Adaptive Security of Practical Garbling Schemes

Zahra Jafargholi[*] Sabine Oechsner[†]

Abstract

A garbling scheme enables one to garble a circuit C and an input x in a way that $C(x)$ can be evaluated, but nothing else is revealed. Since the first construction by Yao, there have been tremendous practical efficiency improvements for selectively secure garbling schemes, where the adversary is forced to choose both input and circuit to be garbled at the same time. However, in the more realistic setting of adaptive security –where an adversary can choose the input adaptively based on the garbled circuit– not much is known about practical efficiency improvements.

In this work, we initiate the study of practical garbling schemes that are both more efficient than Yao's construction and adaptively secure. We provide insights into characteristics of these schemes and highlight the limitations of current techniques for proving adaptive security in this regime. Furthermore, we present an adaptively secure garbling scheme that garbles XOR gates with 2 and AND gates with 3 ciphertexts per gate, thus providing the first practical garbling scheme with adaptive security based on PRFs whose garbled circuit size is smaller than that of Yao's construction.

1 Introduction

Garbled circuits (GC) were introduced in 1982 by Yao [26, 27] and have in the past four decades since turned into a fundamental corner stone of cryptography, with applications such as secure multi-party computation, functional encryption and zero-knowledge protocols. A garbling scheme provides a way to *garble* a circuit C and an input x (into $\widetilde{C}, \tilde{x}$), such that the output $C(x)$ can be computed from $\widetilde{C}$ and $\tilde{x}$, but nothing else is revealed about the circuit or its input. This property of concealing all other information about C and x can be formalized in two different ways. The first flavor is selective security, where the adversary picks C and x at the same time and sends them to the challenger and gets back $\widetilde{C}, \tilde{x}$. The second and more realistic flavor is adaptive security where the adversary first chooses and sends C and gets back $\widetilde{C}$, and then gets to adaptively chose x. In both cases, there must be a simulator that only gets $C(x)$ (once x is known) and produces a simulated garbled circuit and input such that the adversary cannot tell the difference between the real and simulated results. In general, the topology of the graph is assumed to be public knowledge. In case of some optimized schemes, the circuit function is not private either, since the garbling and evaluation of XOR and AND gates are different.

We refer to the time it takes to garble the circuit as off-line complexity of the garbling scheme and the time it takes to garble the input (which is lower bounded by $|\tilde{x}|$) as on-line complexity of the scheme. The importance of efficient on-line complexity is more apparent in adaptive settings. Say a client (with dismal computational power) needs to compute a large circuit on a small (as yet unknown) input as quickly as possible. This client can take their time computing a garbled version of that large circuit and send it to a much more powerful server, and once the client knows x, can quickly garble and send it to the server and get back the output. In this case, clearly the on-line complexity must be much smaller than the time it takes to compute $C(x)$.

From a theoretical point of view, there exists already a range of adaptively secure garbling schemes ([7], [6], [15], [11], [17]) as well as generic transformation from selective to adaptive security ([7], [23]). Unfortunately,

[*]Sepior. Email: `zahra@sepior.com`. This work was done while at Aarhus University, Denmark. This work 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).

[†]Concordium Blockchain Research Center, Aarhus University, Denmark. Email: `oechsner@cs.au.dk`. This work 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) and the Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE).

most of them are prohibitively inefficient for practical applications and/or are based on strong non-standard assumptions. The only exception so far is Yao's garbled circuits (YaoGC) which was shown to be selectively secure based on the existence of one-way functions by Lindell and Pinkas [20], and adaptively secure for NC1 circuits by Jafargholi and Wichs [17] based on the same assumption. For the rest of this work, we will refer to the technique used in [17] as the JW technique. A considerable body of work has been dedicated to creating garbling schemes that improve in off-line complexity over Yao's construction (without sacrificing on-line complexity efficiency) by reducing the garbled circuit size ([24], [25], [19], [18], [28], [14]), and we will refer to this class of garbling schemes as *optimized garbling schemes*. All of these construction are proven selectively secure (based on different assumptions of various strength), yet their adaptive security remained unexamined. This is not surprising since proving adaptive security is notoriously difficult for many cryptographic primitives.

Given the importance of efficient garbling schemes in modern cryptographic protocols, it is natural to ask about the adaptive security of optimized garbling schemes. The challenge is the inherent need for equivocation of the circuit garbling to the real input. Indeed, there exist generic transformation from garbling schemes with selective security to adaptive security. As a drawback, none of the transformations are preserving both efficiency and assumptions. Existing transformations either increase the offline complexity to be proportional to the circuit size by using equivocal encryption [7], assume universal computational extractors [6], or shift the equivocation into the assumption of a programmable random oracle [7], resulting in the number of queries that need to programmed to be linear in the circuit size. That means that all approaches either drastically increase the concrete on-line efficiency or rely on assumptions that are believed to be widely unrealistic. It remains therefore a challenge to find or construct a garbling scheme that can be proved to be adaptively secure based on standard assumptions, with on-line complexity similar to YaoGC and off-line complexity smaller than YaoGC.

1.1 Our contributions

In this work, we present the following contributions:

1. We initiate the study of adaptively secure practical garbling schemes. To this extent, we examine the adaptive security of existing optimized garbling schemes and provide insights into characteristics of practical garbling schemes which can be proven adaptively secure – using currently known techniques. These characteristics provide a shortcut for analyzing adaptive security of future optimized garbling schemes constructions.

2. We construct the first adaptively secure optimized garbling scheme for NC1 circuits with garbled circuit size that is 25% to 50% smaller than that of YaoGC and same on-line complexity, assuming only the existence of PRFs.

3. We successfully use the JW technique for the first time on a garbling scheme built on hash functions (along with the point-and-permute optimization) instead of an encryption scheme (where permutation bits are not needed) and hence show that point-and-permute does not interfere with an adaptive security proof under standard assumptions. Moreover, our construction uses the three-row reduction optimization, proving that this technique is compatible with adaptive security.

We will give a brief overview over our findings here.

1.1.1 Adaptive Security of Existing Optimized Garbling Schemes

Since we are interested in optimized garbling schemes with adaptive security that improve over Yao's garbling scheme, we start by examining the adaptive security of existing optimized garbling schemes constructions. Our starting point will be YaoGC as the only known practical garbling scheme with adaptive security based on standard assumption.

YaoGC: From selective to adaptive security. We briefly revisit YaoGC and the main challenge in proving it adaptively secure. Recall that to garble a Boolean circuit in Yao's scheme, each wire is assigned two random keys encoding zero and one, respectively. In the next step, four ciphertexts are computed for each gate according to the gate function: each output wire key is encrypted under all combinations of input wire keys which evaluate to that output wire key. The selective security proof consists of a simulator and a series of hybrid arguments. In order to create this series of hybrid garbled circuits, it is crucial to know the input x, to make sure that the half-real-half-simulated garbled circuits indeed output correct and indistinguishable distributions. Even though the final simulation must succeed without knowledge of x, the hybrids will depend on this knowledge and will only gradually remove the dependency. The challenge for proving adaptive security thus is the inconvenient absence of x in the adaptive setting when the garbled circuit is supposed to be created. In fact, this lack of knowledge is the crux in any attempt of lifting the selective security proof of a garbling scheme to a proof of adaptive security. For the adaptive security proof of YaoGC, Jafargholi and Wichs [17] define a larger set of hybrid games wherein a few wire values in the circuit are guessed (since they are not known) at the beginning of the game, and the game is terminated if the guesses are not correct. For example if the selective security proof requires $O(n)$ hybrids arguments for a NC1 circuit with n gates, the adaptive security might require $O(n^2)$ hybrid arguments.

