion are described as packages that call each other and otherwise have strictly separated state.

1) Syntactic and local reasoning: Our proof relies almost entirely on *graph-based* reductions (e.g., Fig. 22). Whenever such syntactic reasoning was not possible, we proved code equivalence for suitable subgames, and applied the perfect reduction lemma (Lemma 1) to lift the equivalence result to the more complex games. Composing these subgames with all reduction layers, e.g. $\mathtt{MOD} \to \mathcal{R}_{\mathsf{hyb}} \to \mathtt{2CPA}^b(se)$, would then yield the typical proof presentation as sequence of direct game hops between $\mathtt{PRVSIM}^0(\mathtt{GB}_{\mathsf{tdyao}}, \mathtt{DINF}_{\mathsf{tdyao}})$ and $\mathtt{PRVSIM}^1(\mathtt{SIM}_{\mathsf{tdyao}})$ of a more "traditional" proof. Our proof reduces the number of code equivalence steps to a minimum: alignment of 2-key CPA with standard IND-CPA (Lemma 2) and of circuit garbling security with BHR's selective security (Claims 3 and 4), and the wire key semantic switch (Claim 2) when reasoning about the security of layer garbling.

2) Treating wire keys as their own unit: Wire keys are a central concept in Yao's garbling scheme, and each key is used at least twice: Once as message when garbling a gate, and (at least) once by the layer which uses them as encryption keys. The garbling scheme moreover has the property of being *output projective*, i.e. output decoding is the exact inverse of input decoding. As a result, we can treat input encoding and output decoding information as well as intermediate wire keys uniformly. The SSP focus on *state* rather than *algorithms* thus led us to place special emphasis on the modeling of keys: We use a separate KEYS package which samples and stores keys together with additional information. This uniformity provides the flexibility to interpret such a package as representing input encoding, the generation of output decoding information, or intermediate wire keys during our proof, and hence allows the self-composability of our layer security notion (used in the proof of our hybrid argument, cf. Lemma 5). Interestingly, we do not remove this additional information from KEYS again until the relation with standard garbling scheme security, and instead simply restrict the simulator's access to it.

Implementations typically sample all wire keys in the beginning, analogous to $\mathtt{GB}_{\mathsf{tdyao}}$ (Fig. 15), and before garbling the actual circuit. Our game $\mathtt{SEC}^0(\mathtt{GB}_{\mathsf{yao}})$ could also adopt that style by having a special INITSAMPLE query, but there is no benefit to the proof in having such a query, and since our focus is on the proof, our model of Yao's garbling scheme samples keys only at the point when they are needed.

3) Layer garbling security notion: As mentioned above, modeling wire keys as KEYS package is useful to define a security notion for layer garbling (and thus implicitly gate garbling) in Section VI. Defining this notion allows us to perform the main reduction argument to IND-CPA security locally, at the level of a single circuit layer. Returning to the discussion in Section I, the self-composability of layer garbling security then means security of garbling circuit layers C_1 and C_2 separately implies security of garbling the combined circuit consisting of C_1 and C_2.

4) Circuit garbling security notion: By self-composition, our layer garbling security notion induces a (Yao-specific) circuit garbling security notion (cf. Section VII). We find this security notion of independent interest since it expresses the strong particular security properties of Yao's garbling scheme: It says that a garbling scheme is secure if for any circuit C and input x, garbling can be simulated given only C and an *encoding* of the output $C(x)$ rather than $C(x)$ itself.

On the technical level, the circuit garbling security notion defined by game pair $(\mathtt{SEC}^0(\mathtt{GB}_{\mathsf{yao}}), \mathtt{SEC}^1(\mathtt{SIM}_{\mathsf{yao}}))$ differs from the selective security notion $(\mathtt{PRVSIM}^0(\mathtt{GB}_{\mathsf{tdyao}}, \mathtt{DINF}_{\mathsf{tdyao}}), \mathtt{PRVSIM}^1(\mathtt{SIM}_{\mathsf{tdyao}}))$ in further conceptually interesting ways. For the *real game*, the changes are as follows:

Merging input encoding & input key packages into KEYS to unify the treatment of all wire keys and allow the composition of real and ideal layer garbling security games, which enables the hybrid argument in Section VII.

Factoring out key sampling into KEYS since all wire keys throughout the garbling scheme are treated uniformly.

Splitting garbling interface The selective security game interface is split into separate oracles for choosing an input, obtaining an input encoding, garbling individual circuit layers, and obtaining output decoding information.

Sampling input wire keys before output wire keys A different query order is enforced by MOD to ensure the correct information flow needed for self-composition of layer garbling: Inputs are garbled before the circuit, and output decoding information is only available after that.

The two *ideal games* differ further as follows:

Merging inputs and input wire keys Circuit evaluation is performed directly on the information stored in KEYS, without separate BITS packages.

Simulation based on output *encoding* Since the garbling interface is split, we can easily provide simulator $\mathtt{SIM}_{\mathsf{yao}}$ with access to the active output *keys* instead of active output bits.

B. Comparison with existing proofs

Our proof is the first proof of Yao's garbling scheme to use the state-separating proofs technique. In addition, we rely on different intermediate assumptions that are expressed as local security notions with local simulators, both of which we discuss below.

1) Encryption security and hybrid strategy: All security proofs follow a pattern: Garbling scheme security is first reduced to an intermediate security notion capturing some aspect of gate garbling and encryption security, which is in turn reduced to a standard assumption such as IND-CPA security. Where the proofs differ conceptually is in the intermediate assumptions which in turn impact the details of their hybrids.

The first security proof of Yao's garbling scheme by Lindell and Pinkas [LP09] proposes to abstract the garbling of a gate as *double encryption* security. An adversary inputs two message tuples and is provided with double encryptions, computed like when garbling a gate, as well as encryption oracles corresponding to the inactive input wire keys of the gate. The adversary is asked to distinguish encryptions of left from encryptions of right messages. The hybrid argument ranges then over all gates in the circuit.

