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 $P_1, \ldots, P_n$ to evaluate a public function f on their secret inputs $x_1, \ldots, x_n$ and learn nothing but the result $y = f(x_1, .., x_n)$. In his seminal work [21], 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 [18]. Bellare, Hoang and Rogaway (BHR) [6] 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 [7].

Recently, Brzuska, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss (BDFKK) [9] 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 circuit structure. 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) [6] introduce the notion of a *garbling scheme* as an abstraction of the primitive underlying the garbled circuits approach.

Definition 1 (Garbling scheme [6])**.** *A* (circuit) garbling scheme *consists of* 5 *probabilistic, polynomial-time algorithms* 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, $\Phi(C)$ will be equal to C in this article.

```
$\texttt{PRVSIM}^0_{\text{gs},\Phi,\mathcal{S}}$
$\mathsf{GARBLE}(C, x)$
$(\tilde{C}, e, \text{dinf}) \leftarrow gb(1^\lambda, C)$
$\tilde{x} \leftarrow en(e, x)$
$\textbf{return } (\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\texttt{PRVSIM}^1_{\text{gs},\Phi,\mathcal{S}}$
$\mathsf{GARBLE}(C, x)$
$y \leftarrow C(x)$
$(\tilde{C}, \tilde{x}, \text{dinf}) \leftarrow \mathcal{S}(1^\lambda, y, \Phi(C))$
$\textbf{return } (\tilde{C}, \tilde{x}, \text{dinf})$
```

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

Definition 2 (Garbling scheme correctness [6])**.** *Let* $\lambda \in \mathbb{N}$. *A garbling scheme* gs $= (gb, en, de, gev)$ *is* perfectly correct *if for all circuits C and inputs x,*

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

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 $\texttt{PRVSIM}^b_{\text{gs},\Phi,\mathcal{S}}$ for $b \in \{0, 1\}$ (Figure 1) and define security as indistinguishability between them.

Definition 3 (Garbling scheme security [6])**.** *Let* $\lambda \in \mathbb{N}$. *A garbling scheme* 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*

$$\big|\Pr\big[1 \leftarrow\$ \mathcal{A} \rightarrow \texttt{PRVSIM}^0_{\text{gs},\Phi,\mathcal{S}}\big] - \Pr\big[1 \leftarrow\$ \mathcal{A} \rightarrow \texttt{PRVSIM}^1_{\text{gs},\Phi,\mathcal{S}}\big]\big|$$

of $\mathcal{A}$ *interacting with* $\texttt{PRVSIM}^0_{\text{gs},\Phi,\mathcal{S}}$ *and* $\texttt{PRVSIM}^1_{\text{gs},\Phi,\mathcal{S}}$ *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* [5] 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. [15] 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. Implications of this choice will be discussed in Sections IX-A and IX-B.

III. State-Separating Proofs and Encryption Scheme Security

Security games such as $\mathtt{PRVSIM}^0_{\mathsf{gs},\Phi,\mathcal{S}}$ described in the previous section are not known to come with a natural way of composition such as Universal Composability [10], [19]. However, Brzuska, Delignat-Lavaud, Fournet, Kohbrok, and Kohlweiss (BDFKK [9]) 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 [7], a game is a set of oracles operating on shared state.

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

```
IND-CPA(se)^b

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

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

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 $\mathtt{IND\text{-}CPA}(se)^b$ 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. We use assertions for error handling

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

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. Depending on b, oracle ENC either returns an encryption of the left ($b = 0$) or the right ($b = 1$) message. The oracles SMP and ENC of $\mathtt{IND\text{-}CPA}(se)^b$ share state k, the encryption key. se is IND-CPA secure if any efficient adversary who interacts with the oracles of the $\mathtt{IND\text{-}CPA}(se)^b$ cannot determine b much better than with guessing probability.

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, from the outside, a package's state can only be accessed through oracle calls.

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

The term *game* refers then to the special case of a package $\mathtt{G}$ which has no dependencies, that is $[\mathtt{G} \rightarrow] = \emptyset$. For example, the games $\mathtt{IND\text{-}CPA}(se)^b$ have $[\rightarrow \mathtt{IND\text{-}CPA}(se)^b] = \{\mathsf{SMP}, \mathsf{ENC}\}$ and $[\mathtt{IND\text{-}CPA}(se)^b \rightarrow] = \emptyset$. 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)**.** *An adversary package* $\mathtt{A}$ *is an* adversary *if* $[\rightarrow \mathtt{A}] = \emptyset$*.*

C. Composition

We can compose two packages $\mathtt{M}$ and $\mathtt{N}$ sequentially along matching oracle names and dependencies into a new package $\mathtt{M} \rightarrow \mathtt{N}$. 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* $\mathtt{G}^0$ *and* $\mathtt{G}^1$ *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* $\mathsf{Adv}(\mathcal{A}; \mathtt{G}^0, \mathtt{G}^1)$*, is defined as*

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

By convention, all packages in a package composition receive the *same* security parameter, given implicitly, which is useful to define probabilistic polynomial-time.

Definition 8 (PPT runtime)**.** *A package* $\mathtt{M}$ *with security parameter* λ *is* probabilistic polynomial-time (PPT) *if the following holds: If for any query* x *to oracles of* $[\mathtt{M} \rightarrow]$ *the length of the answer is bounded by* $p(|x|)$ *for some polynomial* p*, then:*

- *For any query* y *to oracles of* $[\rightarrow \mathtt{M}]$*, the runtime is upper bounded by* $q(\lambda, N, L)$ *for some polynomial* q*, where* N *denotes the overall number of queries to oracles* $\mathtt{M}$ *and* L *the length of the concatenation of the inputs of these queries, encoded in binary, and*
- *for any query* w *to oracles of* $[\rightarrow \mathtt{M}]$*, the size of the answer is upper bounded by* $r(|w|, \lambda)$ *for some polynomial* r*.*

In this PPT definition, the runtime of $\mathtt{M}$ may depend on the size of the inputs by its callers, but not on the size of the outputs of its callees. For adversaries (which do not have a caller), we recover the standard PPT definition. Having clarified advantage and runtime, we now define IND-CPA.

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

We can compose an adversary $\mathtt{A}$ with $[\mathtt{A} \rightarrow] = \{\mathsf{SMP}, \mathsf{ENC}\}$ with the real game $\mathtt{IND\text{-}CPA}(se)^0$. The result $\mathtt{A} \rightarrow \mathtt{IND\text{-}CPA}(se)^0$ (cf. Fig. 3) describes the real execution of the IND-CPA game. The adversary $\mathtt{A}$ can interact with the game through calls to its oracles and eventually terminates by outputting a bit. Next, we turn to a *code-equivalent* modular version of the $\mathtt{IND\text{-}CPA}^0$ games. Two games $\mathtt{G}^0$ and $\mathtt{G}^1$ are *code equivalent*, denoted $\mathtt{G}^0 \overset{\text{code}}{\equiv} \mathtt{G}^1$, if for all adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}(\mathcal{A}; \mathtt{G}^0, \mathtt{G}^1)$ is 0. The modular game consists of a $\mathtt{KEY}$ package for key generation/storage and an $\mathtt{ENCRYPT}^b$ package for encryption. We denote the resulting new modular games as $\mathtt{mIND\text{-}CPA}(se)^b$ (cf. Fig. 4). The $\mathtt{KEY}$ package (cf. Fig. 5) provides oracles SMP for key sampling and GET to retrieve a stored key. Oracle ENC of packages $\mathtt{ENCRYPT}^b$ takes two messages m_0 and m_1, queries the key stored in $\mathtt{KEY}$ and outputs an encryption of m_b. $\mathtt{ENCRYPT}^b$ is stateless while $\mathtt{KEY}$ has key k as state, and the packages only share state via oracle calls. A package $\mathtt{M}$ must not call its own oracles, and the directed package call graphs are *acyclic*. This restriction enforces a functional call style, i.e. after a caller $\mathtt{M}$ calls a callee $\mathtt{N}$, the package $\mathtt{N}$ might make further oracle calls to other packages, but eventually returns control to $\mathtt{M}$. Acyclic call graphs contribute to a meaningful PPT notion.

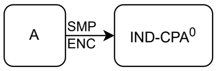


Figure 3: Real $\mathtt{IND\text{-}CPA}^0(se)$ game.

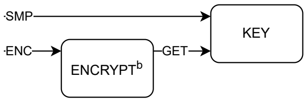


Figure 4: Modular IND-CPA games $\mathtt{mIND\text{-}CPA}(se)^b$.

Notation. $\mathtt{G(M)}$ denotes a composed package $\mathtt{G}$ which is parametrized by package $\mathtt{M}$, i.e. all of $\mathtt{G}$ is fixed except for $\mathtt{M}$. We write $\mathtt{G}(\mathsf{algo})$ for a package (composition) which depends on an algorithm algo, cf. $\mathtt{IND\text{-}CPA}^b(se)$.

```
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 $\mathtt{ENCRYPT}^b$, $b \in \{0, 1\}$, and $\mathtt{KEY}$.

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

D. Reductions

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

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

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

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

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

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

Proof. Associativity of package composition yields:

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

□

The SSP style is particularly useful for finding and expressing perfect reductions. As an example, consider the modified encryption scheme security notion $\mathtt{2CPA}(se)^b$ in Figure 6. In this IND-CPA variant with two encryption keys $Z(0)$ and $Z(1)$, an adversary chooses one of the keys to be corrupt, while security of encryptions under the honest key is still guaranteed. The game consists of packages $\mathtt{KEYS}$ and $\mathtt{ENC}^b$ (cf. Fig. 7, capital letters denote sets and maps: S, $T(x)$.). $\mathtt{KEYS}$ stores both keys and provides oracles SETBIT for choosing which key to corrupt and GETBIT for retrieving this information, $\mathsf{GETA}^{\mathsf{out}}$ for retrieving the adversary key and sampling both keys if they do not exist yet and $\mathsf{GETKEYS}^{\mathsf{in}}$ that returns both keys if they exist. Package $\mathtt{ENC}^b$ provides an encryption oracle that encrypts one of two messages. Importantly, oracle ENC^b retrieves both keys via $\mathsf{GETKEYS}^{\mathsf{in}}$ and computes which to use as encryption key. The real (left) game always encrypts m_0. The ideal (right) game additionally retrieves the bit of the corrupt key via GETBIT and encrypts m_1 if the key is not corrupted. Jumping ahead, $\mathtt{KEYS}$ will model the two keys associated with a circuit wire in projective garbling schemes (cf. Section IV).