Optimized garbling schemes with selective security. The existing optimized garbling schemes with selective security achieve their efficiency improvements by correlating wire keys (within a gate and/or across gates) and thereby removing the need to communicate every wire key separately. Furthermore, this correlation creates opportunities to reduce the number of ciphertexts per gate. Various works proposed different wire key correlations such as row-reduction techniques [24],[25], FreeXOR [19], half gates [28] and the work by Gueron et al. [14] which we will refer to as the fast garbling approach. Note that the FreeXOR and half gates constructions, that is the most efficient constructions in terms of on-line complexity, assume primitives much stronger than PRFs (circular correlation-robust hash functions) and as a result have "simpler" selective security proofs where indistinguishability of real and simulated garbling can be shown without intermediate hybrids. Nevertheless, their selective security proof relies crucially on knowledge of the input x.

From selective to adaptive security. Turning now to adaptive security of existing optimized garbling schemes, we stumble over the same challenge as the one we were facing with YaoGC: The absence of the input x at the beginning of our security game. Since the challenge is the same as before, one might wonder if the same solution is applicable here. Furthermore, since some optimized garbling schemes are based on stronger assumptions and their selective security proofs are somewhat simpler, there is hope that this simplicity translates into simpler adaptive security proofs. However, once studied more closely, the stronger assumptions (e.g. encryption schemes with stronger security) prove to be of no help at all. Remember that the main challenge in proving adaptive security is the absence of x and since these stronger assumptions are still selective in nature (requiring the adversary to commit to a choice in advance), they do not simplify this particular challenge. Moreover, the solution from [17] (JW technique) is in most cases not applicable either. In fact, we argue that in order to prove an optimized garbling schemes adaptively secure using the existing techniques, the scheme needs to have the right balance of wire key correlation. On the one hand, more wire key correlation can yield an efficiency gain. On the other hand, too much correlation and the JW technique is not applicable anymore.

To see this, on a very high level, consider the distribution of wire keys in Yao's construction: independent and uniformly random. Or rather, they start as such, but as soon as the garbled circuit is created and the gate garblings published, all wire keys are correlated with the garbled circuit (except for the input wire keys to the circuit), because they appear as part of the garbled circuit inside ciphertexts. To prove (adaptive or selective) security, for every wire in the circuit, all occurrences of one of the wire keys (call it k_1) have to be removed from the garbled circuit through a sequence of hybrids. These occurrences are replaced by the other wire key (k_0). This sequence of hybrids should be defined such that each two consecutive hybrids can be shown to be indistinguishable via a reduction from the underlying assumption. If the sequence starts from using two keys on each wire and ends in using only one key in all wires, then real and simulated garbling are indistinguishable.

In [17], a pebbling game is defined to capture the requirements of a valid sequence of hybrids in order to prove adaptive security. In this game, hybrid garbling gate modes are defined as nodes without a pebble/with black pebbles/with gray pebbles. The graph state representing the real garbling is the graph of the circuit with no pebbles and the graph state representing the simulated garbling is the circuit graph with a gray pebble on each node. The rules of the pebbling game ensure that if in between two hybrid garbled circuits, an encryption of key k'_1 under key k_1 is replaced with an encryption of key k'_0 under k_1, then k_1 has to be both random and cannot appear anywhere in the hybrid garbled circuit, not even in encrypted form. This is crucial, because to be able to invoke the security of the underlying encryption scheme –to argue the change of the message cannot be detected– we have to make sure the key k_1 of the encryption follows the distribution of the security game, i.e. is independent and uniformly random.

It is now discernible that if we add more dependency between the wire keys by not only creating ciphertexts of them, but also by correlating them to each other from the beginning, the pebbling rules might have to be changed to make sure that during reduction, k_1 is completely independent of the garbled circuit. Indeed this revision of rules might not even be possible if the keys are actually derived directly from other keys in the circuit. Even if this revision is possible, the resulting pebbling game might be so costly that makes this approach to proving security futile. If we can find a way to correlate wire keys that preserve the JW pebbling rules (or at least does not make them too involved), then we can use the JW technique to prove adaptive security for the new garbling scheme.

Unfortunately, it turns out that the pebbling rules cannot be adapted for FreeXOR and garbling schemes using the FreeXOR technique. For Two-Row Reduction and the garbling scheme by Gueron et al., the pebbling game could in principle be modified, but the resulting proof strategy would be prohibitively expensive. The only existing optimized garbling scheme that can be proven secure using the JW technique is YaoGC combined with Three-Row Reduction.

1.1.2 Our New Construction

With these insights, we move on to constructing a new garbling scheme that can in fact be proven adaptively secure. The scheme is similar in spirit to that of Gueron et al. [14] and adaptive security relies only on the existence of PRFs. XOR gates can be garbled with two ciphertexts and AND gates with three. The key insight is that the garbling of XOR gates in Gueron et al.'s garbling scheme almost achieves adaptive security. Consider therefore the garbling of XOR gates. As observed by Kolesnikov and Schneider [19], an XOR gate can be evaluated without any ciphertext if both the input wire key pairs as well as the output wire key pair are correlated by the same offset. As a consequence, the output wire keys are determined deterministically by the input wire keys. This approach requires however to either prove selective security in the random oracle model or assume the existence of circular-correlation robust hash functions [10] or the corresponding notion of related-key key-dependent message security for encryption schemes [4]. In order to enable a reduction to PRFs, the garbling scheme by Gueron et al. first "translates" the input wire keys (evaluates a PRF on them) to hide the dependency on the predecessor gates and enable a reduction to PRFs. The construction manages to still keep the number of ciphertexts for a garbled XOR gate at just one by computing the gate offset deterministically from the input wire keys on one gate and forcing the output wire key pair to have the same offset. The remaining ciphertext encrypts the gate-specific offset. However, the input wire keys still determine the output wire keys uniquely.

By choosing the offset in XOR gates uniformly at random and using the strongest known row reduction technique compatible with adaptive security proofs, that is three-row reduction, for AND gates, we achieve the first garbling scheme that improves in off-line complexity over Yao's construction and has adaptive security based on weak standard assumptions. Our construction uses the point-and-permute technique [24] and is therefore also the first efficient adaptively secure garbling scheme using point-and-permute, proving that point-and-permute is compatible with adaptive security from standard assumptions.

1.2 Applications of Adaptively Secure Garbling Schemes

A garbling scheme needs to provide adaptive security in all settings where the input is not yet determined at the time of circuit garbling. Examples include one-time programs [13], verifiable computation [12] and succinct adaptively secure functional encryption [1], [3].