BHR [BHR12b] shift the focus from the encryptions associated with garbling a gate to (double) encryptions using a specific wire key. In their *dual-key cipher* assumption, an adversary is given access to an encryption oracle for a dual-key cipher that encrypts under two keys, the challenge key and another key that is chosen by the adversary. The adversary is asked to distinguish real encryptions from encryptions of random strings. The hybrid argument ranges then over all wires in the circuit.

In our proof, we wanted to capture the best of both worlds: Focusing on *gate* and layer security on the one hand allows to stay close to the inherent modularity that real circuit garbling has. Moreover, a core argument in the proof is the semantic switch from 0/1 keys to active/inactive keys (Claim 2). The latter is an argument about the specific way encryption is used and not about the encryption scheme itself. On the other hand, we want to be able to relate the security of encryption under (inactive) *wire keys* to IND-CPA security. The result is a two-step approach: First we introduce a layer security notion that *garbles a circuit layer* and show security of garbling a circuit via a sequence of hybrids ranging over all *layers* in the circuit (Lemma 5). The use of a layer assumption can be seen as close in spirit to Lindell and Pinkas. Layer security is then further reduced to an encryption assumption with two keys (2-key CPA). In this assumption, each encryption is only under one of the keys, thus capturing the contribution of one wire to the double encryption when garbling a gate, which is reminiscent of the intention of BHR's dual-key cipher. The application of the BDFKK multi-instance lemma to obtain Corollary 1 then implicitly contains another sequence of hybrids iterating over all keys in the circuit layer, even though there are no explicit gates or circuit wires at this point.

2) Local security and local simulation: We construct our circuit garbling security notion as composition of layer garbling security games, and hence a circuit simulator can be a composition of *local* layer simulators. Ananth and Lombardi [AL18] recently defined a *local simulation* property for garbling schemes. This property can be seen as a subcircuit (e.g. layer) garbling security notion that maintains some of the state for garbling the rest of the circuit, e.g. wire keys for the whole circuit. As a consequence, their local simulators can be composed to obtain a circuit simulator which they do to construct adaptively secure garbling schemes and garbling schemes for Turing machines. Ananth and Lombardi outline why Yao's garbling scheme, when restricted to layered circuits, has the local simulation property. The argument is derived from the work of Hemenway et al. [HJO$^+$16] on adaptive security of a modification of Yao's garbling scheme, which ultimately follows the proof outline of Lindell and Pinkas [LP09]. In particular, Ananth and Lombardi do not derive a security proof of Yao's garbling scheme from the local simulation property. Their result can thus be interpreted as extracting a layer security property from the Lindell and Pinkas proof, though it differs from ours in that it cannot be composed directly to yield circuit garbling security.

C. State-separating proofs

1) Impact on our proof: Following SSP ideas impacted our proof style and as a result also the proof size. One of the salient benefits of SSPs is the ability to reason syntactically about game equivalences via graph-based reductions, cf. the perfect reduction lemma (Lemma 1). To use this feature, we developed a new modular description of Yao's garbling scheme that expresses the construction as a graph of package dependencies. The packages and their interplay are carefully chosen to simplify the presentation of all subsequent arguments. We strove to state every argument on the smallest subgame (and hence subgraph) possible and reconnect with the large game through the perfect reduction lemma. This is the most visible in our approach to reducing selective security to our intermediate encryption scheme security notion 2-key IND-CPA. While existing proofs perform the equivalent proof step as one big reduction, we break it down into multiple parts: It suffices to reduce layer garbling security to encryption security (Lemma 4), then circuit garbling security to layer garbling security (Lemma 5), until we can finally reduce selective security to circuit garbling security to obtain Thm. 1. The concept of packages was particularly beneficial to express the state sharing between gates in the form of wire keys which ultimately made it possible to split the reduction to encryption scheme security. The split reduces the complexity of each step as well as the size of the game to reason about at a time, which hopefully makes it easier to verify the individual steps.

2) Adapting SSP: We adapted several existing SSP strategies for our setting. When composing a package with itself in parallel, BDFKK add indices to its name and oracles. To lighten notation, we omit the index when package and oracles are uniquely identified by the call graph. `KEY` packages that store key material are another standard concept in SSP, introduced by BDFKK for sharing state between different protocols/primitives, e.g., a KEM and a DEM or a key exchange protocol and a secure channel. We adapt the concepts for keys shared between different layers of the garbling scheme by adding bit semantic or active/inactive semantic of the keys to our `KEYS` packages. Finally, our work is the first to use *simulators* in an SSP context. We find games parameterized by simulators convenient to work with as they do not require to transform the real game precisely into the ideal game. Rather, the ideal game contains a spot for the simulator and it suffices to construct a simulator which can fill it out, typically as whichever code emerges after a sufficient number of game transformations. In follow-up work, Brzuska, Delignat-Lavaud, Egger, Fournet, Kohbrok and Kohlweiss [BDE+21] also used simulation-based security for their definition of the TLS 1.3 key schedule, a core component of the TLS 1.3 handshake protocol.

D. Formal verification

Security of Yao's garbling scheme has been formally verified: Li and Micciancio [LM18] provide a symbolic analysis with computational soundness, a result that is incomparable with our game hopping-style proof. Almeida, Barbosa, Barthe, Dupressoir, Grégoire, Laporte, and Pereira [ABB+17] mechanized the BHR proof for Yao's garbling scheme in EasyCrypt [BGHZ11], a proof assistant for code-based game-playing proofs. Our structured code-based proof can be seen as another target for mechanization. We remark that recent works on mechanizing SSP-style proofs (SSProve [AHR+21], Dupressoir, Kohbrok and Oechsner [DKO21]) as well as discussions with the authors of [ABB+17] give us hope that our proof can indeed be mechanized. However, formal verification–whether in existing general-purpose or SSP-specific tools like SSProve–is a goal that is orthogonal to the scope of this work, and we leave this question as future work.