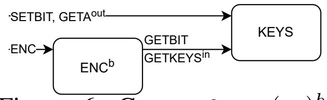


Figure 6: Games $\mathtt{2CPA}(se)^b$. Only $\mathtt{ENC}^1$ calls GETBIT.

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

Lemma 2. *Let se be a symmetric encryption scheme. For reduction* $\mathcal{R}_{cpa} := \mathtt{RED}$ *(cf. Fig. 9), it holds that for any PPT adversary* $\mathcal{A}$,

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

Proof. $\mathtt{RED}$ samples and stores the corrupt key $Z(z)$ and answers all queries related to it, while queries regarding the honest key are forwarded to the IND-CPA game. We prove

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

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

$\mathsf{ENC}(d, m_0, m_1)$
$Z^{\text{in}} \leftarrow \mathsf{GETKEYS}^{\text{in}}()$
assert $|m_0| = |m_1|$
$c \leftarrow\!\$\ enc(Z^{\text{in}}(d), m_0)$
if $b = 1$
  $z^{\text{in}} \leftarrow \mathsf{GETBIT}()$
  if $z^{\text{in}} \neq d$ then
    $c \leftarrow\!\$\ enc(Z^{\text{in}}(d), m_b)$
return $c$
```

```
Oracles of KEYS

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

$\mathsf{GETKEYS}^{\text{in}}()$
assert flag
return $Z$

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

$\mathsf{GETA}^{\text{out}}()$
assert $z \neq \bot$
flag $\leftarrow 1$
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 $\mathtt{ENC}^0$, $\mathtt{ENC}^1$. Further oracles of KEYS will be introduced in Fig. 18. Oracle GETBIT is only called by $\mathtt{ENC}^1$.

Lemma 2 now follows from Lemma 1 by observing that with $\mathsf{G}^b_{big} := \mathtt{2CPA}^b(se)$ and $\mathsf{G}^b_{sml} := \mathtt{IND\text{-}CPA}^b(se)$, (3) corresponds to (1). It thus remains to prove (3). On a high-level, the state of each game $\mathtt{2CPA}^b(se)$ consists of a bit z and two key $Z(0)$, $Z(1)$, all of which are stored in KEYS. In $\mathtt{RED} \rightarrow \mathtt{IND\text{-}CPA}^b(se)$ (shown in Fig. 8), on the other hand, the same state is split between RED and $\mathtt{IND\text{-}CPA}^b(se)$: RED stores z and $Z(z)$, while $\mathtt{IND\text{-}CPA}^b(se)$'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. □

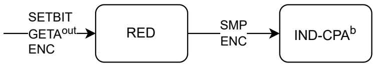


Figure 8: Games $\mathtt{RED} \rightarrow \mathtt{IND\text{-}CPA}^b(se)$.

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, for example to obtain a 2CPA security notion:

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*

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

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

Using the BDFKK multi-instance lemma, we can shows that single-instance security of a game implies multi-instance security of the same game. We here reproduce the lemma for $\mathtt{2CPA}^b(se)$ to obtain its multi-instance version $\mathtt{2CPA}^b_{1..n}(se)$.

```
RED

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

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

$\mathsf{ENC}(d, m_0, m_1)$
assert flag $= 1$
assert $|m_0| = |m_1|$
$c \leftarrow\!\$\ enc(Z(z), m_0)$
if $z \neq d$ then
  $c \leftarrow \mathsf{ENC}(m_0, m_1)$
return $c$
```

Figure 9: Oracles of reduction package RED.

Lemma 3. *([9, Appendix B, Lemma 38]) There exists a PPT reduction $\mathcal{R}_{se}$ such that for all PPT $\mathcal{A}$, we have that*

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

The following corollary combines the results of this section.

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

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

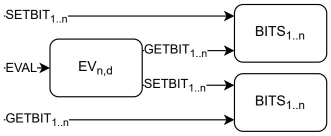
IV. State-Separated Garbling Schemes

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 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 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.

1) Circuit evaluation: We start with an SSP equivalent of circuit evaluation ev, game $\mathtt{CEV}_{n,d}$. The game provides oracles SETBIT for setting input bits, EVAL for evaluating a layered circuit of depth d and width n, 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 $\mathtt{CEV}_{n,d}$ as composition of three packages: two $\mathtt{BITS}_{1..n}$ packages that model the input bits and output bits, respectively, and a circuit evaluation package $\mathtt{EV}_{n,d}$ which performs the actual computation. $\mathtt{BITS}_{1..n}$ is a simple package for storing bits (Fig. 10b). Package $\mathtt{EV}_{n,d}$ on the other hand is stateless and provides an oracle that evaluates an input circuit C on the



(a) Circuit evaluation game $\texttt{CEV}_{n,d}$ with $\texttt{EV}_{n,d}$ and $\texttt{BITS}_{1..n}$.

```
EV_{n,d}
EVAL(C)
assert width(C) = n
assert depth(C) = d
for j = 1..n do
    z_{0,j} ← GETBIT_j
for i = 1..d do
    (ℓ, r, op) ← C[i]
    for j = 1..n do
        (ℓ, r, op) ← (ℓ(j), r(j), op(j))
        z_{i,j} ← op(z_{i-1,ℓ}, z_{i-1,r})
for j = 1..d do
    SETBIT_j(z_{d,j})
return ()

BITS_j
SETBIT_j(z)
assert z_j = ⊥
z_j ← z
return ()

GETBIT_j
assert z_j ≠ ⊥
return z_j
```

(b) Code of packages $\texttt{EV}_{n,d}$ and $\texttt{BITS}_j$.

Figure 10: Graph and code of circuit evaluation game $\texttt{CEV}_{n,d}$.

bits stored in the top $\texttt{BITS}_{1..n}$ package, and stores the result in bottom $\texttt{BITS}_{1..n}$ package.

Definition 11 (Circuit evaluation)**.** *Circuit evaluation game* $\texttt{CEV}_{n,d}$ *for layered Boolean circuits of width* n *and depth* d *is defined in Fig. 10 and has* $[\to \texttt{CEV}_{n,d}]$: $\mathsf{SETBIT}_{1..n}$, EVAL, $\mathsf{GETBIT}_{1..n}$ *and* $[\texttt{CEV}_{n,d} \to] = \emptyset$.

```
EKEYS_j
SETKEYS_j(Z)
assert Z_j = ⊥
Z_j ← Z
return ()

GETKEYS_j
assert Z_j ≠ ⊥
return Z_j

EN_j
SETBIT_j(z)
assert z_j = ⊥
z_j ← z
return ().

GETA_j
assert z_j ≠ ⊥
Z ← GETKEYS^in_j
return Z(z_j)
```

Figure 11: Code of $\texttt{EKEYS}_j$ and $\texttt{EN}_j$.

2) Garbling scheme syntax: We capture the garbling scheme syntax as tuple of packages $\texttt{GB}_{n,d}$, $\texttt{EN}_{1..n}$, $\texttt{EKEYS}$, $\texttt{DE}_n$, $\texttt{DINF}$, $\texttt{GEV}_{n,d}$ and $\texttt{EV}_{n,d}$, corresponding to the BHR algorithms *gb*, *en*, *de*, *gev*, *ev* plus some shared state. Each package captures the algorithm of the same name, except for the two added packages that store shared state between algorithms: $\texttt{EKEYS}$ models input encoding information e as pairs of keys and $\texttt{DINF}$ models output decoding information d. Since we only consider projective garbling schemes, we *fix* packages $\texttt{EKEYS}$ and $\texttt{EN}_{1..n}$ to sample wire keys Z and choose one of them, respectively (cf. Fig. 11). Hence, we define garbling schemes as tuple $(\texttt{GB}_{n,d}, \texttt{DE}_n, \texttt{DINF}, \texttt{GEV}_{n,d})$. The call graph (Fig. 12) composes the packages such that the adversary can use them meaningfully: Garble a circuit and input, then evaluate the garbled circuit on the garbled input and decode the result.

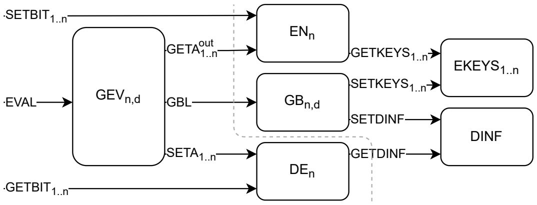


Figure 12: Real garbling scheme correctness game $\texttt{GCORR}(\texttt{GB}_{n,d}, \texttt{DE}_n, \texttt{DINF}, \texttt{GEV}_{n,d})$.

3) Correctness: Garbling scheme $(\texttt{GB}_{n,d}, \texttt{DE}_n,$ $\texttt{DINF},$ $\texttt{GEV}_{n,d})$ is correct if the game $\texttt{GCORR}(\texttt{GB}_{n,d},$ $\texttt{DE}_n,$ $\texttt{DINF},$ $\texttt{GEV}_{n,d})$ (Fig. 12) behaves as $\texttt{CEV}_{n,d}$. Due to decryption ambiguities, a negligible statistical gap might exist.

Definition 12 (Garbling Scheme)**.** *A family* $\mathsf{gs} = \{(\texttt{GB}_{n,d}, \texttt{DE}_n, \texttt{DINF}, \texttt{GEV}_{n,d})\}_{n,d \in \mathbb{N}}$ *of package tuples is a garbling scheme if for all* $n, d \in \mathbb{N}$ *the games* $\texttt{GCORR}(\texttt{GB}_{n,d}, \texttt{DE}_n, \texttt{DINF}, \texttt{GEV}_{n,d})$ *(Fig. 12) and* $\texttt{CEV}_{n,d}$ *(Fig. 10) are statistically indistinguishable, i.e.,* $\mathsf{Adv}(\mathcal{A}; \texttt{GCORR}(\texttt{GB}_{n,d}, \texttt{DE}_n, \texttt{DINF}, \texttt{GEV}_{n,d}), \texttt{CEV}_{n,d})$ *is negligible for any adversary* $\mathcal{A}$*, and*

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

B. Security

We encode the garbling scheme security games $\texttt{PRVSIM}^b_{\mathsf{gs},\Phi,\mathcal{S}}$ from Section II-A in SSP for fixed leakage $\Phi(C) = C$. The package $\texttt{MOD-PRVSIM}^b_{n,d}$ (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.