On the more practical side, one of the main applications of garbling schemes are secure two-party computation (2PC) protocols. This requires the garbling scheme to be projective, that is the circuit garbling produces two tokens for each bit position of the input as input encoding information. The input is then garbled by choosing one of the tokens per input bit. The two parties in the 2PC protocol each take on a role: One party acts as garbler and the other one as evaluator. The garbling party garbles the circuit C to obtain circuit garbling $\widetilde{C}$ as well as some input encoding information. The garbler then encodes their own input x_2 directly and sends the input garbling $\tilde{x_2}$ together with the circuit garbling $\widetilde{C}$ to the evaluator. To garble the evaluator's input x_1, the parties run an oblivious transfer protocol. The garbler inputs the input encoding information and the evaluator their input. The evaluator will then learn the corresponding input garbling $\tilde{x_1}$. Together with $\widetilde{C}$ and $\tilde{x_2}$, the evaluator can now evaluate the garbled circuit on the garbled inputs to learn the result y of the computation. Finally, the evaluator sends y to the garbler.

In many cases, garbling the circuit C is actually expensive, and therefore various works studied secure computation in the preprocessing model. In the context of two-party computation, this approach was first considered by Lindell and Riva [22], [23]. Protocols in the preprocessing model are split into two phases: An input-independent offline phase that can be executed in advance and an on-line phase where the parties have inputs. If all expensive operations can be performed in the offline phase, then the on-line phase is extremely efficient. This division of protocols is only possible if the garbling scheme has adaptive security.

1.3 Related Work

Garbling schemes as an abstraction of the cryptographic technique underlying the garbled circuits approach were introduced by Bellare, Hoang and Rogaway [8]. The subsequent work by the same authors [7] complemented the static security definitions of [8] with adaptive notions and provided the first garbling schemes satisfying these adaptive notions.

The work of Bellare, Hoang and Rogaway [7] presented a general transformation from a selectively secure garbling scheme to an adaptively secure one (in their terminology from prv to prv1). The transformation adds a layer of one-time pad encryption to the circuit garbling and reveals the key as part of the input garbling, thus yielding an on-line complexity that is proportional to the circuit size. The alternative transformation from the same work avoids sending the one-time pad key by resorting to the programmable random oracle and instead programs all queries needed to evaluate the circuit to match the desired input. Subsequently, Bellare, Hoang and Keelveedhi [6] presented a transformation from selective to adaptive security in the same spirit that replaces the random oracle by a universal computational extractor. These results are applicable to any selectively secure garbling scheme, but either incur an overhead to the on-line complexity that is proportional to the circuit size or are based on strong non-standard assumptions.

Another line of work tries to reduce the overhead and assumptions associated with adaptive security proofs of specific garbling schemes from the literature, starting with the work of Hemenway et al. [15]. They showed adaptive security for a modified version of YaoGC where a layer of somewhere equivocal encryption is added to the circuit garbling. Intuitively, somewhere equivocal encryption allows to specify positions in the message that can later be equivocated. The proof proceeds by carefully crafting a sequence of hybrid games. Only a small number of gates needs to be equivocated in each hybrid game which the somewhere equivocal encryption layer enables. The sequence of hybrids and hence the security loss is determined by the pebble complexity of a pebbling game on the circuit topology. As a result, Hemenway et al. could show that the modified Yao scheme has on-line complexity proportional to the width of the circuit or alternatively proportional to the sum of depth, input size and output size of the circuit. As Hemenway et al. [15] showed, somewhere equivocal encryption can be constructed from one-way functions. However, the construction is far from practically efficient even though it provides nice asymptotic guarantees. The approach by Hemenway et al. was refined by Garg and Srinivasan [11] who modify Yao's construction with an updatable laconic OT [9] to achieve better asymptotic on-line complexity. Ananth and Lombardi [2] generalized some of the results from the two previously mentioned works by presenting a transformation relying on an underlying local simulation property, meaning that the garbling scheme admits local simulation for the gates where the adversary can see the input-*in*dependent components of the garbled circuit before choosing the input. Any garbling scheme exhibiting the local simulation property can be transformed into an adaptively secure by adding a layer of somewhere equivocal encryption. However, Ananth and Lombardi do not provide any additional instantiations of their result.

Apart from equivocation, adaptive security can be proved using complexity leveraging by simply guessing the input, resulting in a security loss that is exponential in the input size, and assuming that the security parameter is even larger to cope with the security loss. In fact, Jafargholi and Wichs [17] showed that the approach of Hemenway et al. can be combined with guessing parts of the input (instead of equivocation). As a result, they could show adaptive security of the unmodified Yao construction for NC1 circuits or under the assumption of exponentially secure one-way function.

1.4 Outline

We start by providing the necessary notation and background on garbling schemes in Section 2. Then, we revisit the JW technique in Section 3. We also derive characteristics of adaptively secure garbling schemes and examine the applicability of the JW technique to existing optimized garbling schemes. Section 4 presents our new garbling scheme which is proven to be adaptively secure in Section 5.

2 Preliminaries

2.1 General Notation

We will use $[q]$ as a shorthand for the set $\{1, \dots, q\}$. For a bitstring $s = s_1 \dots s_k$ of arbitrary length k, we will denote by s_{abl} the substring $s_1 \dots s_{k-1}$ and by s_{lsb} the last bit s_k.

Definition 1 ([17])**.** Two distributions X and Y are (T, ε)-indistinguishable, denoted $\mathbf{D}_T[X, Y] = \varepsilon$, if for any probabilistic algorithm $\mathcal{A}$ running in time T,

$$|\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]| \le \varepsilon.$$

For two games $\textsc{Game}$ and $\textsc{Game}'$, we say that they are $(T(\lambda), \varepsilon(\lambda))$-indistinguishable, $\mathbf{D}_{T(\lambda)}\left[\textsc{Game}, \textsc{Game}'\right] = \varepsilon(\lambda)$, if for any adversary $\mathcal{A}$ running in time $T(\lambda)$,

$$|\Pr[\textsc{Game}_{\mathcal{A}} = 1] - \Pr[\textsc{Game}'_{\mathcal{A}} = 1]| \le \varepsilon(\lambda).$$

Let games $\textsc{Game}(\lambda)$ and $\textsc{Game}'(\lambda)$ be games parameterized by the security parameter λ. If for any polynomial function $T(\lambda)$, there exists a negligible function $\varepsilon(\lambda)$ such that for all λ, $\mathbf{D}_{T(\lambda)}\left[\textsc{Game}(\lambda), \textsc{Game}'(\lambda)\right] \le \varepsilon(\lambda)$, we say that the two games are computationally indistinguishable, denoted $\textsc{Game}(\lambda) \overset{\text{comp}}{\approx} \textsc{Game}'(\lambda)$.

2.2 Garbling Schemes

The bulk of this section defining what garbled circuits are and presenting Yao's construction, is taken verbatim from [15]. There are many variants of such definitions in the literature, and we refer the reader to [8] for a comprehensive treatment.