Acknowledgments

We are grateful to Pihla Karanko, Markulf Kohlweiss, Kirthivaasan Puniamurthy, Luisa Zeppelin, and the participants of the *Advanced Topics in Cryptography* course 2021 at Aalto University for useful suggestions on the presentation. We thank François Dupressoir for insightful discussions about the EasyCrypt security proof of Yao's garbled circuits in [ABB+17]. Finally, we thank the reviewers of CSF 2022 for their thorough comments and helpful suggestions that significantly improved the article. All remaining shortcomings of this article are, of course, our responsibility.

Sabine Oechsner was supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO), the Concordium Blockhain Research Center, Aarhus University, Denmark, and the Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE). This work was supported by the Blockchain Technology Laboratory at the University of Edinburgh and funded by Input Output Global and the Academy of Finland.

References

[ABB+17] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. A fast and verified software stack for secure function evaluation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1989–2006. ACM Press, October / November 2017.

[AHR+21] Carmine Abate, Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Catalin Hritcu, Kenji Maillard, and Bas Spitters. Ssprove: A foundational framework for modular cryptographic proofs in coq. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–15. IEEE, 2021.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Heidelberg, November 2018.

[BDE+21] Chris Brzuska, Antoine Delignat-Lavaud, Christoph Egger, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Key-schedule security for the TLS 1.3 standard. Cryptology ePrint Archive, Report 2021/467, 2021. https://eprint.iacr.org/2021/467.

[BDF+18] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249. Springer, Heidelberg, December 2018.

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, Heidelberg, August 2011.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[DKO21] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing state-separating proofs to easycrypt - A security proof for cryptobox. *IACR Cryptol. ePrint Arch.*, page 326, 2021.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 433–458. Springer, Heidelberg, October / November 2016.

[LM18] Baiyu Li and Daniele Micciancio. Symbolic security of garbled circuits. In Steve Chong and Stephanie Delaune, editors, *CSF 2018Computer Security Foundations Symposium*, pages 147–161. IEEE Computer Society Press, 2018.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[Mau12] Ueli Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *Theory of Security and Applications - Joint Workshop, TOSCA 2011, Saarbrücken, Germany, March 31 - April 1, 2011, Revised Selected Papers*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2012.

[Weg87] Ingo Wegener. *The complexity of Boolean functions*. BG Teubner, 1987.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

Appendix

A. Standard IND-CPA security implies 2-CPA security

Proof of Lemma 2, continued. The proof proceeds via an inlining argument, shown in Fig. 26. Starting from game $\texttt{2CPA}^b(se)$ $(b \in \{0,1\})$ in the leftmost column, we gradually transform the code until we reach game $\texttt{RED} \rightarrow \texttt{IND-CPA}^b(se)$ in the rightmost column. Inlining the oracles of $\texttt{KEYS}$ yields the second column from the first column. To get to the third column, the two versions of the ENC oracle are combined. Isolating the uncorrupted key $Z(1-z)$ and renaming it to k yields the forth column. Finally, factor out $\texttt{IND-CPA}^b(se)$ to obtain game $\texttt{RED} \rightarrow \texttt{IND-CPA}^b(se)$ in the last column. □

B. Proof of Claim 2

Proof of Claim 2, continued. It remains to prove Equation 5: $\texttt{GGATE} \overset{\text{code}}{\equiv} \texttt{GGATE}_{\text{sim}}$. The equivalence follows from an inlining argument shown in Fig. 27. The second column is obtained from the first by inlining $\texttt{ENC}^1_{1..n}$ and rearranging code, and similarly for the third column and inlining $\texttt{KEYS}$. The next step is where the wire key semantic is switched: To garble a gate, oracle GBLG in the third column loops over all combinations of bit pairs (b_ℓ, b_r). To get to the forth column, we apply the bijection $(b_\ell, b_r) \mapsto (b_\ell \oplus z^{\text{in}}_\ell, b_r \oplus z^{\text{in}}_r)$ that maps bit values to active/inactive status to their corresponding, as indicated by z^{in}_ℓ and z^{in}_r. Further rearranging the code yields the fifth column, and factoring out $\texttt{EV}$ and $\texttt{KEYS}$ yields the last column. □

C. Proof of Claim 3

Proof of Claim 3. The claim follows from an inlining argument shown in Fig. 28. Starting from $\texttt{G}^1_{\text{real}} := MPRVSIM^0(\texttt{GB}_{\text{tdyao}}, \texttt{DINF}_{\text{tdyao}})$, we first inline packages $\texttt{EN}_{1..n}$ and $\texttt{GB}_{\text{tdyao}}$ to obtain $\texttt{G}^2_{\text{real}}$. Further inlining $\texttt{EKEYS}_{1..n}$ and $\texttt{DINF}_{\text{tdyao}}$ yields game $\texttt{G}^3_{\text{real}}$. To obtain $\texttt{G}^4_{\text{real}}$, we move the assertions about circuit size and split the wire key sampling. Input wire key sampling remains while all other wire keys are sampled on demand inside the encryption loop. Factoring out $\texttt{KEYS}$ and $\texttt{GATE}$ packages and further $\texttt{MODGB}_i$ yields games $\texttt{G}^5_{\text{real}}$ and $\texttt{G}^6_{\text{real}} := \texttt{MOD} \rightarrow \texttt{SEC}^0(\texttt{GB}_{\text{yao}})$. □