Real game $\texttt{PRVSIM}^0_{n,d}(\texttt{GB}_{n,d},$ $\texttt{DINF})$ garbles as in the garbling scheme (Fig. 14a). Ideal game $\texttt{PRVSIM}^1_{n,d}(\texttt{SIM}_{n,d})$ is parametrized by package $\texttt{SIM}_{n,d}$ that simulates the garbling, 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 $\texttt{SIM}_{n,d}$ such that the real and ideal game are indistinguishable for every efficient adversary.

```
MOD-PRVSIM^b_{n,d}
GARBLE(C, x)
assert $\tilde{C} = \bot$
assert width(C) = n
assert depth(C) = d
for j = 1..n do
    SETBIT_j(x_j)
if b = 1 then EVAL(C)
$\tilde{C}$ ← GBL(C)
dinf ← GETDINF
for j = 1..n do
    $\tilde{x}[j]$ ← GETA^out_j
return ($\tilde{C}$, $\tilde{x}$, dinf)
```

Figure 13: Code of $\texttt{MOD-PRVSIM}^b_{n,d}$.

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

$$\mathsf{Adv}(\mathcal{A}; \mathtt{PRVSIM}^0_{n,d}(\mathtt{GB}_{n,d}, \mathtt{DINF}), \mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{n,d}))$$

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

Appendix E sketches the required modifications to prove security of Yao's garbling scheme for $\Phi(C) = \mathsf{topo}(C)$.

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

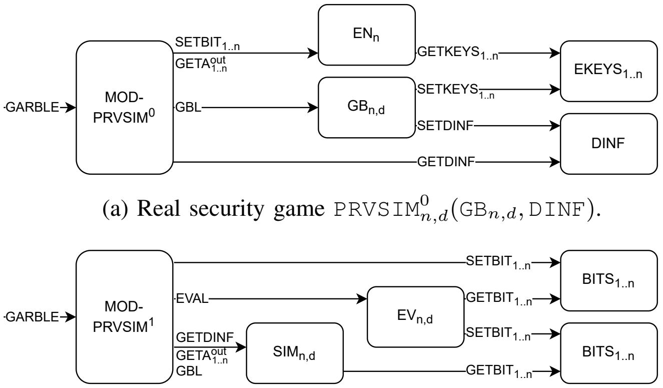
Yao's construction uses an IND-CPA secure symmetric encryption scheme $(kgen, enc, dec)$ where $kgen$ outputs uniformly random bitstrings[2].

1) Circuit garbling: Remember our assumption that circuits are layered. To garble a circuit C of depth d with n inputs and width n, Yao's garbling scheme first chooses 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. 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 $g_{i,j}$. Four ciphertexts are computed by 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 $x = x_1||..||x_n$.



(a) Real security game $\mathtt{PRVSIM}^0_{n,d}(\mathtt{GB}_{n,d}, \mathtt{DINF})$.

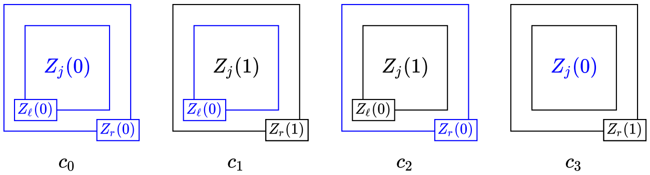
(b) Ideal security game $\mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{n,d})$.

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

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 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 ac-

[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 [18, Definition 2] for details.

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

tive/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 tuples

$\mathsf{gs}_{\text{tdyao}} = \{(\mathtt{GB}_{\text{tdyao},n,d}, \mathtt{DE}_{\text{tdyao},n}, \mathtt{DINF}_{\text{tdyao}}, \mathtt{GEV}_{\text{tdyao},n,d})\}_{n,d\in\mathbb{N}}$. Since the behaviour of garbled evaluation package $\mathtt{GEV}_{\text{tdyao},n,d}$ and output decoding package $\mathtt{DE}_{\text{tdyao},n}$ are conceptually induced by the behaviour of the garbling package $\mathtt{GB}_{\text{tdyao},n,d}$, and since the security definition only depends on $\mathtt{GB}_{\text{tdyao},n,d}$ and $\mathtt{DINF}_{\text{tdyao}}$, we omit the description of the former. $\mathtt{GB}_{\text{tdyao},n,d}$ has a single oracle GBL, and $\mathtt{DINF}_{\text{tdyao}}$ is similar to $\mathtt{EKEYS}$(cf. Fig. 15). To garble a circuit C, oracle GBL first performs some input checks, 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 Yao's garbling scheme as defined in Section IV-B:

Theorem 1 (Security of Yao's garbling scheme)**.** *Let se be the symmetric encryption scheme used within gs_{tdyao}. Then for all $n, d \in \mathbb{N}$, there exists a PPT simulator $\mathtt{SIM}_{tdyao,n,d}$ and reduction $\mathcal{R}$ such for all PPT adversaries $\mathcal{A}$,*

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

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

Looking ahead, the strategy of simulator $\mathtt{SIM}_{\text{tdyao},n,d}$ (shown for completeness in Fig. 15) will be to perform all computations using the active/inactive semantics of wire keys. Remember that a key is *active* if the evaluator will learn it. The simulator's strategy is thus consistent with *any* possible input and hence does not require knowledge of the concrete input chosen by the adversary, as long as decoding the garbled output still yields the correct output $C(x)$. The simulator starts by sampling wire keys $S_{i,j}$ with active/inactive semantics for all circuit wires. For each gate, $\mathtt{SIM}_{\text{tdyao},n,d}$ then computes four ciphertexts: One is an encryption of the active output wire key $S_{i,j}(0)$ under both active input wire keys $S_{i\text{-}1,\ell}(0)$, $S_{i\text{-}1,r}$, the remaining ciphertexts encrypt 0^λ under all other input wire key combinations. The decoding information maps active output wire keys $S_{d,j}$ to the correct output bits, and the garbled input consists of the active input wire keys $S_{0,j}$.

Oracle of $\mathtt{GB}_{\text{tdyao},n,d}$

```
GBL(C)
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
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  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 $\mathtt{DINF}_{\text{tdyao}}$

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

GETDINF
return dinf
```

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

```
GBL(C)
for $i = 1..d$ do
  for $j = 1..n$ do
    $S_{i,j}(0) \leftarrow\$ \{0,1\}^\lambda$
    $S_{i,j}(1) \leftarrow\$ \{0,1\}^\lambda$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$
    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
  $z_{d,j} \leftarrow \mathsf{GETBIT}_j$
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  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).

C. Proof outline

The challenge will be to show that active/inactive semantics during simulation are applied correctly throughout the circuit, and that the two garbling strategies are indistinguishable, which is reduced to encryption scheme security. The argument is broken down to the level of individual gates. We use symmetric encryption security once for each gate in the circuit. Afterwards, the proof connects the *gate* garbling arguments and turns them into an argument about *circuit* garbling, and relates 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.

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

```
Oracle of MODGB_{n,i}
GBL_i($\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}$)
assert $\tilde{C}[i] = \bot$
assert $|\boldsymbol{\ell}|, |\boldsymbol{r}|, |\boldsymbol{op}| = n$
for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{C}[i]_j \leftarrow \mathsf{GBLG}(\ell, r, op, j)$
$\tilde{C}[i] \leftarrow \tilde{C}[i]_{1..n}$
return $\tilde{C}[i]$

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

Oracle of SIM_{gate,n}
GBLG($\ell, r, op, j$)
$\tilde{g}_j \leftarrow \bot$
$\mathsf{EVAL}_j(\ell, r, op)$
$S_j^{\text{out}}(0) \leftarrow \mathsf{GETA}_j^{\text{out}}$
$S_r^{\text{in}}(0) \leftarrow \mathsf{GETA}_r^{\text{in}}$
$S_r^{\text{in}}(1) \leftarrow \mathsf{GETINA}_r^{\text{in}}$
$S_\ell^{\text{in}}(0) \leftarrow \mathsf{GETA}_\ell^{\text{in}}$
$S_\ell^{\text{in}}(1) \leftarrow \mathsf{GETINA}_\ell^{\text{in}}$
for $(d_\ell, d_r) \in \{0,1\}^2$ :
    $k_\ell^{\text{in}} \leftarrow S_\ell^{\text{in}}(d_\ell)$
    $k_r^{\text{in}} \leftarrow S_r^{\text{in}}(d_r)$
    if $d_\ell = d_r = 0$ :
        $k_j^{\text{out}} \leftarrow S_j^{\text{out}}(0)$
    else $k_j^{\text{out}} \leftarrow 0^\lambda$
    $c_{\mathsf{in}} \leftarrow\$ enc(k_r^{\text{in}}, k_j^{\text{out}})$
    $c \leftarrow\$ enc(k_\ell^{\text{in}}, c_{\mathsf{in}})$
    $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$
```

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

(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 $\mathsf{gs}_{\text{tdyao}}$.

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 $\mathtt{GB}^0_{yao,n,i}$

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 $\mathtt{GB}^0_{\text{yao},n,i}$ reflects this structure: Each $\mathtt{GB}^0_{\text{yao},n,i}$ is composed of the packages $\mathtt{MODGB}_{n,i}$, $\mathtt{GATE}_n$ and $\mathtt{ENC}^0_{1\ldots n}$ (cf. Fig. 16 and 17). $\mathtt{GATE}_n$ garbles a gate and makes (simple) encryption queries to $\mathtt{ENC}^0_{1\ldots n}$. We here assume that se encrypts messages of length λ always to ciphertext of the same length. $\mathtt{MODGB}_{n,i}$ modularizes the garbling of a layer and queries oracle GBLG of $\mathtt{GATE}$ for each gate in the layer.

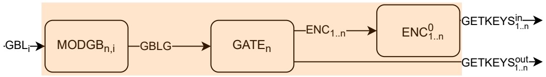


Figure 17: Layer garbling package $\mathtt{GB}^0_{\text{yao},n,i}$

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

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

where Fig. 16 defines $\mathtt{MODGB}_{n,i}$ and $\mathtt{GATE}_n$, 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 $\mathtt{GB}^1_{\text{yao},n,i}$ and the real game uses layer garbling package $\mathtt{GB}^0_{\text{yao},n,i}$ 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 $\mathtt{LEV}_j$ as shown in Fig. 18.

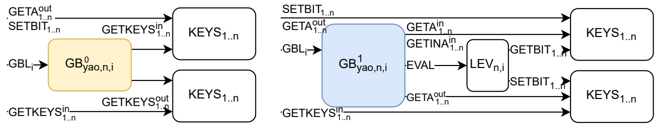
```
LEV_j
EVAL_j($\ell, r, op$)
$z_\ell \leftarrow \mathsf{GETBIT}_\ell$
$z_r \leftarrow \mathsf{GETBIT}_r$
$z_j \leftarrow op(z_{i-1,\ell}, z_{i-1,r})$
$\mathsf{SETBIT}_j(z_j)$