Definition 2. A garbling scheme is a tuple of PPT algorithms $\mathsf{GC} = (\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$ such that:

- $(\widetilde{C}, k) \overset{\$}{\leftarrow} \mathsf{GCircuit}(1^\lambda, C)$: takes as input a security parameter λ, a circuit $C : \{0,1\}^n \to \{0,1\}^m$, and outputs the *garbled circuit* $\widetilde{C}$, and *key* k.
- $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$: takes as input, $x \in \{0,1\}^n$, and key k and outputs $\tilde{x}$.
- $y = \mathsf{Eval}(\widetilde{C}, \tilde{x})$: given a garbled circuit $\widetilde{C}$ and a garbled input $\tilde{x}$ output $y \in \{0,1\}^m$.

Correctness There is a negligible function ν such that for any $\lambda \in \mathbb{N}$, any circuit C and input x it holds that $\Pr[C(x) = \mathsf{Eval}(\widetilde{C}, \tilde{x})] = 1 - \nu(\lambda)$, where $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$, $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$.

Adaptive Security. There exists a PPT simulator $\mathsf{Sim} = (\mathsf{SimC}, \mathsf{SimIn})$ such that, for any PPT adversary $\mathcal{A}$, there exists a negligible function ε such that:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 1) = 1]\right| \le \varepsilon(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, b)$ is defined as follows:

1. The adversary $\mathcal{A}$ specifies C and gets $\widetilde{C}$ where $\widetilde{C}$ is created as follows:

- if $b = 0$: $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$,
- if $b = 1$: $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SimC}(1^\lambda, \Phi(C))$,

2. The adversary $\mathcal{A}$ specifies x and gets $\tilde{x}$ created as follows:

   - if $b = 0$, $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$,
   - if $b = 1$, $\tilde{x} \leftarrow \mathsf{SimIn}(C(x), \mathsf{state})$.

3. Finally, the adversary outputs a bit b', which is the output of the experiment.

In other words, we say GC is **adaptively secure** if

$$\mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Sim}}(\lambda, 0), \mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Sim}}(\lambda, 1)\right] = \varepsilon(\lambda).$$

Online complexity. The time it takes to garble an input x (i.e. time complexity of GInput) is the *online complexity* of the scheme. Clearly the online complexity of the scheme gives a bound on the size of the garbled input $\tilde{x}$.

Projective Scheme. We say a garbling scheme is *projective* if each bit of the garbled input $\tilde{x}$ only depends on one bit of the actual input x. In other words, each bit of the input, is garbled independently of other bits of the input. Projective schemes are essential for two-party computation where the garbled input is transmitted using an oblivious transfer (OT) protocol. Our constructions will be projective.

Leakage functions. In the optimized garbling schemes we consider in this work (including the one we propose), the way a single gate is garbled differs significantly depending on the gate type. Hence, we cannot hope to hide anything about the circuit C to be garbled from an outsider, and therefore, the leakage function applied to C to compute SimC's input is the identity function in this case.

2.2.1 Yao's Scheme

For each wire w in the circuit, we pick two keys k^0_w, k^1_w for a symmetric-key encryption scheme. For each gate in the circuit computing a function $g : \{0,1\}^2 \to \{0,1\}$ and having input wires a, b and output wire c we create a *garbled gate* consisting of 4 randomly ordered ciphertexts created as:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^0_b}(k_c^{g(0,0)})) \qquad & c_{1,0} &= \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^0_b}(k_c^{g(1,0)})), \\ c_{0,1} &= \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^1_b}(k_c^{g(0,1)})) \qquad & c_{1,1} &= \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^1_b}(k_c^{g(1,1)})) \end{aligned} \tag{1}$$

where $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure encryption scheme. The garbled circuit $\widetilde{C}$ consists of all of the gabled gates, along with an *output map*

$$\{k^0_w \to 0, k^1_w \to 1\}$$

which maps the keys to the bits they represent for each output wire w. To garble an n-bit value $x = x_1 x_2 \cdots x_n$, the garbled input $\tilde{x}$ consists of the keys $k^{x_i}_{w_i}$ for the n input wires w_i.

To evaluate the garbled circuit on the garbled input, it's possible to decrypt exactly one ciphertext in each garbled gate and get the key $k^{v(w)}_w$ corresponding to the bit $v(w)$ going over the wire w during the computation $C(x)$. Once the keys for the output wires are computed, it's possible to recover the actual output bits by looking them up in the output map.

2.3 Pseudorandom Functions

Definition 3. A *pseudorandom function* (PRF) is an efficiently computable family of functions $\mathcal{F} = \{F_n\}_{n\in\mathbb{N}}$ such that for every function $F_n \colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$, no efficient PPT algorithm $\mathcal{D}$ can distinguish the outputs of a randomly chosen function from those of a function that outputs uniformly random values.

We will write $F_k(x)$ where $k \in \{0,1\}^n$ is the key and $x \in \{0,1\}^m$ is the input of the function.

Experiment $\mathsf{Exp}^{2PRF}_{\mathcal{F},\mathcal{A}}(n,\sigma)$

1. Choose random keys $k_1, k_2 \leftarrow \{0,1\}^n$ for the pseudorandom function, and choose two truly random functions f^1, f^2.
   If $\sigma = 0$, set $(\mathcal{O}^{(1)}, \mathcal{O}^{(2)}, \mathcal{O}^{(3)}, \mathcal{O}^{(4)}) = (\mathcal{F}_{k_1}, \mathcal{F}_{k_1}, \mathcal{F}_{k_2}, \mathcal{F}_{k_2})$.
   Else, set $(\mathcal{O}^{(1)}, \mathcal{O}^{(2)}, \mathcal{O}^{(3)}, \mathcal{O}^{(4)}) = (f^1, \mathcal{F}_{k_1}, f^2, \mathcal{F}_{k_2})$.
2. The adversary $\mathcal{A}$ is invoked upon input 1^n.
3. When $\mathcal{A}$ makes a query (j, x) to its oracles with $j \in \{1,2,3,4\}$ and $x \in \{0,1\}^{n+1}$, answer as follows:
   - If $j \in \{1,2\}$ and x was already queried to $\{1,2\}\backslash\{j\}$, return $\perp$.
   - If $j \in \{3,4\}$ and x was already queried to $\{3,4\}\backslash\{j\}$, return $\perp$.
   - Otherwise, return $\mathcal{O}^{(j)}(x)$.
4. $\mathcal{A}$ outputs a bit σ', and this is the output of the experiment.

Figure 1: The 2PRF experiment from [14].

Definition 4. Let $\mathcal{F} = \{F_n\}_{n\in\mathbb{N}}$ be an efficient family of functions where for every n, $F_n\colon \{0,1\}^n \times \{0,1\}^{n+1} \to \{0,1\}^{n+1}$. Family $\mathcal{F}$ is a $(T(\lambda), \varepsilon(\lambda))$-secure 2PRF if for every n and every probabilistic adversary $\mathcal{A}$, running in time T

$$\mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{2PRF}_{\mathcal{F},\mathcal{A}}(n,1), \mathsf{Exp}^{2PRF}_{\mathcal{F},\mathcal{A}}(n,0)\right] \leq \varepsilon(n).$$

Lemma 1 ([14])**.** If $\mathcal{F}$ is a family of pseudorandom functions, then it is a 2PRF.