D. Proof of Claim 4

Proof of Claim 4. The claim follows from an inlining argument that is shown in Fig. 29 and 30. Starting from game $\texttt{G}^1_{\text{ideal}} := \texttt{PRVSIM}^1(\texttt{SIM}_{\text{tdyao}})$ in the first column, we first inline packages $\texttt{CEV}$, $\texttt{BITS}$, $\texttt{GB}_{\text{tdyao}}$ and $\texttt{DINF}_{\text{tdyao}}$ to obtain $\texttt{G}^2_{\text{ideal}}$. Game $\texttt{G}^3_{\text{ideal}}$ is the result of moving the input garbling code into the loop where the input wire keys are sampled. Combining the two loops ranging of j yields game $\texttt{G}^4_{\text{ideal}}$. Game $\texttt{G}^5_{\text{ideal}}$ introduces $Z_{i,j}$ in addition of $S_{i,j}$ as well as a more complicated way of computing $S_{i,j}$ to highlight the computation of $S_{i,j}(0)$ and $S_{i,j}(1)$ as active and inactive wire keys. Finally, we factor out $\texttt{KEYS}$ (game $\texttt{G}^5_{\text{ideal}}$), $\texttt{LEV}$ (game $\texttt{G}^6_{\text{ideal}}$) and $\texttt{GB}_{\text{yao}}$ (game $\texttt{G}^7_{\text{ideal}}$) to obtain $\texttt{G}^8_{\text{ideal}} := \texttt{MOD} \rightarrow \texttt{SEC}^1(\texttt{SIM}_{\text{yao}})$. □

```
Oracle of $\mathtt{ENC}^b$

$\mathsf{ENC}(d, m_0, m_1)$
$Z^{\mathsf{in}} \leftarrow \mathsf{GETKEYS}^{\mathsf{in}}()$
$z^{\mathsf{in}} \leftarrow \mathsf{GETBIT}()$
assert $|m_0| = |m_1|$
if $z^{\mathsf{in}} \neq d$ then
    $c \leftarrow\!\$\ enc(Z^{\mathsf{in}}(d), m_b)$
if $z^{\mathsf{in}} = d$ then
    $c \leftarrow\!\$\ enc(Z^{\mathsf{in}}(d), m_0)$
return $c$

Oracles of $\mathtt{KEYS}$

$\mathsf{SETBIT}(z)$
assert $z = \bot$
$z \leftarrow z$
return ()

$\mathsf{GETA}^{\mathsf{out}}()$
assert $z \neq \bot$
aflag $\leftarrow 1$
if $Z = \bot$ then
    $Z(0) \leftarrow\!\$\ \{0,1\}^\lambda$
    $Z(1) \leftarrow\!\$\ \{0,1\}^\lambda$
return $Z(z)$

$\mathsf{GETBIT}()$
assert $z \neq \bot$
return $z$

$\mathsf{GETKEYS}^{\mathsf{in}}()$
assert aflag
    $\vee$ bflag
return $Z$
```

```
$\mathtt{2CPA}^b(se)$

$\mathsf{ENC}(d, m_0, m_1)$
assert aflag
assert $z \neq \bot$
assert $|m_0| = |m_1|$
if $z \neq d$ then
    $c \leftarrow\!\$\ enc(Z(d), m_b)$
if $z = d$ then
    $c \leftarrow\!\$\ enc(Z(d), m_0)$
return $c$

$\mathsf{SETBIT}(z)$
assert $z = \bot$
$z \leftarrow z$
return ()

$\mathsf{GETA}^{\mathsf{out}}()$
assert $z \neq \bot$
aflag $\leftarrow 1$
if $Z = \bot$ then
    $Z(0) \leftarrow\!\$\ \{0,1\}^\lambda$
    $Z(1) \leftarrow\!\$\ \{0,1\}^\lambda$
return $Z(z)$
```

```
$\mathtt{2CPA}^b(se)$

$\mathsf{ENC}(d, m_0, m_1)$
assert aflag
assert $|m_0| = |m_1|$
if $z \neq d$ then
    $c \leftarrow\!\$\ enc(Z(d), m_b)$
if $z = d$ then
    $c \leftarrow\!\$\ enc(Z(d), m_0)$
return $c$

$\mathsf{SETBIT}(z)$
assert $z = \bot$
$z \leftarrow z$
return ()

$\mathsf{GETA}^{\mathsf{out}}()$
assert $z \neq \bot$
aflag $\leftarrow 1$
if $Z = \bot$ then
    $Z(0) \leftarrow\!\$\ \{0,1\}^\lambda$
    $Z(1) \leftarrow\!\$\ \{0,1\}^\lambda$
return $Z()$
```

```
$\mathtt{2CPA}^b(se)$

$\mathsf{ENC}(d, m_0, m_1)$
assert aflag
assert $|m_0| = |m_1|$
if $z \neq d$ then
    assert $k \neq \bot$
    $c \leftarrow\!\$\ enc(k, m_b)$
if $z = d$ then
    $c \leftarrow\!\$\ enc(Z(z), m_0)$
return $c$

$\mathsf{SETBIT}(z)$
assert $z = \bot$
$z \leftarrow z$
return ()

$\mathsf{GETA}^{\mathsf{out}}()$
assert $z \neq \bot$
aflag $\leftarrow 1$
if $Z = \bot$ then
    $Z(z) \leftarrow\!\$\ \{0,1\}^\lambda$
    assert $k = \bot$
    $k \leftarrow\!\$\ \{0,1\}^\lambda$
return $Z(z)$
```

```
$\mathtt{RED}$

$\mathsf{ENC}(d, m_0, m_1)$
assert aflag
assert $|m_0| = |m_1|$
if $z \neq d$ then
    $c \leftarrow \mathsf{ENC}(m_0, m_1)$
if $z = d$ then
    $c \leftarrow\!\$\ enc(k_a, m_0)$
return $c$

$\mathsf{SETBIT}(z)$
assert $z = \bot$
$z \leftarrow z$
return ()

$\mathsf{GETA}^{\mathsf{out}}()$
assert $z \neq \bot$
aflag $\leftarrow 1$
if $Z = \bot$ then
    $k_a \leftarrow\!\$\ \{0,1\}^\lambda$
    $\mathsf{SMP}()$
return $Z(z)$

Oracles of $\mathtt{IND\text{-}CPA}(se)^b$

$\mathsf{SMP}()$
assert $k = \bot$
$k \leftarrow\!\$\ \{0,1\}^\lambda$
return $k$

$\mathsf{ENC}(m_0, m_1)$
assert $k \neq \bot$
assert $|m_0| = |m_1|$
$c \leftarrow\!\$\ enc(k, m_b)$
return $c$
```