KEYS
GETKEYS^out
assert flag $= 0$
flag $\leftarrow 1$
if $Z = \bot$ then
    $Z(0) \leftarrow\$ \{0,1\}^\lambda$
    $Z(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z$

GETA^in
assert flag
return $Z(z)$

GETINA^in
assert flag
return $Z(1 - z)$
```

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

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

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

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



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

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

Figure 19: Layer security games.

Games $\mathtt{LSEC}^b_n(\mathtt{GB}^b_{\text{yao},n,i})$, $b \in \{0,1\}$, are layer version of the selective security games $\mathtt{PRVSIM}^0_{n,d}(\mathtt{GB}_{\text{tdyao},n,d}, \mathtt{DINF}_{\text{tdyao}})$ and $\mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{\text{tdyao},n,d})$, modified to be composable:

- The adversary inputs a *single circuit layer* to the game,
- the adversary's query to the game is split into SETBIT, GBL, $\mathsf{GETA}^{\text{out}}$ and $\mathsf{GETDINF}$ queries,
- input/output keys and bits are stored in $\mathtt{KEYS}$ packages,
- the simulator gets the input keys via $\mathsf{GETINA}^{\text{in}}$ and $\mathsf{GETA}^{\text{in}}$ queries from the top $\mathtt{KEYS}$ package and the *active* output key via $\mathsf{GETA}^{\text{out}}$ from the lower $\mathtt{KEYS}$ 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 is 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 $\texttt{GB}^0_{\text{yao,n,i}}$ reduces to 2-key IND-CPA security of the underlying encryption scheme se.

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

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

Gate garbling simulator $\texttt{SIM}_{\text{gate},n}$ (Fig. 16) works as follows: Instead of garbling a gate based on the 0/1 semantics of wire keys like the real $\texttt{GB}^0_{\text{yao,n,i}}$, it uses their inactive/active semantics. The simulator first retrieves all relevant wire keys except for the inactive output wire key. The ciphertext 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 $\texttt{GB}^1_{\text{yao,n,i}} :=$ $\texttt{MODGB}_{n,i} \to \texttt{SIM}_{\text{gate},n}$ extends this to layer garbling.

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

Claim 1 (Real Code Equivalence)**.** *Let $n \in \mathbb{N}$. $\forall 1 \leq i \leq d$,*

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

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

Claim 1 follows by graph equality of $\texttt{LSEC}^0_n(\texttt{GB}^0_{\text{yao},n,i})$ (Fig. 20c) and $\mathcal{R}^i_{\text{layer},n} \to \texttt{2CPA}^0_{1..n}(se)$ (Fig. 20b).

Claim 2 (Ideal Code Equivalence)**.** *Let $n \in \mathbb{N}$. $\forall 1 \leq i \leq d$,*

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

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

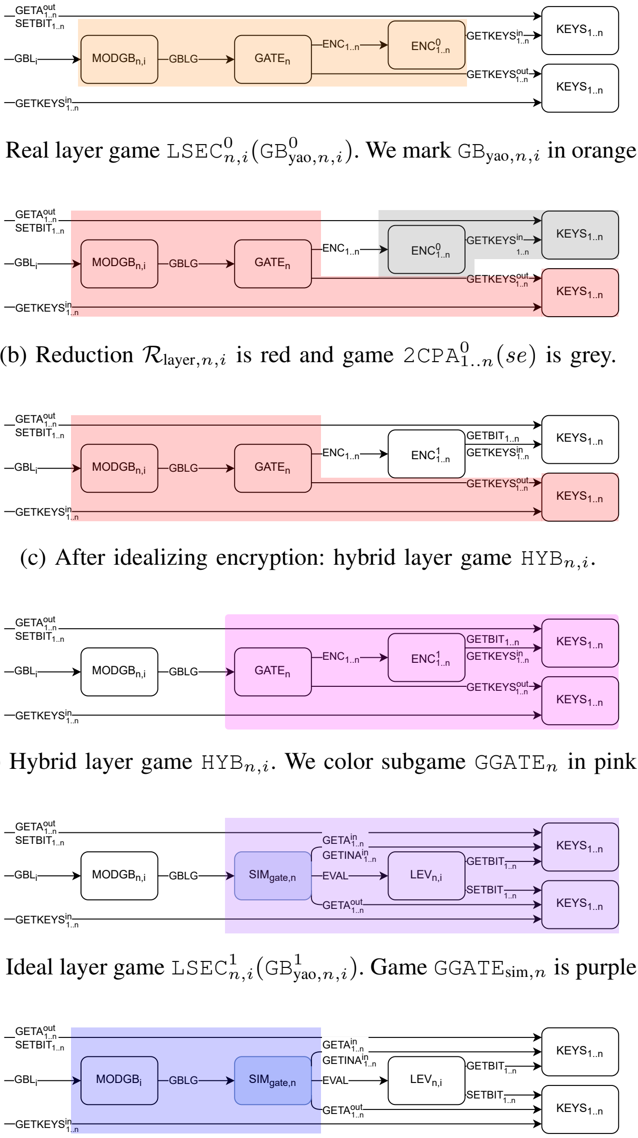
Claim 2 will be proved in a moment. Applying the perfect reduction lemma with Claims 1 and 2 concludes our proof. □

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. In order to show code equivalence of $\mathcal{R}^i_{\text{layer},n} \to \texttt{IND-CPA}^1_{1..n}(se)$ and $\texttt{LSEC}^1_n(\texttt{GB}^1_{\text{yao},n,i})$, we define real and ideal gate garbling subgames $\texttt{GGATE}_n$ and $\texttt{GGATE}_{\text{sim},n}$ (Fig. 20d and Fig. 20e). If we show that

$$\texttt{GGATE}_n \overset{\text{code}}{\equiv} \texttt{GGATE}_{\text{sim},n}, \tag{4}$$

then the layer garbling games in Fig. 20c and 20f are functionally equivalent and we obtain Claim 2. The proof of Equation 4 is an inlining argument that changes how gates are garbled, see Appendix B for details. The argument first inlines all packages, then changes the garbling of a gate from using 0/1 semantics to the equivalent computation using active/inactive semantics, and then factors out the relevant packages again. □



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

(b) Reduction $\mathcal{R}_{\text{layer},n,i}$ is red and game $\texttt{2CPA}^0_{1..n}(se)$ is grey.

(c) After idealizing encryption: hybrid layer game $\texttt{HYB}_{n,i}$.

(d) Hybrid layer game $\texttt{HYB}_{n,i}$. We color subgame $\texttt{GGATE}_n$ in pink.

(e) Ideal layer game $\texttt{LSEC}^1_{n,i}(\texttt{GB}^1_{\text{yao},n,i})$. Game $\texttt{GGATE}_{\text{sim},n}$ is purple.

(f) Ideal layer game $\texttt{LSEC}^1_{n,i}(\texttt{GB}^1_{\text{yao},n,i})$. Simulator $\texttt{GB}^1_{\text{yao},n,i}$ is blue.

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

VII. CIRCUIT SECURITY

A. Yao's circuit garbling package $\texttt{GB}_{yao,n,d}$

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

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

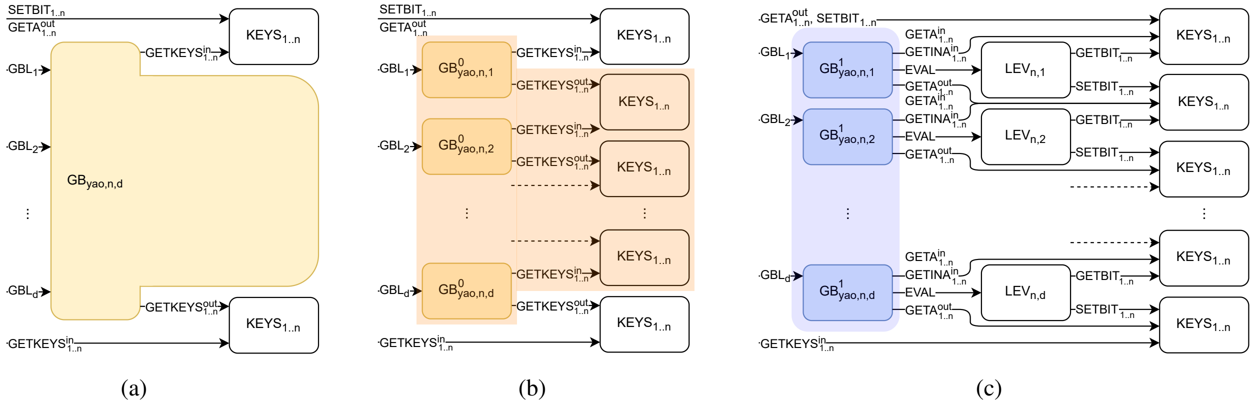


Figure 21: Fig. 21a-21b display game $\mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d})$ and package $\mathtt{GB}_{\text{yao},n,d}$ (orange) in different representations. Fig. 21c defines $\mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d})$ where $\mathtt{SIM}_{\text{yao},n,d}$ is the parallel composition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

B. Circuit garbling security

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

Lemma 5 (Circuit Security)**.** *Let $n, d \in \mathbb{N}$. Then, for each $1 \leq i \leq d$, there exists a PPT reduction $\mathcal{R}^i_{circ,n,d}$ such that for all PPT adversaries $\mathcal{A}$,*

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

Proof of Lemma 5. We reduce circuit garbling security to layer garbling security via a hybrid argument over the d layers of the circuit. Real game $\mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d})$ is hybrid 0, and hybrid d is the ideal game $\mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d})$. We start by rewriting $\mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d})$ and $\mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d})$ as

$$\mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d}) \overset{\text{code}}{\equiv} \mathcal{R}^1_{\text{circ},n,d} \to \mathtt{LSEC}^0_n(\mathtt{GB}^0_{\text{yao},1,n}) \quad (5)$$

$$\mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d}) \overset{\text{code}}{\equiv} \mathcal{R}^d_{\text{circ},n,d} \to \mathtt{LSEC}^1_n(\mathtt{GB}^1_{\text{yao},d,n}), \quad (6)$$

where Fig. 22b and Fig. 22e define the reductions $\mathcal{R}^1_{\text{circ},n,d}$ and $\mathcal{R}^d_{\text{circ},n,d}$, respectively. Both equivalences hold by associativity of package composition (cf. Fig. 22b and Fig. 22e).

Generalizing the reductions yields $\mathcal{R}^i_{\text{circ},n,d}$ for all $i \in \{1, .., d\}$ as in Fig. 22c. Then for any $i \in \{1, .., d\text{-}1\}$, we can define the i-th hybrid in the following two equivalent ways:

$$\begin{aligned}&\mathcal{R}^i_{\text{circ},n,d} \to \mathtt{LSEC}^1_n(\mathtt{GB}^1_{\text{yao},n,i})\\ &\overset{\text{code}}{\equiv} \mathcal{R}^{i+1}_{\text{circ},n,d} \to \mathtt{LSEC}^0_n(\mathtt{GB}^0_{\text{yao},n,i+1}) \qquad (7)\end{aligned}$$

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

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}; \mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d}), \mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d}))\\ =\;& \Pr\left[1 \leftarrow\!\$\; \mathcal{A} \to \mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d})\right]\\ &- \Pr\left[1 \leftarrow\!\$\; \mathcal{A} \to \mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d})\right]\\ \overset{(5),(6)}{=}\;& \Pr\left[1 \leftarrow\!\$\; \mathcal{A} \to \mathcal{R}^1_{\text{circ},n,d} \to \mathtt{LSEC}^0_n(\mathtt{GB}^0_{\text{yao},1,n})\right]\\ &- \Pr\left[1 \leftarrow\!\$\; \mathcal{A} \to \mathcal{R}^d_{\text{circ},n,d} \to \mathtt{LSEC}^1_n(\mathtt{GB}^1_{\text{yao},d,n})\right]\end{aligned}$$

Applying a telescopic sum and (7) then yields

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

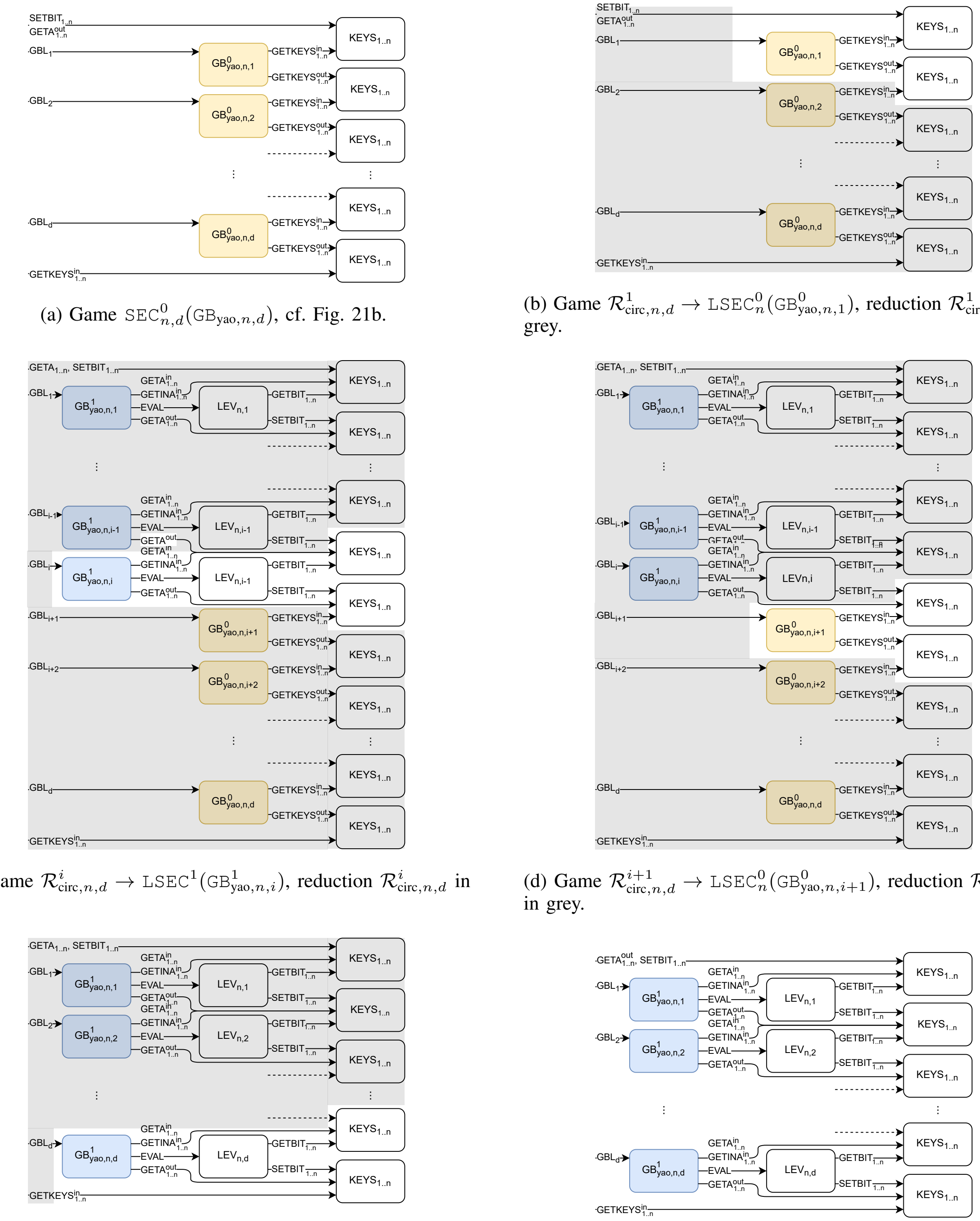
□

Combining circuit and layer security with 2CPA security (Lemma 5, 4 and Corollary 1) yields the following corollary.

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

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

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



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

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

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

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

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

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

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

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

VIII. Alignment with $\mathtt{PRVSIM}^b_{n,d}$

To conclude our proof of Theorem 1, it remains to reduce selective security of Yao's garbling scheme to its circuit garbling security established in the previous section. Towards this goal, we define a reduction package $\mathtt{MOD}_{n,d}$ (Fig. 23) and apply the perfect reduction lemma (Lemma 1) one last time. $\mathtt{MOD}_{n,d}$ provides a single GARBLE oracle and queries the oracles of a circuit security game in the right order to garble circuit C and input x.