3 Adaptive and Practical Garbling Schemes: Background and A Way Forward

This section revisits the only adaptive security proof of a practical garbling scheme currently known based on standard assumptions. We start by a brief overview of YaoGC and the challenges in proving it adaptively secure. Section 3.2 then revisits the JW technique which is due to Jafargholi and Wichs [17], with a focus on how adaptive security is achieved. Following this, in Section 3.3 we identify properties of a garbling scheme that can be proven adaptively secure using the JW technique and examine well-known optimized garbling schemes with respect to these findings (Section 3.4).

3.1 Yao's Scheme and The Challenge of Adaptive Security ([17])

To understand the difficulty with analyzing the adaptive security of optimized garbling schemes, we need to start from selective security of Yao's scheme (see Section 2.2.1). This subsection captures the basic information on the construction of YaoGC, the selective security proof and the difficulty proving adaptive security quite succinctly, and for this reason appears in multiple works on this topic ([15], [17], [16]) and we gratefully make use of it in this work as well.

Selective Security Simulator. To prove the selective security of Yao's scheme, we need to define a simulator that gets the output $y = y_1 y_2 \cdots y_m = C(x)$ and must produce $\widetilde{C}$, $\tilde{x}$. The simulator picks random keys k^0_w, k^1_w for each wire w just like the real scheme, but it creates the garbled gates as follows:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^0_b}(k^0_c)) \qquad & c_{1,0} &= \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^0_b}(k^0_c)), \\ c_{0,1} &= \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^1_b}(k^0_c)) \qquad & c_{1,1} &= \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^1_b}(k^0_c)) \end{aligned} \tag{2}$$

where all four ciphertexts encrypt the same key k_c^0. It creates the output mapping $\{k_w^0 \to y_w, k_w^1 \to 1 - y_w\}$ by "programming it" so that the key k_w^0 corresponds to the correct output bit y_w for each output wire w. This defines the simulated garbled circuit $\widetilde{C}$. To create the simulated garbled input $\tilde{x}$ the simulator simply gives out the keys k_w^0 for each input wire w. Note that, when evaluating the simulated garbled circuit on the simulated garbled input, the adversary only sees the keys k_w^0 for every wire w.

Selective Security Hybrids. To prove indistinguishability between the real world and the simulation, there is a series of carefully defined hybrid games that switch the distribution of one garbled gate at a time. Unfortunately, we cannot directly switch a gate from the real distribution (1) to the simulated one (2) and therefore must introduce an intermediate distribution (3) as below:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})) \qquad & c_{1,0} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})), \\ c_{0,1} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) \qquad & c_{1,1} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) \end{aligned} \tag{3}$$

where $v(c)$ is the correct *value* of the bit going over the wire c during the computation of $C(x)$.

Let us give names to the three modes for creating garbled gates that we defined above: (1) is called RealGate mode, (2) is called SimGate mode, and (3) is called InputDepSimGate mode, since the way that it is defined depends adaptively on the choice of the input x.

We can switch a gate from RealGate to InputDepSimGate mode if we are in the input level or the *predecessor* gates are in InputDepSimGate mode. The latter follows by CPA security of encryption. In particular, we are *not* changing the value contained in ciphertext $c_{v(a),v(b)}$ encrypted under the keys $k_a^{v(a)}, k_b^{v(b)}$ that the adversary obtains during evaluation, but we *can* change the values contained in all of the other ciphertexts since the keys $k_a^{1-v(a)}, k_b^{1-v(b)}$ do not appear anywhere inside the predecessor garbled gates as long as they are already in InputDepSimGate mode.

We can also switch a gate from InputDepSimGate to SimGate mode if we are at the output level or the *successor* gates are in InputDepSimGate or SimGate mode. This is actually an information theoretic step; since the keys k_c^0, k_c^1 are used completely symmetrically in the successor gates there is no difference between always encrypting $k_c^{v(c)}$ as in InputDepSimGate mode or encrypting k_c^0 as in SimGate. This allows us to first switch every gate from RealGate to InputDepSimGate mode and then from InputDepSimGate to SimGate, proving the selective security of Yao's construction.

Challenges in Achieving Adaptive Security. There are two issues in using the above strategy in the adaptive setting: an immediate but easy to fix problem and a more subtle but difficult to overcome problem.

The first immediate issue is that the selective simulator needs to know the output $y = C(x)$ to create the garbled circuit $\widetilde{C}$ and in particular to program the output mapping $\{k_w^0 \to y_w, k_w^1 \to 1 - y_w\}$ for the output wires w. However, the adaptive simulator does not get the output y until *after* it creates the garbled circuit $\widetilde{C}$. Therefore, we cannot (even syntactically) use the selective security simulator in the adaptive setting. This issue turns out to be easy to fix by modifying the construction to send the output-mapping as part of the garbled input $\tilde{x}$ in the on-line phase[1], rather than as part of the garbled circuit $\widetilde{C}$ in the off-line phase. This modification raises on-line complexity to also being linear in the output size of the circuit, which we know to be necessary by the lower bound of [5] [2]. We refer to this modification as YaoGC construction in the adaptive setting. With this modification, the adaptive simulator can program the output mapping after it learns the output $y = C(x)$ in the on-line phase and therefore we get a syntactically meaningful simulation strategy in the adaptive setting.

The second problem is where the true difficulty lies. Although we have a syntactically meaningful simulation strategy, the previous proof of indistinguishability of the real world and the simulation completely breaks down in the adaptive setting. In particular InputDepSimGate mode as specified in equation (3) is syntactically undefined in the adaptive setting. Recall that in this mode the garbled gate is created in a

[1]In considering the adaptive security of any selectively secure garbling scheme, we in fact consider a slightly modified variation of that scheme, where the output map is part of the garbled input.

[2][16] shows how to circumvent this lower-bound by considering indistinguishability-based security instead of simulation based security. However since the size of the garbled circuit increases by a factor of 2 (compared to YaoGC), it is not in the scope of this current work

way that depends on the input x, but in the adaptive setting the input x is chosen adaptively after the garbled circuit is created! Therefore, *although we have a syntactically meaningful simulation strategy for the adaptive setting, we do not have any syntactically meaningful sequence of intermediate hybrids to prove indistinguishability between the real world and the simulated world.*

3.2 A Tool and a Trick: Adaptive Security from the JW Technique

To understand how Jafargholi and Wichs overcome the challenges in proving adaptive security of YaoGC, we revisit their strategy here. The first part of their strategy is to introduce **a tool**: pebbling games. Think of the circuit as a graph where the gates are nodes and the input and output wires of each gate are (directed) ingoing and outgoing edges of that node. Then let predecessor and successors of a node be naturally nodes with edges outgoing to and ingoing from that node. Finally translate "gate in RealGate/InputDepSimGate/SimGate mode" to "node with no pebble/a black pebble/a gray pebble". This way, we can describe the sequence of hybrids in the selective security proof as series of pebbling steps that start with a graph with no pebbles and end up with a graph with a gray pebble on each node, following the 2 rules below:

1. A black pebble can be added to or removed from a node if its predecessors already have a black pebble, or if it has no predecessor.