Figure 26: Proof of Lemma 2: $\mathtt{2CPA}^b(se) \overset{\text{code}}{\equiv} \mathtt{RED} \to \mathtt{IND\text{-}CPA}^b(se)$

```
Oracles of $\texttt{GGATE}$

$\mathsf{SETBIT}_i(z)$
return $\mathsf{SETBIT}_i(z)$

$\mathsf{GETA}^{\text{out}}_i$
return $\mathsf{GETA}^{\text{out}}_i$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
$Z^{\text{out}}_j \leftarrow \mathsf{GETKEYS}^{\text{out}}_j$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
    $b_j \leftarrow op(b_\ell, b_r)$
    $k_j \leftarrow Z^{\text{out}}_j(b_j)$
    $c^0_{\mathsf{in}} \leftarrow \mathsf{ENC}_\ell(b_\ell, k_j, 0^\lambda)$
    $c^1_{\mathsf{in}} \leftarrow \mathsf{ENC}_\ell(b_\ell, 0^\lambda, 0^\lambda)$
    $c \leftarrow\$ \mathsf{ENC}_r(b_r, c^1_{\mathsf{in}}, c^1_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
return $\mathsf{GETKEYS}^{\text{in}}_j$
```

```
Oracles of $\texttt{GGATE}$

$\mathsf{SETBIT}_i(z)$
return ()

$\mathsf{GETA}^{\text{out}}_i$
return $\mathsf{GETA}^{\text{out}}_i$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
$z^{\text{in}}_\ell \leftarrow \mathsf{GETBIT}(\ell)$
$z^{\text{in}}_r \leftarrow \mathsf{GETBIT}(r)$
$Z^{\text{out}}_j \leftarrow \mathsf{GETKEYS}^{\text{out}}_j$
$Z^{\text{in}}_\ell \leftarrow \mathsf{GETKEYS}^{\text{in}}_\ell$
$Z^{\text{in}}_r \leftarrow \mathsf{GETKEYS}^{\text{in}}_r$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
    $b_j \leftarrow op(b_\ell, b_r)$
    $k^{\text{out}}_j \leftarrow Z^{\text{out}}_j(b_j)$
    $k^{\text{in}}_\ell \leftarrow Z^{\text{in}}_\ell(b_\ell)$
    if $z^{\text{in}}_\ell = b_\ell$ :
        $c^0_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_\ell, k^{\text{out}}_j)$
    if $z^{\text{in}}_\ell \neq b_\ell$ :
        $c^0_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_\ell, 0^\lambda)$
    $c^1_{\mathsf{in}} \leftarrow\$ enc(k_\ell, 0^\lambda)$
    $k^{\text{in}}_r \leftarrow Z^{\text{in}}_r(b_r)$
    if $z^{\text{in}}_r = b_r$ :
        $c \leftarrow\$ enc(k^{\text{in}}_r, c^0_{\mathsf{in}})$
    if $z^{\text{in}}_r \neq b_r$ :
        $c \leftarrow\$ enc(k^{\text{in}}_r, c^1_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
return $\mathsf{GETKEYS}^{\text{in}}_j$
```

```
Oracles of $\texttt{GGATE}$

$\mathsf{SETBIT}_i(z)$
assert $z^{\text{in}}_i = \bot$
$z^{\text{in}}_i \leftarrow z$
return ()

$\mathsf{GETA}^{\text{out}}_i$
assert $z^{\text{in}}_i \neq \bot$
$\texttt{aflag}^{\text{in}}_i \leftarrow 1$
if $Z^{\text{in}}_i = \bot$ :
    $Z^{\text{in}}_i(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{in}}_i(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z^{\text{in}}_i(z^{\text{in}}_i)$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z^{\text{in}}_\ell \neq \bot$
assert $z^{\text{in}}_r \neq \bot$
$\texttt{bflag}^{\text{out}}_j \leftarrow 1$
if $Z^{\text{out}}_j = \bot$ :
    $Z^{\text{out}}_j(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{out}}_j(1) \leftarrow\$ \{0,1\}^\lambda$
assert $\texttt{aflag}^{\text{in}}_\ell = 1$
      $\vee\, \texttt{bflag}^{\text{in}}_\ell = 1$
assert $\texttt{aflag}^{\text{in}}_r = 1$
      $\vee\, \texttt{bflag}^{\text{in}}_r = 1$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
    $k^{\text{in}}_\ell \leftarrow Z^{\text{in}}_\ell(b_\ell)$
    $k^{\text{in}}_r \leftarrow Z^{\text{in}}_r(b_r)$
    if $b_\ell = z^{\text{in}}_\ell \wedge b_r = z^{\text{in}}_r$ :
        $b_j \leftarrow op(b_\ell, b_r)$
        $k^{\text{out}}_j \leftarrow Z^{\text{out}}_j(b_j)$
    else $k^{\text{out}}_j \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_\ell, k^{\text{out}}_j)$
    $c \leftarrow\$ enc(k^{\text{in}}_r, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
assert $\texttt{aflag}^{\text{out}}_j = 1$
      $\vee\, \texttt{bflag}^{\text{out}}_j = 1$
assert $Z^{\text{out}}_j \neq \bot$
return $Z^{\text{out}}_j$
```

```
Oracles of $\texttt{GGATE}$

$\mathsf{SETBIT}_i(z)$
assert $z^{\text{in}}_i = \bot$
$z^{\text{in}}_i \leftarrow z$
return ()

$\mathsf{GETA}^{\text{out}}_i$
assert $z^{\text{in}}_i \neq \bot$
$\texttt{aflag}^{\text{in}}_i \leftarrow 1$
if $Z^{\text{in}}_i = \bot$ :
    $Z^{\text{in}}_i(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{in}}_i(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z^{\text{in}}_i(z^{\text{in}}_i)$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z^{\text{in}}_\ell \neq \bot$
assert $z^{\text{in}}_r \neq \bot$
$\texttt{aflag}^{\text{out}}_j \leftarrow 1$
if $Z^{\text{out}}_j = \bot$ :
    $Z^{\text{out}}_j(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{out}}_j(1) \leftarrow\$ \{0,1\}^\lambda$
assert $\texttt{aflag}^{\text{in}}_r = 1$
assert $\texttt{aflag}^{\text{in}}_\ell = 1$
for $(b_\ell \oplus z^{\text{in}}_\ell, b_r \oplus z^{\text{in}}_r)$
      $\in \{0,1\}^2$ :
    $k^{\text{in}}_\ell \leftarrow Z^{\text{in}}_\ell(b_\ell)$
    $k^{\text{in}}_r \leftarrow Z^{\text{in}}_r(b_r)$
    if $b_\ell \oplus z^{\text{in}}_\ell = b_r \oplus z^{\text{in}}_r = 0$ :
        $b_j \leftarrow op(b_\ell, b_r)$
        $k^{\text{out}}_j \leftarrow Z^{\text{out}}_j(b_j)$
    else $k^{\text{out}}_j \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_\ell, k^{\text{out}}_j)$
    $c \leftarrow\$ enc(k^{\text{in}}_r, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
assert $\texttt{aflag}^{\text{out}}_j = 1$
      $\vee\, \texttt{bflag}^{\text{out}}_j = 1$
assert $Z^{\text{out}}_j \neq \bot$
return $Z^{\text{out}}_j$
```

```
Oracles of $\texttt{GGATE}_{\text{sim}}$

$\mathsf{SETBIT}_i(z)$
assert $z^{\text{in}}_i = \bot$
$z^{\text{in}}_i \leftarrow z$
return ()

$\mathsf{GETA}^{\text{out}}_i$
assert $z^{\text{in}}_i \neq \bot$
$\texttt{aflag}^{\text{in}}_i \leftarrow 1$
if $Z^{\text{in}}_i = \bot$ :
    $Z^{\text{in}}_i(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{in}}_i(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z^{\text{in}}_i(z^{\text{in}}_i)$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z^{\text{in}}_\ell \neq \bot$
assert $z^{\text{in}}_r \neq \bot$
$z^{\text{out}}_j \leftarrow op(z^{\text{in}}_\ell, z^{\text{in}}_r)$
assert $z^{\text{out}}_j \neq \bot$
$\texttt{aflag}^{\text{out}}_j \leftarrow 1$
if $Z^{\text{out}}_j = \bot$ :
    $Z^{\text{out}}_j(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z^{\text{out}}_j(1) \leftarrow\$ \{0,1\}^\lambda$
$S^{\text{out}}_j(0) \leftarrow Z^{\text{out}}_j(z^{\text{in}}_j)$
assert $\texttt{aflag}^{\text{in}}_r = 1$
$S^{\text{in}}_r(0) \leftarrow Z^{\text{in}}_r(z^{\text{in}}_r)$
assert $\texttt{aflag}^{\text{in}}_r = 1$
$S^{\text{in}}_r(1) \leftarrow Z^{\text{in}}_r(1 \oplus z^{\text{in}}_r)$
assert $\texttt{aflag}^{\text{in}}_\ell = 1$
$S^{\text{in}}_\ell(0) \leftarrow Z^{\text{in}}_\ell(z^{\text{in}}_\ell)$
assert $\texttt{aflag}^{\text{in}}_\ell = 1$
$S^{\text{in}}_\ell(1) \leftarrow Z^{\text{in}}_\ell(1 \oplus z^{\text{in}}_\ell)$
for $(d_\ell, d_r) \in \{0,1\}^2$ :
    $k^{\text{in}}_\ell \leftarrow S^{\text{in}}_\ell(d_\ell)$
    $k^{\text{in}}_r \leftarrow S^{\text{in}}_r(d_r)$
    if $d_\ell = d_r = 0$ :
        $k^{\text{out}}_j \leftarrow S^{\text{out}}_j(0)$
    else $k^{\text{out}}_j \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_r, k^{\text{out}}_j)$
    $c \leftarrow\$ enc(k^{\text{in}}_\ell, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
assert $\texttt{aflag}^{\text{out}}_j = 1$
      $\vee\, \texttt{bflag}^{\text{out}}_j = 1$
assert $Z^{\text{out}}_j \neq \bot$
return $Z^{\text{out}}_j$
```

```
Oracles of $\texttt{GGATE}_{\text{sim}}$

$\mathsf{SETBIT}_i(z)$
return $\mathsf{SETBIT}_i(z)$

$\mathsf{GETA}^{\text{out}}_i$
return $\mathsf{GETA}^{\text{out}}_i$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
$\mathsf{EVAL}_j(\ell, r, op)$
$S^{\text{out}}_j(0) \leftarrow \mathsf{GETA}^{\text{out}}_j$
$S^{\text{in}}_r(0) \leftarrow \mathsf{GETA}^{\text{in}}_r$
$S^{\text{in}}_r(1) \leftarrow \mathsf{GETINA}^{\text{in}}_r$
$S^{\text{in}}_\ell(0) \leftarrow \mathsf{GETA}^{\text{in}}_\ell$
$S^{\text{in}}_\ell(1) \leftarrow \mathsf{GETINA}^{\text{in}}_\ell$
for $(d_\ell, d_r) \in \{0,1\}^2$ :
    $k^{\text{in}}_\ell \leftarrow S^{\text{in}}_\ell(d_\ell)$
    $k^{\text{in}}_r \leftarrow S^{\text{in}}_r(d_r)$
    if $d_\ell = d_r = 0$ :
        $k^{\text{out}}_j \leftarrow S^{\text{out}}_j(0)$
    else $k^{\text{out}}_j \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\$ enc(k^{\text{in}}_r, k^{\text{out}}_j)$
    $c \leftarrow\$ enc(k^{\text{in}}_\ell, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}^{\text{in}}_j$
return $\mathsf{GETKEYS}^{\text{in}}_j$
```