```
Oracle of $\mathtt{MOD}_{n,d}$
GARBLE$(C, x)$
assert $\tilde{C} = \bot$
assert width$(C) = n$
assert depth$(C) = d$
for $j = 1..n$ do
    SETBIT$_j(x_j)$
    $\tilde{x}[j] \leftarrow$ GETA$^{\mathrm{out}}_j$
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$^{\mathrm{in}}_j$
return $(\tilde{C}, \tilde{x}, \mathrm{dinf})$
```

Figure 23: $\mathtt{MOD}_{n,d}$.

Proof of Theorem 1. Consider simulator $\mathtt{SIM}_{\mathrm{tdyao},n,d}$ which we have already seen in Fig. 15. We show the following two claims and then apply Lemma 1 with reduction $\mathtt{MOD}_{n,d}$.

Claim 3 (Real game equivalence). *For all $n, d \in \mathbb{N}$,*

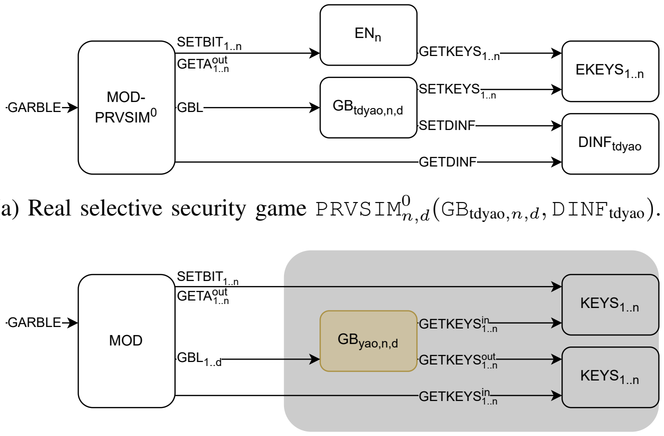
$$
\begin{aligned}
&\mathtt{PRVSIM}^0_{n,d}(\mathrm{GB}_{\mathit{tdyao},n,d}, \mathtt{DINF}_{\mathit{tdyao}})\\
&\overset{\mathit{code}}{\equiv} \mathtt{MOD}_{n,d} \to \mathtt{SEC}^0_{n,d}(\mathrm{GB}_{\mathit{yao},n,d})
\end{aligned}
$$

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

Claim 4 (Ideal game equivalence). *For all $n, d \in \mathbb{N}$,*

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

Fig. 25 shows the games. The claim follows directly after inlining all packages. For details, see Appendix D.



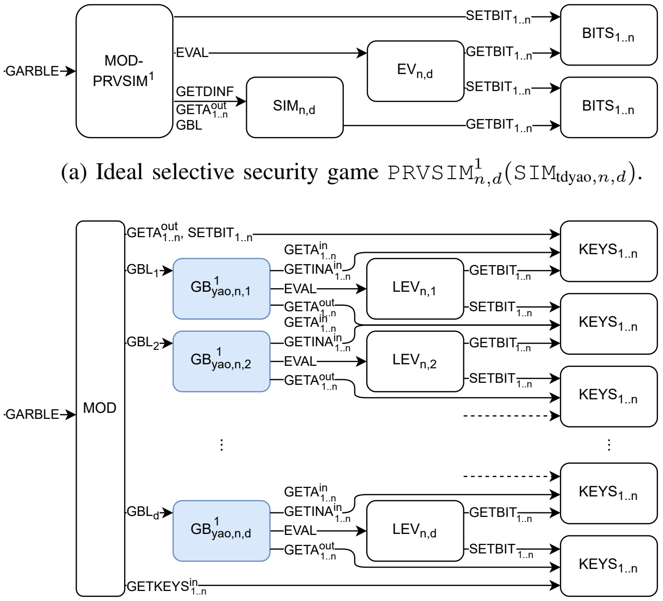
(a) Real selective security game $\mathtt{PRVSIM}^0_{n,d}(\mathrm{GB}_{\mathrm{tdyao},n,d}, \mathtt{DINF}_{\mathrm{tdyao}})$.

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

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

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

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



(a) Ideal selective security game $\mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{\mathrm{tdyao},n,d})$.

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

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

IX. Discussion

We now revisit our new security proof for Yao's garbling scheme to discuss insights and compare with existing proofs.

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. $\texttt{MOD}_{n,d} \rightarrow \mathcal{R}_{\text{hyb},n,d} \rightarrow \texttt{2CPA}^b(se)$, would then yield the typical proof presentation as sequence of direct game hops between $\texttt{PRVSIM}^0_{n,d}(\texttt{GB}_{\text{tdyao},n,d}, \texttt{DINF}_{\text{tdyao}})$ and $\texttt{PRVSIM}^1_{n,d}(\texttt{SIM}_{\text{tdyao},n,d})$ 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) for 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 $\texttt{GB}_{\text{tdyao},n,d}$ (Fig. 15), and before garbling the actual circuit. Our game $\texttt{SEC}^0_{n,d}(\texttt{GB}_{\text{yao},n,d})$ 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, our circuit garbling security notion in Section VII differs from the selective security notion introduced in Section IV 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 $\texttt{MOD}_{n,d}$ 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 $\texttt{SIM}_{\text{yao},n,d}$ 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 about circuit layers with local simulators, all 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 [18] 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 [6] 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 adversarially chosen key. 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) Layered circuits: Starting with the work of Hemenway et al. [12], the assumption of layered circuits has been leveraged successfully in the context of *adaptive* security of various garbling schemes ([15], [14], [13], [16]). The additional circuit structure is used to identify the sequence of hybrid games during the security proof, however neither a layer security notion nor layer simulation are considered explicitly.

Transforming an arbitrary circuit into a layered one incurs at most a quadratic increase in size [20], though circuits in many practical scenarios (e.g. AES) are naturally layered. Depending on the concrete circuit to be garbled, the security loss incurred by our proof is thus potentially larger than in previous selective security proofs for Yao's garbling scheme [18], [6], and hence there seems to be a trade-off between proof modularity and tightness of the security reduction. Our results can be modified to non-layered circuits by formulating appropriate gate assumptions instead of layer assumptions, at the cost of losing the convenient visual representation since we are not aware of a general notation for the resulting package composition with arbitrary dependencies between gate assumptions.

3) 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 [3] 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. [12] on adaptive security of a modification of Yao's garbling scheme, which ultimately follows the proof outline of Lindell and Pinkas [18]. 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: The SSP-style shaped our proof and impacted its length. One core benefit 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 helps particularly 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 to our setting. When self-composing a package 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 SSP concept, 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 semantic information of the keys to our KEYS packages. Finally, our work is the first to use *simulators* in an SSP context. Games which are parameterized by simulators are convenient because they offer flexibility. Our simulator is simply the code/package composition which emerges after a sufficient number of game transformations and has the correct oracles and dependencies. In follow-up work, Brzuska, Delignat-Lavaud, Egger, Fournet, Kohbrok and Kohlweiss [8] also use simulation-based security for their definition of the TLS 1.3 key schedule.

D. Formal verification

Security of Yao's garbling scheme has been formally verified: Li and Micciancio [17] 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 [2] mechanized the BHR proof for Yao's garbling scheme in EasyCrypt [4], a proof assistant for code-based game-playing proofs. Recent works on mechanizing SSP-style proofs (SSProve [1], Dupressoir, Kohbrok and Oechsner [11]) as well as discussions with the authors of [2] give us hope that also our structured code-based proof can be mechanized. However, formal verification is orthogonal to the scope of this paper, and we leave this question as future work.

Acknowledgments

We are grateful to Christoph Egger, Pihla Karanko, Markulf Kohlweiss, Kirthivaasan Puniamurthy, Jan Winkelmann, 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 [2]. 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

[1] 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. *CSF 2021*, pages 1–15, 2021.

[2] 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. *ACM CCS 2017*, pages 1989–2006, 2017.

[3] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472, 2018.

[4] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90, 2011.

[5] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153, 2012.

[6] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. *ACM CCS 2012*, pages 784–796, 2012.

[7] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, 2006.

[8] Chris Brzuska, Antoine Delignat-Lavaud, Christoph Egger, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Key-schedule security for the TLS 1.3 standard. *ASIACRYPT 2022*, volume 13791 of *LNCS*, pages 621–650, 2022. https://static.siccegge.de/pdfs/BDEFKK22.pdf.

[9] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249, 2018.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *42nd FOCS*, pages 136–145, 2001.

[11] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing State-Separating Proofs to EasyCrypt - A Security Proof for Cryptobox. CSF 2022, pages 211–226, 2022.

[12] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178, 2016.

[13] Zahra Jafargholi and Sabine Oechsner. Adaptive security of practical garbling schemes. *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 741–762, 2012.

[14] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. *TCC 2017, Part II*, pages 40–71, 2017.

[15] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. *TCC 2016-B, Part I*, pages 433–458, 2016.

[16] Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. On treewidth, separators and yao's garbling. *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 486–517, 2021.

[17] Baiyu Li and Daniele Micciancio. Symbolic security of garbled circuits. CSF 2018, pages 147–161, 2018.

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

[19] Ueli Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. TOSCA 2011, volume 6993 of *LNCS*, pages 33–56, 2012.

[20] Ingo Wegener. *The complexity of Boolean functions*. 1987.

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

Appendix

A. Proof of Lemma 2

The proof of Lemma 2 establishes that $\mathtt{2CPA}^b(se)$ is functionally equivalent to $\rightarrow$ $\mathtt{IND\text{-}CPA}^b(se)$ via inlining, which is provided in Fig. 26. The left-most column contains the packages KEYS and $\mathtt{ENC}^b$ of the $\mathtt{2CPA}^b(se)$ game, while