2. A black pebble on a node can replaced with gray pebble, if its successors already have a pebble, or it has no successors, and vice versa.

Using pebbling games as a way to describe – and maybe more importantly, to think about the sequence of hybrids, simplifies matters greatly. However this may not be apparent looking at the selective security, since there, first every node gets a black pebble starting from the roots (nodes with no ingoing edges) and moving toward the leaves. And then every black pebble is replace with a gray pebble (say, in the reverse order). After all there was no restriction on the number of pebbles used and no incentive to minimize them. Now let us move to the adaptive security proof, here the objective is the same, except that the InputDepSimGate mode is not well-defined because the value on the output wire of the gate is not known when the garbled circuit is produced. That brings us to **the trick**: guess the unknown output wire value of the gates that are in InputDepSimGate mode. By adding this guessing step at the beginning of the hybrid games and aborting if any of the guesses are wrong (once the input is fixed) it is possible to once more define gates in InputDepSimGate mode. This almost solves our problem, except now we need to make sure the probability of guessing correctly is reasonably high, in other words the number of guessed values are as small as possible, and this in turn means the number of gates in InputDepSimGate mode should be as small as possible. [17] shows that if there is a pebbling strategy to pebble the graph of circuit C in γ steps and using t black pebbles then YaoGC is $2^t\gamma\varepsilon$ adaptively secure if encryption used in the construction of the garbled circuit is ε IND-CPA secure.

3.3 Characteristics of Adaptively Secure Garbling Schemes

We are now ready to explain our observations about the adaptive security proof of YaoGC by Jafargholi and Wichs [17] and derive necessary characteristics of optimized garbling schemes with adaptive security.

Jafargholi and Wichs manage to reduce the security loss by limiting the number of gates in InputDepSimGate mode and hence the guessing needed in each hybrid. To this extent, the proof needs to be able to change the gate mode of any gate g from InputDepSimGate back to RealGate i.e. from input-dependent to input-*in*dependent garbling, leading to the case of gate g' in InputDepSimGate mode with predecessor g in RealGate mode. (For ease of explanation, assume g is a gate at the input layer.) To argue indistinguishability of two garbled circuits that only differ in gate g, one now has to reduce to a computational assumption by embedding the challenger's key as one of the input wire keys of g. Note that this case does not need to be considered in a proof of selective security, as limiting the number of gates in InputDepSimGate mode is not an issue. Conveniently, YaoGC has the property that each input wire key is chosen uniformly at random which happens to be the challenger's key distribution in the IND-CPA security game. The key point here is that changes to the distribution of gate g do not automatically change the garbling distribution of its successor g'.

More generally, if a selectively secure garbling scheme that is a variant of Yao's construction has either of the following properties the JW technique cannot be used to prove adaptive security:

Undesirable Property 1: The selective security proof is a global argument reasoning about all gates at once.

If the selective security proof is a global argument, (usually comprising of a constant number of hybrid arguments) it is not possible to gradually introduce and remove input dependencies during the security proof. Instead, the whole argument requires knowledge of the entire input. This might be rather counter-intuitive at first glance, since these constructions are often based on much stronger assumptions and might give one the impression that the stronger assumption might help with proof of adaptive security. However since these stronger assumptions remain selective in flavor, they do not help with proving adaptive security. This is typically the case when wire key pairs are correlated across all gates.

Undesirable Property 2: The input wire keys uniquely determine both output wire keys.

When both output wire keys are uniquely determined from the input wire keys, the pebbling game must be changed to reflect this additional dependency, because the pebbling rules are designed to make sure when a gate g is changed from one mode to another, the necessary underlying reduction (from a computational assumption) is possible. And this is possible if one wire key of each input to the gate g is completely independent and random, i.e. they do not appear inside any ciphertext in the garbled circuit. But now if we do not update the pebbling rules to reflect the additional dependency, the following situation can occur: Even though key k (an input wire to g') is not encrypted in any gate, the keys that compute k are encrypted in some other gate (say predecessor g which is now in $\mathsf{RealGate}$ mode). This makes it impossible to use k for the reduction (to replace a challenger's key), since its distribution is no longer independent of the garbled circuit. However changing the pebbling rules to solve this problem will prove to be futile too. The new rule would require a black pebble to be removed as soon as one of its predecessors does not have a pebble. Therefore the savings gained by the JW technique are lost.

3.4 How the Existing Schemes Do with Regards to These Characteristics

We will now examine existing selectively secure constructions and optimizations that are practically efficient with respect to the findings from Section 3.3.

3.4.1 Row Reduction Techniques

The Three-Row Reduction technique (also referred to as 4-3 GRR) by Naor, Pinkas and Sumner [24] sets the first ciphertext in a garbled gate to be the all-zero string (which in turn determines one of the output wire keys), and therefore it is not necessary to include that ciphertext in the garbled gate. This means that one of the output wire keys is determined by the input wire keys while the other output wire key is chosen uniformly at random. Indeed, our new construction presented in Section 4 makes use of this idea.

Pinkas et al.'s Two-Row Reduction technique [25] (or 4-2 GRR) uses polynomial interpolation to further reduce the number of ciphertexts for a gate garbling to just two. In particular, both output wire keys are uniquely determined by the input wire keys. Therefore, a garbling scheme that uses the 4-2 GRR reduction cannot be proven adaptively secure using the JW technique.

Gueron et al. [14] propose another method for garbling AND gate with two ciphertexts only (referred to as Fast 4-2 GRR). The idea is to employ simple XOR operations in a clever way to stretch two ciphertext masks to be used in three different ways, and to combine this with 4-3 GRR to get a new two-row reduction. Once more, the input wire keys determine both output wire keys uniquely and hence the JW adaptive security proof fails.

3.4.2 FreeXOR, FleXor and Half gates

The FreeXOR optimization by Kolesnikov and Schneider [19] is one of the most successful optimization techniques for garbling schemes. It allows to garble XOR gates "for free", i.e. without sending any ciphertext, while AND gates require three ciphertexts using 4-3 GRR. The idea for garbling XOR gates is to correlate wire keys by a global *shift* or *offset* Δ that is only known to the circuit garbler. For each wire a, the two keys are some random k_a^0 and the derived $k_a^1 = k_a^0 \oplus \Delta$. When garbling an XOR gate, the garbler uses the 4 input wire keys $k_a^0, k_a^1, k_b^0, k_b^1$ to (implicitly) define output wire keys

$$k_c^0 = k_a^0 \oplus k_b^0 = k_a^1 \oplus k_b^1 \text{ and } k_c^1 = k_a^0 \oplus k_b^0 \oplus \Delta = k_a^0 \oplus k_b^1 = k_a^1 \oplus k_b^0$$

This way, the garbled XOR gate can be evaluated without any ciphertext by computing the output wire key as

$$k_c^{b_a \oplus b_b} = k_a^{b_a} \oplus k_b^{b_b} = k_c^0 \oplus (b_a \oplus b_b)\Delta$$

In summary, all wire key pairs in the garbled circuit are correlated by the same offset. The selective security of FreeXOR is based on strong assumptions such as circular-correlation robust hash functions [10] or RK-KDM encryption scheme [4] and fall into the category of proofs that satisfy Undesirable Property 1. Moreover, in XOR gates the input wire keys uniquely determine both output wire keys (satisfying Undesirable Property 2). Thus the FreeXOR construction cannot be proven adaptively secure using the JW technique.