Figure 27: Inlining for code-equivalence of $\texttt{GGATE}$ and $\texttt{GGATE}_{\text{sim}}$ in proof of Claim 2.

```
$\mathsf{G}^1_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
for $j = 1..n$ do
  $\mathsf{SETBIT}_j(x_j)$

$\tilde{C} \leftarrow \mathsf{GBL}(C)$

dinf $\leftarrow$ $\mathsf{GETDINF}$
for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\text{out}}_j$

return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^2_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
for $j = 1..n$ do
  $\mathsf{SETBIT}_j(x_j)$

assert $\tilde{C} = \perp$
assert $\mathsf{depth}(C) = d$
for $i = 0..d$ do
  for $j = 1..n$ do
    $Z_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$

  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \perp$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \perp$
    for $(b_\ell, b_r) \in \{0,1\}^2$ do
      $b_j \leftarrow op(b_\ell, b_r)$
      $k_j \leftarrow Z_{i,j}(b_j)$
      $c_{\mathsf{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), k_j)$
      $c \leftarrow\$ enc(Z_{i-1,r}(b_r), c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}[i, j] \leftarrow \tilde{g}_j$
for $j = 1..n$ do
  $\mathsf{SETKEYS}_j(Z_{0,j})$
$\mathsf{SETDINF}(Z_{d,1}, \ldots, Z_{d,n})$
dinf $\leftarrow$ $\mathsf{GETDINF}$
for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\text{out}}_j$

return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^3_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \perp$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_j = \perp$
  $z_j \leftarrow x_j$
for $i = 0..d$ do
  for $j = 1..n$ do
    $Z_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$

  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \perp$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \perp$
    for $(b_\ell, b_r) \in \{0,1\}^2$ do
      $b_j \leftarrow op(b_\ell, b_r)$
      $k_j \leftarrow Z_{i,j}(b_j)$
      $c_{\mathsf{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), k_j)$
      $c \leftarrow\$ enc(Z_{i-1,r}(b_r), c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}[i, j] \leftarrow \tilde{g}_j$

dinf $\leftarrow$ $(Z_{d,1}, \ldots, Z_{d,n})$
for $j = 1..n$ do
  assert $z_j \neq \perp$
  $\tilde{x}[j] \leftarrow Z_{0,j}(z_j)$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^4_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \perp$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \perp$
  $z_{0,j} \leftarrow x_j$
  $Z_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$
  $\tilde{x}[j] \leftarrow Z_{0,j}(z_{0,j})$

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$

  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \perp$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{C}_j \leftarrow \perp$
    $Z_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
    for $(b_\ell, b_r) \in \{0,1\}^2$ :
      $b_j \leftarrow op(b_\ell, b_r)$
      $k^0_j \leftarrow Z_{i,j}(b_j)$
      $c^0_{\mathsf{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), k^0_j)$
      $c^1_{\mathsf{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), 0^\lambda)$
      $c \leftarrow\$ enc(Z_{i-1,r}(b_r), c^0_{\mathsf{in}})$

      $\tilde{C}_j \leftarrow \tilde{C}_j \cup \{c\}$
  $\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $\text{dinf}[j] \leftarrow$ $Z_{d,j}$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^5_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \perp$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_j(x_j)$

  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\text{out}}_j$

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \perp$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \perp$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow$
    $(\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{C}_j \leftarrow \mathsf{GBLG}(\ell, r, op, j)$

  $\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $\text{dinf}[j] \leftarrow \mathsf{GETKEYS}^{\text{in}}_j$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^6_{\text{real}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \perp$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_j(x_j)$

  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\text{out}}_j$

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  $\tilde{C}[i] \leftarrow \mathsf{GBL}_i(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op})$

for $j = 1..n$ do
  $\text{dinf}[j] \leftarrow \mathsf{GETKEYS}^{\text{in}}_j$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

Figure 28: Proof of Claim 3.