```
Oracle of $\texttt{ENC}^b$
$\mathsf{ENC}(d, m_0, m_1)$
$Z^{\text{in}} \leftarrow \mathsf{GETKEYS}^{\text{in}}()$
assert $|m_0| = |m_1|$
$c \leftarrow\$ enc(Z^{\text{in}}(d), m_0)$
if $b = 1$
    $z^{\text{in}} \leftarrow \mathsf{GETBIT}()$
    if $z^{\text{in}} \neq d$ then
        $c \leftarrow\$ enc(Z^{\text{in}}(d), m_b)$
return $c$

Oracles of $\texttt{KEYS}$
$\mathsf{SETBIT}(z')$
assert $z = \bot$
$z \leftarrow z'$
return $()$

$\mathsf{GETA}^{\text{out}}()$
assert $z \neq \bot$
$\texttt{flag} \leftarrow 1$
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 $z$

$\mathsf{GETKEYS}^{\text{in}}()$
assert $\texttt{flag}$
return $Z$
```

```
$\texttt{GAME}_2^b$
$\mathsf{ENC}(d, m_0, m_1)$
assert $\texttt{flag}$
assert $|m_0| = |m_1|$
$c \leftarrow\$ enc(Z(d), m_0)$
if $b = 1$ then
    assert $z \neq \bot$
    if $z \neq d$ then
        $c \leftarrow\$ enc(Z(d), m_b)$
return $c$

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

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

```
$\texttt{GAME}_3^b$
$\mathsf{ENC}(d, m_0, m_1)$
assert $\texttt{flag}$
assert $|m_0| = |m_1|$
$c \leftarrow\$ enc(Z(d), m_0)$
if $z \neq d$ then
    $c \leftarrow\$ enc(Z(d), m_b)$
return $c$

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

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

```
$\texttt{GAME}_4^b$
$\mathsf{ENC}(d, m_0, m_1)$
assert $\texttt{flag}$
assert $|m_0| = |m_1|$
$c \leftarrow\$ enc(Z(z), m_0)$
if $z \neq d$ then
    assert $k \neq \bot$
    $c \leftarrow\$ enc(k, m_b)$
return $c$

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

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

```
$\texttt{RED}$
$\mathsf{ENC}(d, m_0, m_1)$
assert $\texttt{flag}$
assert $|m_0| = |m_1|$
$c \leftarrow\$ enc(Z(z), m_0)$
if $z \neq d$ then
    $c \leftarrow \mathsf{ENC}(m_0, m_1)$
return $c$

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

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

Oracles of $\texttt{IND-CPA}^b(se)$
$\mathsf{SMP}()$
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: $\texttt{2CPA}^b(se) \overset{\text{code}}{\equiv} \texttt{RED} \rightarrow \texttt{IND-CPA}^b(se)$

the right-most column contains the package of $\texttt{RED}$ and $\texttt{IND-CPA}^b(se)$. Then, from the left-most column to $\texttt{GAME}_2^b$, we use inlining to describe the $\texttt{2CPA}^b(se)$ game as a single package. That is, we merge the state of $\texttt{ENC}^b$ (originally no state) $\texttt{KEYS}$ (originally had state Z^{in} and z) and inline the $\mathsf{GETKEYS}^{\text{in}}$ and GETBIT calls into the $\texttt{ENC}^b$ oracle while renaming z^{in} to z; $\texttt{GAME}_2^b$ continues to expose the SETBIT and $\mathsf{GETA}^{\text{out}}$ queries.

In turn, from the right-most column to $\texttt{GAME}_4^b$, we inline the SMP and ENC calls of $\texttt{RED}$ to $\texttt{IND-CPA}^b(se)$ into $\texttt{RED}$ and also merge their state. Note that the ENC oracle would now contain the line **assert** $|m_0| = |m_1|$ twice and we omit this redundancy already in the inlining step.

From $\texttt{GAME}_4^b$ to $\texttt{GAME}_3^b$, we rename k to $Z(z-1)$ and sample into $Z(0)$ and $Z(1)$ instead of $Z(z)$ and $Z(z-1)$, which does not affect behaviour since the sampling operations are independent. Additionally, we omit the **assert** $k \neq \bot$ condition, since $\texttt{flag} = 1$ implies that $\mathsf{GETA}^{\text{in}}$ was called before, and it already sampled the keys. Moreover, before the **if**-clause, $\texttt{GAME}_3^b$ encrypts using $Z(d)$ instead of $Z(z)$. In the case that $d \neq z$, this change does not affect the output behaviour, since c is overwritten in the **if**-branch which follows.

From $\texttt{GAME}_2^b$ to $\texttt{GAME}_3^b$, we remove **assert** $z \neq \bot$, because it is redundant as $\texttt{flag} = 1$ already implies that **assert** $z \neq \bot$, since $\texttt{flag}$ is set to 1 by $\mathsf{GETA}^{\text{out}}$ which asserts that $z \neq \bot$.