Since the FreeXOR allows to garble XOR gate for free but is incompatible with 4-2 GRR, the FleXor technique by Kolesnikov et al. [18] gives a compromise between them, i.e. using FreeXOR for as many XOR gates as possible and likewise two-row reduction for AND. Assume for a start that each gate has its own gate offset. The challenge is then to minimize the garbled circuit size with respect to *all* gates and hence to find out which offset to use for which wire. This is important because two-row reduction requires that two of the ciphertexts can be chosen freely. To this extent, Kolesnikov et al. group wires according to equivalence classes, i.e. wires that can use the same offset without violating circularity restrictions.

The number of ciphertexts per AND gate was further reduced to two in the Half Gates scheme by Zahur et al. [28]. The scheme retains the FreeXOR procedure for XOR gate garbling and in fact crucially relies on it for garbling AND gates with two ciphertexts. The half gates construction observes that each AND gate can be split into two "half" AND gates where in each half gate, one of the parties knows one of the inputs in the clear. Moreover, each half gate can be garbled with one ciphertext, and the two half gates are combined using a (free) XOR operation. As both FleXor and half gates rely on the FreeXOR approach, they cannot be proven to be adaptively secure using the JW technique.

3.4.3 Fast Garbling

Gueron et al. [14] propose another approach to reducing garbled circuit size using shared wire offsets. The work is motivated by the question of reducing the assumptions required to prove security (and by the wish to leverage special-purpose hardware for AES). Instead of ensuring that subsequent gates share common offsets, their work shows a garbling method that correlates the keys of *all* wires within an XOR gate by the same offset, but in a way that makes this offset independent of the offsets in the surrounding gates. This way they circumvent both the "circularity" difficulties present in FreeXOR and the "correlation-robustness" requirement of FleXor.

For simplicity, consider the case where the permutation bits of both input wires are 0. AND gates are garbled with three ciphertexts using 4-3 GRR, or with two ciphertexts following Fast 4-2 GRR. To garble an XOR gate g with input wires a and b and output wire c, the garbler computes the keys to be used in the gate as follows:

1. Randomize ("translate") wire keys k_a^0, k_a^1 to $\tilde{k}_a^0$, $\tilde{k}_a^1$ using the PRF to ensure that keys are independent from those of other gates.
2. Compute the gate offset $\Delta = \tilde{k}_a^0 \oplus \tilde{k}_a^1$.
3. Translate k_b^0 to $\tilde{k}_b^0$ using the PRF and set the corresponding translated input wire key $\tilde{k}_b^1 = \tilde{k}_b^0 \oplus \Delta$.
4. Compute output wire keys as $k_c^0 = \tilde{k}_a^0 \oplus \tilde{k}_b^0$ and $k_c^1 = k_c^0 \oplus \Delta$.

Finally, the only ciphertext of the XOR gate contains $\tilde{k}_b^1$. However, the offset of a gate and therefore also the two output wire keys are still computed from the input wire keys (Undesirable Property 2) and so again, we cannot use the JW technique to prove adaptive security.

4 Construction

In this section, we are going to present our construction of a garbling scheme with adaptive security based on the existence of pseudorandom functions. It allows to garble XOR gates with 2 ciphertexts and AND gates with 3. The garbling scheme is similar in spirit to [14] for XOR gates. The main difference is that in XOR gates, a fresh gate offset is chosen at random instead of being derived deterministically from the input wire keys. As a consequence, XOR gates are garbled using 2 ciphertexts (instead of one). While this seems like a step back at first, we will demonstrate in the Section 5 how this modification facilitates the proof of adaptive security. AND gates are garbled following the three-row reduction technique by Naor, Pinkas and Sumner [24], demonstrating for the first time that three-row reduction is compatible with adaptive security.

We start by giving some prerequisites, followed by a description of how to garble an XOR gate, and will then provide a full specification of our garbling scheme. For simplicity, we assume that the circuit to be garbled consists solely of AND and XOR gates.

Point-and-permute. While not resulting in a reduction of garbled circuit size, this optimization is crucial for the following constructions. In the original construction of Yao's garbled circuits, each garbled gate consists of four ciphertexts. In order to hide to which input wire bit each ciphertext corresponds, the ciphertexts are randomly permuted. Evaluating the gate requires now attempting to decrypt all four, while only one of them can be successfully decrypted. To reduce the evaluation time, Naor et al. [24] proposed the *point-and-permute* technique. It assigns a random permutation bit to each wire. The permutation bits on the two input wires of a gate allow the evaluator to determine which ciphertext to decrypt, thus reducing the number of ciphertexts to be decrypted per gate to 1. We will follow the convention of [14] and refer to this bit as *permutation bit* during the garbling and as *signal bit* when evaluating.

Primitives. Our garbling scheme uses on a pseudorandom function $\mathcal{F}$ as defined in Def. 3 to compute and evaluate garbled gates. More specifically, the scheme uses the method of [21] as a form of double encryption: For message m and two keys k_1, k_2, the ciphertext is computed as $\mathcal{F}_{k_1}(r_1) \oplus \mathcal{F}_{k_2}(r_2) \oplus m$ for fresh nonces r_1, r_2.

```
$\mathbf{PAG}\mathsf{Xor}\left(g, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b\right)$    //HybXor0

  1. Set $\pi_c := \pi_a \oplus \pi_b$.
  2. $\Delta \leftarrow \{0,1\}^\lambda$.
  3. Compute the translated wire keys:
       $\tilde{k}_a^{\pi_a} := \mathcal{F}_{k_a^{\pi_a}}(g\|0)_{\mathsf{abl}}$,    $\tilde{k}_a^{\bar{\pi}_a} := \tilde{k}_a^{\pi_a} \oplus \Delta$
       $\tilde{k}_b^{\pi_b} := \mathcal{F}_{k_b^{\pi_b}}(g\|0)_{\mathsf{abl}}$,    $\tilde{k}_b^{\bar{\pi}_b} := \tilde{k}_b^{\pi_b} \oplus \Delta$
  4. Set output wire keys: $k_c^0 := \tilde{k}_a^0 \oplus \tilde{k}_b^0$,    $k_c^1 := k_c^0 \oplus \Delta$.
  5. Compute the ciphertexts:
       $T_a := \mathcal{F}_{k_a^{\bar{\pi}_a}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_a^{\bar{\pi}_a}$,    $T_b := \mathcal{F}_{k_b^{\bar{\pi}_b}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_b^{\bar{\pi}_b}$.
  6. Output $\left(k_c^0, k_c^1, \pi_c, \tilde{g} = (T_a, T_b)\right)$.
```

Figure 2: Garbling procedure **PAG**Xor for an XOR gate.