```
$\mathsf{G}^1_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_j(x_j)$
$\mathsf{EVAL}_j(C)$
$\tilde{C} \leftarrow \mathsf{GBL}(C)$
$\text{dinf} \leftarrow \mathsf{GETDINF}$
for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow \mathsf{GETA}_j^{\text{out}}$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^2_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $z_{i,j} \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
for $j = 1..n$ do
  $S_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $S_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $S_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $S_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^\lambda$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
  $\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  $\text{dinf}[j] \leftarrow Z_{d,j}$
for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow S_{0,j}(0)$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^3_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $z_{i,j} \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
for $j = 1..n$ do
  $S_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $S_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$
  $\tilde{x}[j] \leftarrow S_{0,j}(0)$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $S_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $S_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^\lambda$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  $\text{dinf}[j] \leftarrow Z_{d,j}$

return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

Figure 29: Proof of Claim 4.

```
$\mathsf{G}^{4}_{\mathsf{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
  $\texttt{aflag}_{0,j} \leftarrow 1$
  $S_{0,j}(0) \leftarrow\$ \{0,1\}^{\lambda}$
  $S_{0,j}(1) \leftarrow\$ \{0,1\}^{\lambda}$
  $\tilde{x}[j] \leftarrow S_{0,j}(0)$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $z_{i,j} \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
    $S_{i,j}(0) \leftarrow\$ \{0,1\}^{\lambda}$
    $S_{i,j}(1) \leftarrow\$ \{0,1\}^{\lambda}$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^{\lambda}$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  $\mathrm{dinf}[j] \leftarrow Z_{d,j}$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