```
Oracles of $\mathtt{GGATE}_n$
$\mathsf{SETBIT}_i(z)$
return $\mathsf{SETBIT}_i(z)$

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

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

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

```
Oracles of $\mathtt{GGATE}_n$
$\mathsf{SETBIT}_i(z)$
return ()

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

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
$z_\ell^{\text{in}} \leftarrow \mathsf{GETBIT}(\ell)$
$z_r^{\text{in}} \leftarrow \mathsf{GETBIT}(r)$
$Z_j^{\text{out}} \leftarrow \mathsf{GETKEYS}_j^{\text{out}}$
$Z_\ell^{\text{in}} \leftarrow \mathsf{GETKEYS}_\ell^{\text{in}}$
$Z_r^{\text{in}} \leftarrow \mathsf{GETKEYS}_r^{\text{in}}$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
  $b_j \leftarrow op(b_\ell, b_r)$
  $k_j^{\text{out}} \leftarrow Z_j^{\text{out}}(b_j)$
  $k_\ell^{\text{in}} \leftarrow Z_\ell^{\text{in}}(b_\ell)$
  if $z_\ell^{\text{in}} = b_\ell$ :
    $c_{\mathsf{in}}^0 \leftarrow\$ enc(k_\ell^{\text{in}}, k_j^{\text{out}})$
  if $z_\ell^{\text{in}} \neq b_\ell$ :
    $c_{\mathsf{in}}^0 \leftarrow\$ enc(k_\ell^{\text{in}}, 0^\lambda)$
  $c_{\mathsf{in}}^1 \leftarrow\$ enc(k_\ell, 0^\lambda)$
  $k_r^{\text{in}} \leftarrow Z_r^{\text{in}}(b_r)$
  if $z_r^{\text{in}} = b_r$ :
    $c \leftarrow\$ enc(k_r^{\text{in}}, c_{\mathsf{in}}^0)$
  if $z_r^{\text{in}} \neq b_r$ :
    $c \leftarrow\$ enc(k_r^{\text{in}}, c_{\mathsf{in}}^1)$
  $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

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

```
Oracles of $\mathtt{GGATE}_n$
$\mathsf{SETBIT}_i(z)$
assert $z_i^{\text{in}} = \bot$
$z_i^{\text{in}} \leftarrow z$
return ()

$\mathsf{GETA}_i^{\text{out}}$
assert $z_i^{\text{in}} \neq \bot$
$\mathtt{flag}_i^{\text{in}} \leftarrow 1$
if $Z_i^{\text{in}} = \bot$ :
  $Z_i^{\text{in}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_i^{\text{in}}(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z_i^{\text{in}}(z_i^{\text{in}})$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z_\ell^{\text{in}} \neq \bot$
assert $z_r^{\text{in}} \neq \bot$
$\mathtt{flag}_j^{\text{out}} \leftarrow 1$
if $Z_j^{\text{out}} = \bot$ :
  $Z_j^{\text{out}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_j^{\text{out}}(1) \leftarrow\$ \{0,1\}^\lambda$
assert $\mathtt{flag}_\ell^{\text{in}} = 1$
assert $\mathtt{flag}_r^{\text{in}} = 1$
for $(b_\ell, b_r) \in \{0,1\}^2$ :
  $k_\ell^{\text{in}} \leftarrow Z_\ell^{\text{in}}(b_\ell)$
  $k_r^{\text{in}} \leftarrow Z_r^{\text{in}}(b_r)$
  if $b_\ell = z_\ell^{\text{in}} \wedge b_r = z_r^{\text{in}}$ :
    $b_j \leftarrow op(b_\ell, b_r)$
    $k_j^{\text{out}} \leftarrow Z_j^{\text{out}}(b_j)$
  else $k_j^{\text{out}} \leftarrow 0^\lambda$
  $c_{\mathsf{in}} \leftarrow\$ enc(k_\ell^{\text{in}}, k_j^{\text{out}})$
  $c \leftarrow\$ enc(k_r^{\text{in}}, c_{\mathsf{in}})$
  $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}_j^{\text{in}}$
assert $\mathtt{flag}_j^{\text{out}} = 1$
assert $Z_j^{\text{out}} \neq \bot$
return $Z_j^{\text{out}}$
```

```
Oracles of $\mathtt{GGATE}_n$
$\mathsf{SETBIT}_i(z)$
assert $z_i^{\text{in}} = \bot$
$z_i^{\text{in}} \leftarrow z$
return ()

$\mathsf{GETA}_i^{\text{out}}$
assert $z_i^{\text{in}} \neq \bot$
$\mathtt{flag}_i^{\text{in}} \leftarrow 1$
if $Z_i^{\text{in}} = \bot$ :
  $Z_i^{\text{in}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_i^{\text{in}}(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z_i^{\text{in}}(z_i^{\text{in}})$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z_\ell^{\text{in}} \neq \bot$
assert $z_r^{\text{in}} \neq \bot$
$\mathtt{flag}_j^{\text{out}} \leftarrow 1$
if $Z_j^{\text{out}} = \bot$ :
  $Z_j^{\text{out}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_j^{\text{out}}(1) \leftarrow\$ \{0,1\}^\lambda$
assert $\mathtt{flag}_r^{\text{in}} = 1$
assert $\mathtt{flag}_\ell^{\text{in}} = 1$
for $(b_\ell \oplus z_\ell^{\text{in}}, b_r \oplus z_r^{\text{in}})$
    $\in \{0,1\}^2$ :
  $k_\ell^{\text{in}} \leftarrow Z_\ell^{\text{in}}(b_\ell)$
  $k_r^{\text{in}} \leftarrow Z_r^{\text{in}}(b_r)$
  if $b_\ell \oplus z_\ell^{\text{in}} = b_r \oplus z_r^{\text{in}} = 0$ :
    $b_j \leftarrow op(b_\ell, b_r)$
    $k_j^{\text{out}} \leftarrow Z_j^{\text{out}}(b_j)$
  else $k_j^{\text{out}} \leftarrow 0^\lambda$
  $c_{\mathsf{in}} \leftarrow\$ enc(k_\ell^{\text{in}}, k_j^{\text{out}})$
  $c \leftarrow\$ enc(k_r^{\text{in}}, c_{\mathsf{in}})$
  $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}_j^{\text{in}}$
assert $\mathtt{flag}_j^{\text{out}} = 1$
assert $Z_j^{\text{out}} \neq \bot$
return $Z_j^{\text{out}}$
```

```
Oracles of $\mathtt{GGATE}_{\text{sim},n}$
$\mathsf{SETBIT}_i(z)$
assert $z_i^{\text{in}} = \bot$
$z_i^{\text{in}} \leftarrow z$
return ()

$\mathsf{GETA}_i^{\text{out}}$
assert $z_i^{\text{in}} \neq \bot$
$\mathtt{flag}_i^{\text{in}} \leftarrow 1$
if $Z_i^{\text{in}} = \bot$ :
  $Z_i^{\text{in}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_i^{\text{in}}(1) \leftarrow\$ \{0,1\}^\lambda$
return $Z_i^{\text{in}}(z_i^{\text{in}})$

$\mathsf{GBLG}(\ell, r, op, j)$
$\tilde{g}_j \leftarrow \bot$
assert $z_\ell^{\text{in}} \neq \bot$
assert $z_r^{\text{in}} \neq \bot$
$z_j^{\text{out}} \leftarrow op(z_\ell^{\text{in}}, z_r^{\text{in}})$
assert $z_j^{\text{out}} \neq \bot$
$\mathtt{flag}_j^{\text{out}} \leftarrow 1$
if $Z_j^{\text{out}} = \bot$ :
  $Z_j^{\text{out}}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_j^{\text{out}}(1) \leftarrow\$ \{0,1\}^\lambda$
$S_j^{\text{out}}(0) \leftarrow Z_j^{\text{out}}(z_j^{\text{in}})$
assert $\mathtt{flag}_r^{\text{in}} = 1$
$S_r^{\text{in}}(0) \leftarrow Z_r^{\text{in}}(z_r^{\text{in}})$
assert $\mathtt{flag}_r^{\text{in}} = 1$
$S_r^{\text{in}}(1) \leftarrow Z_r^{\text{in}}(1 \oplus z_r^{\text{in}})$
assert $\mathtt{flag}_\ell^{\text{in}} = 1$
$S_\ell^{\text{in}}(0) \leftarrow Z_\ell^{\text{in}}(z_\ell^{\text{in}})$
assert $\mathtt{flag}_\ell^{\text{in}} = 1$
$S_\ell^{\text{in}}(1) \leftarrow Z_\ell^{\text{in}}(1 \oplus z_\ell^{\text{in}})$
for $(d_\ell, d_r) \in \{0,1\}^2$ :
  $k_\ell^{\text{in}} \leftarrow S_\ell^{\text{in}}(d_\ell)$
  $k_r^{\text{in}} \leftarrow S_r^{\text{in}}(d_r)$
  if $d_\ell = d_r = 0$ :
    $k_j^{\text{out}} \leftarrow S_j^{\text{out}}(0)$
  else $k_j^{\text{out}} \leftarrow 0^\lambda$
  $c_{\mathsf{in}} \leftarrow\$ enc(k_r^{\text{in}}, k_j^{\text{out}})$
  $c \leftarrow\$ enc(k_\ell^{\text{in}}, c_{\mathsf{in}})$
  $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
return $\tilde{g}_j$

$\mathsf{GETKEYS}_j^{\text{in}}$
assert $\mathtt{flag}_j^{\text{out}} = 1$
assert $Z_j^{\text{out}} \neq \bot$
return $Z_j^{\text{out}}$
```

```
Oracles of $\mathtt{GGATE}_{\text{sim},n}$
$\mathsf{SETBIT}_i(z)$
return $\mathsf{SETBIT}_i(z)$

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

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

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

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

Moreover, we remove the additional condition in the **if**-branch that $b = 1$—this condition is unnecessary, since if $b = 0$, the operations $c \leftarrow\$ enc(Z(d), m_b)$ and $c \leftarrow\$ enc(Z(d), m_0)$ produce the same output distribution.

B. Proof of Claim 2

Proof of Claim 2, continued. It remains to prove Equation 4: $\mathtt{GGATE}_n \overset{\text{code}}{\equiv} \mathtt{GGATE}_{\text{sim},n}$. The equivalence follows from an inlining argument shown in Fig. 27. The second column is obtained from the first by inlining $\mathtt{ENC}^1_{1..n}$ and rearranging code, and similarly for the third column and inlining $\mathtt{KEYS}$. 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 $(b_\ell, b_r) \mapsto (b_\ell \oplus z_\ell^{\text{in}}, b_r \oplus z_r^{\text{in}})$ that maps bit values to active/inactive status to their corresponding, as indicated by z_ℓ^{in} and z_r^{in}. Further rearranging the code yields the fifth column, and factoring out $\mathtt{EV}$ and $\mathtt{KEYS}$ yields the last column. □

```
$\mathsf{G}^1_{\text{real}}$
$\mathsf{GARBLE}(C,x)$
assert $\tilde{C} = \bot$
assert $\mathsf{width}(C) = n$
assert $\mathsf{depth}(C) = d$
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 $(\tilde{C}, \tilde{x}, \text{dinf})$
```

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

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
    $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$

    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
            $b_j \leftarrow op(b_\ell, b_r)$
            $k_j \leftarrow Z_{i,j}(b_j)$
            $c_{\text{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), k_j)$
            $c \leftarrow\$ enc(Z_{i-1,r}(b_r), c_{\text{in}})$
            $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
        $\tilde{C}[i,j] \leftarrow \tilde{g}_j$
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 $\tilde{C} = \bot$
assert $\mathsf{width}(C) = n$
assert $\mathsf{depth}(C) = d$
for $j = 1..n$ do
    assert $z_j = \bot$
    $z_j \leftarrow x_j$
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
    $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$

    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
            $b_j \leftarrow op(b_\ell, b_r)$
            $k_j \leftarrow Z_{i,j}(b_j)$
            $c_{\text{in}} \leftarrow\$ enc(Z_{i-1,\ell}(b_\ell), k_j)$
            $c \leftarrow\$ enc(Z_{i-1,r}(b_r), c_{\text{in}})$
            $\tilde{g}_j \leftarrow \tilde{g}_j \cup c$
        $\tilde{C}[i,j] \leftarrow \tilde{g}_j$

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

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

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

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

            $\tilde{C}_j \leftarrow \tilde{C}_j \cup \{c\}$
    $\tilde{C}[i] \leftarrow \tilde{C}_{1..n}$
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 $\tilde{C} = \bot$
assert $\mathsf{width}(C) = n$
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 $\tilde{C}[i] = \bot$
    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 $\tilde{C} = \bot$
assert $\mathsf{width}(C) = n$
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.

C. Proof of Claim 3

The claim follows from an inlining argument shown in Fig. 28. Starting from $\mathsf{G}^1_{\text{real}} := \mathtt{PRVSIM}^0_{n,d}(\mathtt{GB}_{\text{tdyao},n,d}, \mathtt{DINF}_{\text{tdyao},n})$, we first inline packages $\mathtt{EN}_{1..n}$ and $\mathtt{GB}_{\text{tdyao},n,d}$ to obtain $\mathsf{G}^2_{\text{real}}$. Further inlining $\mathtt{EKEYS}_{1..n}$ and $\mathtt{DINF}_{\text{tdyao},n}$ yields game $\mathsf{G}^3_{\text{real}}$. To obtain $\mathsf{G}^4_{\text{real}}$, we 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 $\mathtt{KEYS}$ and $\mathtt{GATE}$ packages and further $\mathtt{MODGB}_{n,i}$ yields games $\mathsf{G}^5_{\text{real}}$ and $\mathsf{G}^6_{\text{real}} := \mathtt{MOD}_{n,d} \to \mathtt{SEC}^0_{n,d}(\mathtt{GB}_{\text{yao},n,d})$.

D. Proof of Claim 4

The claim follows from an inlining argument that is shown in Fig. 29 and 30. Starting from game $\mathsf{G}^1_{\text{ideal}} := \mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{\text{tdyao},n,d})$ in the first column, we first inline packages $\mathtt{EV}_{n,d}$, $\mathtt{BITS}$, and $\mathtt{SIM}_{\text{tdyao},n,d}$ to obtain $\mathsf{G}^2_{\text{ideal}}$. Game $\mathsf{G}^3_{\text{ideal}}$ is the result of moving the sampling all of wire keys except for input wire keys into the gate garbling loop, and moving the input garbling code into the loop where the input wire keys are sampled. Combining the first two loops ranging over $j = 1..n$ yields game $\mathsf{G}^4_{\text{ideal}}$. Game $\mathsf{G}^5_{\text{ideal}}$ 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 $\mathtt{KEYS}$ (game $\mathsf{G}^5_{\text{ideal}}$), $\mathtt{LEV}$ (game $\mathsf{G}^6_{\text{ideal}}$) and $\mathtt{GB}_{\text{yao},n,d}$ (game $\mathsf{G}^7_{\text{ideal}}$) to obtain $\mathsf{G}^8_{\text{ideal}} := \mathtt{MOD}_{n,d} \to \mathtt{SEC}^1_{n,d}(\mathtt{SIM}_{\text{yao},n,d})$.

E. Security Proof for Leakage $\Phi(C) = \mathsf{topo}(C)$

In this work, we restricted ourselves to proving selective security of Yao's garbling scheme with respect to a simulator that is given the circuit C to be garbled. The observant reader might have noticed though that our simulator $\mathtt{SIM}_{\text{tdyao},n,d}$ in Fig. 15 does not actually use the concrete gate operations of the circuit, but only its topology. Indeed, Yao's garbling scheme is known to provide security in this stronger sense [6]. The goal of this work was to present a simplified security proof, hence the simplification in terms of leakage function. Nevertheless, our proof can be modified as follows to achieve security with respect to leakage function $\Phi(C) = \mathsf{topo}(C)$: Garbling scheme security (Def. 13) can be defined using an ideal game $\mathtt{PRVSIM}^1_{n,d}(\mathtt{SIM}_{n,d})$ where package $\mathtt{MOD\text{-}PRVSIM}^1_{n,d}$ queries the simulator $\mathtt{SIM}_{n,d}$'s GBL oracle with $\mathsf{topo}(C)$ instead of C. That is, package $\mathtt{MOD\text{-}PRVSIM}^1_{n,d}$ removes the operation information for each gate, leaving only the circuit topology. All that remains is to adapt the proof of Claim 4 in Appendix D: Since the layer simulators $\mathtt{GB}^1_{\text{yao},n,i}$ do not use the concrete gate operation either to produce the simulated output, the gate operations can simply be removed in an additional game hop.

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

EVAL$_j(C)$

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

dinf $\leftarrow$ GETDINF

for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow$ GETA$^{\text{out}}_{j}$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

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

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  for $j = 1..n$ do
    $(\ell, r, op) \leftarrow (\boldsymbol{\ell}(j), \boldsymbol{r}(j), \boldsymbol{op}(j))$
    $\tilde{g}_j \leftarrow \bot$

    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
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  dinf$[j] \leftarrow Z_{d,j}$
for $j = 1..n$ do
  $\tilde{x}[j] \leftarrow S_{0,j}(0)$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^{3}_{\text{ideal}}$
GARBLE$(C, x)$
assert $\tilde{C} = \bot$
assert width$(C) = n$
assert depth$(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$

for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  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
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  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
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  dinf$[j] \leftarrow Z_{d,j}$

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

Figure 29: Proof of Claim 4.

```
$\mathsf{G}^4_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\text{width}(C) = n$
assert $\text{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
  $\texttt{flag}_{0,j} \leftarrow 1$
  $S_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $S_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$
  $\tilde{x}[j] \leftarrow S_{0,j}(0)$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  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
  $Z_{d,j}(z_{d,j}) \leftarrow S_{d,j}(0)$
  $Z_{d,j}(1 - z_{d,j}) \leftarrow S_{d,j}(1)$
  $\text{dinf}[j] \leftarrow Z_{d,j}$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

```
$\mathsf{G}^5_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\text{width}(C) = n$
assert $\text{depth}(C) = d$
for $j = 1..n$ do
  assert $z_{0,j} = \bot$
  $z_{0,j} \leftarrow x_j$
  $\texttt{flag}_{0,j} \leftarrow 1$
  $Z_{0,j}(0) \leftarrow\$ \{0,1\}^\lambda$
  $Z_{0,j}(1) \leftarrow\$ \{0,1\}^\lambda$
  $\tilde{x}[j] \leftarrow Z(z_{0,j})$
for $i = 1..d$ do
  $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{op}) \leftarrow C[i]$
  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
  $\text{dinf}[j] \leftarrow Z_{d,j}$
return $(\tilde{C}, \tilde{x}, \text{dinf})$
```

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

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

```
$\mathsf{G}^8_{\text{ideal}}$
$\mathsf{GARBLE}(C, x)$
assert $\tilde{C} = \bot$
assert $\text{width}(C) = n$
assert $\text{depth}(C) = d$
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 30: Proof of Claim 4, continued.