```
$\mathsf{G}^{5}_{\mathsf{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
  $\texttt{aflag}_{0,j} \leftarrow 1$
  $Z_{0,j}(0) \leftarrow\$ \{0,1\}^{\lambda}$
  $Z_{0,j}(1) \leftarrow\$ \{0,1\}^{\lambda}$
  $\tilde{x}[j] \leftarrow Z(z_{0,j})$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $z_{i,j} \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
    $Z_{i,j}(0) \leftarrow\$ \{0,1\}^{\lambda}$
    $Z_{i,j}(1) \leftarrow\$ \{0,1\}^{\lambda}$
    $S_{i,j}(0) \leftarrow Z_{i,j}(z_{i,j})$
    $S_{i-1,r}(0) \leftarrow Z_{i-1,r}(z_{i-1,r})$
    $S_{i-1,r}(1) \leftarrow Z_{i-1,r}(1 - z_{i-1,r})$
    $S_{i-1,\ell}(0) \leftarrow Z_{i-1,r}(z_{i-1,\ell})$
    $S_{i-1,\ell}(1) \leftarrow Z_{i-1,r}(1 - z_{i-1,\ell})$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^{\lambda}$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $\mathrm{dinf}[j] \leftarrow Z_{d,j}$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

```
$\mathsf{G}^{6}_{\mathsf{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_{0,j}(x_j)$
  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\mathsf{out}}_{0,j}$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $z_{i-1,\ell} \leftarrow \mathsf{GETBIT}_{i-1,\ell}$
    $z_{i-1,r} \leftarrow \mathsf{GETBIT}_{i-1,r}$
    $z_{i,j} \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
    $\mathsf{SETBIT}_{i,j}(z_{i,j})$
    $S_{i,j}(0) \leftarrow \mathsf{GETA}^{\mathsf{out}}_{i,j}$
    $S_{i-1,r}(0) \leftarrow \mathsf{GETA}^{\mathsf{in}}_{i-1,r}$
    $S_{i-1,r}(1) \leftarrow \mathsf{GETINA}^{\mathsf{in}}_{i-1,r}$
    $S_{i-1,\ell}(0) \leftarrow \mathsf{GETA}^{\mathsf{in}}_{i-1,\ell}$
    $S_{i-1,\ell}(1) \leftarrow \mathsf{GETINA}^{\mathsf{in}}_{i-1,\ell}$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k^{\mathsf{out}}_{i,j} \leftarrow S_{i,j}(0)$
      else $k^{\mathsf{out}}_{i,j} \leftarrow 0^{\lambda}$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $\mathrm{dinf}[j] \leftarrow \mathsf{GETKEYS}^{\mathsf{in}}_{d,j}$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

```
$\mathsf{G}^{7}_{\mathsf{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_{0,j}(x_j)$
  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\mathsf{out}}_{0,j}$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  assert $\tilde{C}[i] = \bot$
  assert $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op} \neq \bot$
  assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    $\mathsf{EVAL}_{i,j}(\ell, r, op)$
    $S^{\mathsf{out}}_{i,j}(0) \leftarrow \mathsf{GETA}^{\mathsf{out}}_{i,j}$
    $S^{\mathsf{in}}_{i-1,r}(0) \leftarrow \mathsf{GETA}^{\mathsf{in}}_{i-1,r}$
    $S^{\mathsf{in}}_{i-1,r}(1) \leftarrow \mathsf{GETINA}^{\mathsf{in}}_{i-1,r}$
    $S^{\mathsf{in}}_{i-1,\ell}(0) \leftarrow \mathsf{GETA}^{\mathsf{in}}_{i-1,\ell}$
    $S^{\mathsf{in}}_{i-1,\ell}(1) \leftarrow \mathsf{GETINA}^{\mathsf{in}}_{i-1,\ell}$
    for $(d_\ell, d_r) \in \{0,1\}^2$ :
      $k_{i-1,\ell} \leftarrow S^{\mathsf{in}}_{i-1,\ell}(d_\ell)$
      $k_{i-1,r} \leftarrow S^{\mathsf{in}}_{i-1,r}(d_r)$
      if $d_\ell = d_r = 0$ :
        $k_{i,j} \leftarrow S^{\mathsf{out}}_{i,j}(0)$
      else $k_{i,j} \leftarrow 0^{\lambda}$
      $c_{\mathsf{in}} \leftarrow\$ enc(k_{i-1,r}, k_{i-1,j})$
      $c \leftarrow\$ enc(k_{i-1,\ell}, c_{\mathsf{in}})$
      $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
    $\tilde{C}_j \leftarrow \tilde{g}_j$
$\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
for $j = 1..n$ do
  $\mathrm{dinf}[j] \leftarrow \mathsf{GETKEYS}^{\mathsf{in}}_{d,j}$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

```
$\mathsf{G}^{8}_{\mathsf{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
  $\mathsf{SETBIT}_{j}(x_j)$
  $\tilde{x}[j] \leftarrow \mathsf{GETA}^{\mathsf{out}}_{j}$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  $\tilde{C}[i] \leftarrow \mathsf{GBL}_i(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op})$
for $j = 1..n$ do
  $\mathrm{dinf}[j] \leftarrow \mathsf{GETKEYS}^{\mathsf{in}}_{j}$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

Figure 30: Proof of Claim 4, continued.