4.1 Garbling XOR Gates

We will now describe how to garble an XOR gate. The full description can be found in Fig. 2. To garble an XOR gate g with input wires a, b and output wire c given four input wire keys $\{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}$ and permutation bits π_a, π_b, do the following:

1. Compute the permutation bit π_c of the output wire as $\pi_a \oplus \pi_b$.
2. Sample a uniformly random gate offset Δ.
3. Rerandomize each input wire key by computing a translated input wire key. We are going to denote by $\tilde{k}$ the translated key of k. For each input wire, one of the translated keys is derived from $\mathcal{F}$ using the input wire key as PRF key. The other translated input wire key is set to be correlated by the gate offset Δ:
$$\tilde{k}_a^{\pi_a} := \mathcal{F}_{k_a^{\pi_a}}(g\|0)_{\mathsf{abl}}, \qquad \tilde{k}_a^{\bar{\pi}_a} := \tilde{k}_a^{\pi_a} \oplus \Delta$$
$$\tilde{k}_b^{\pi_b} := \mathcal{F}_{k_b^{\pi_b}}(g\|0)_{\mathsf{abl}}, \qquad \tilde{k}_b^{\bar{\pi}_b} := \tilde{k}_b^{\pi_b} \oplus \Delta$$
Observe that since $\mathcal{F}$ is a pseudorandom function, the translated input wire keys are independent of the original ones.
4. Compute the output wire keys as $k_c^0 := \tilde{k}_a^0 \oplus \tilde{k}_b^0$ and $k_c^1 := k_c^0 \oplus \Delta$, i.e. the output wire keys are correlated by the same offset as the translated input wire keys. Now the evaluator can XOR two translated input wire keys to compute the corresponding output wire key.
5. For each input wire $i \in \{a, b\}$, the evaluator can only derive at most one translated input wire key $\tilde{k}_i^{\pi_i}$ since they do not know the offset Δ. For the case that the evaluator needs the other translated input wire key $\tilde{k}_i^{\bar{\pi}_i}$, we are going to provide an encryption $T_i := \mathcal{F}_{k_i^{\bar{\pi}_i}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_i^{\bar{\pi}_i}$ of it, using the input wire key as encryption key.

In order to evaluate an XOR gate, the evaluator computes the translated input wire keys where they can, and decrypt the ciphertexts corresponding to the remaining ones. The XOR of the translated input wire keys yields the output wire key.

```
PAGAnd $(g, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b)$    //HybAnd0

1. Set  $\mathsf{K}_0 := \mathcal{F}_{k_a^{\pi_a}}(g\|00) \oplus \mathcal{F}_{k_b^{\pi_b}}(g\|00)$,   $\mathsf{K}_1 := \mathcal{F}_{k_a^{\pi_a}}(g\|01) \oplus \mathcal{F}_{k_b^{\bar{\pi}_b}}(g\|01)$
        $\mathsf{K}_2 := \mathcal{F}_{k_a^{\bar{\pi}_a}}(g\|10) \oplus \mathcal{F}_{k_b^{\pi_b}}(g\|10)$,   $\mathsf{K}_3 := \mathcal{F}_{k_a^{\bar{\pi}_a}}(g\|11) \oplus \mathcal{F}_{k_b^{\bar{\pi}_b}}(g\|11)$

2.  Set output wire keys:
      If $\pi_a = \pi_b = 1$ then  $k_c^1 := \mathsf{K}_{0\mathsf{abl}}$,   $k_c^0 \leftarrow \{0,1\}^\lambda$,   $\pi_c := \overline{\mathsf{K}_{0\mathsf{lsb}}}$.
                         else  $k_c^1 \leftarrow \{0,1\}^\lambda$,   $k_c^0 := \mathsf{K}_{0\mathsf{abl}}$,   $\pi_c := \mathsf{K}_{0\mathsf{lsb}}$.

3. let $t^0 := k_c^0\|\pi_c$ and $t^1 := k_c^1\|\bar{\pi}_c$. //auxiliary variables

4. Compute the ciphertexts:
   $T_1 := \mathsf{K}_1 \oplus t^{\pi_a \wedge \bar{\pi}_b}$,   $T_2 := \mathsf{K}_2 \oplus t^{\bar{\pi}_a \wedge \pi_b}$,   $T_3 := \mathsf{K}_3 \oplus t^{\bar{\pi}_a \wedge \bar{\pi}_b}$.

5. Output $(k_c^0, k_c^1, \pi_c, \tilde{g} = (T_1, T_2, T_3))$.
```

Figure 3: Garbling procedure **PAG**And for an AND gate.

4.2 Our Garbling Scheme

We can now provide a full description of our garbling scheme.

Garbling single gates. Full descriptions of garbling AND and XOR gates can be found in Fig. 3 and 2. The garbling of AND gates follows the three-row reduction technique from [14] based on [24]. XOR gates are garbled as in the modification described above. Each gate garbling procedure takes as input a gate index g, two input wire keys $\{k^\sigma\}_{\sigma\in\{0,1\}}$ per input wire, and a permutation bit π per input wire. The output consists of two output wire keys $\{k_c^\sigma\}_{\sigma\in\{0,1\}}$, a permutation bit π for the output wire, and the actual garbled gate $\widetilde{g}$.

```
PAGGCircuit(λ, C)
    1. Garble Circuit:
        ◦ (Input wires) For $i \in [n]$, $\sigma \in \{0,1\}$,
          $k^\sigma_{\mathsf{in}_i} \leftarrow \{0,1\}^\lambda$, $\pi_{\mathsf{in}_i} \leftarrow \{0,1\}$
          $K := (k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i}, \pi_{\mathsf{in}_i})_{i\in[n]}$.
        ◦ (Gates) For $g_i = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$:
          $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i) \leftarrow \mathbf{PAG}\mathsf{Gate}\left(i, \{k^\sigma_a, k^\sigma_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b\right)$.
    2. Output $\widetilde{C} = (\widetilde{g}_i)_{i\in[p]}$, $k = \left(K, (\pi_j)_{j\in[m]}\right)$.

GInput(x, k)
    1. Parse $k$ into $(k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i}, \pi_{\mathsf{in}_i})_{i\in[n]}$ and $(\pi_j)_{j\in[m]}$
    2. Select input keys $K^x = \left(k^{x_1}_{\mathsf{in}_1} \| (\pi_{\mathsf{in}_1} \oplus x_1), \ldots, k^{x_n}_{\mathsf{in}_n} \| (\pi_{\mathsf{in}_n} \oplus x_n)\right)$.
    3. Output $\tilde{x} = \left(K^x, (\pi_j)_{j\in[m]}\right)$.
```

Figure 4: **PAG** Garbling Scheme: $\mathsf{GCircuit}$ and GInput functions. See Figure 5 for function Eval.

Garbling circuit and input. Fig. 4 presents the procedures for circuit and input garbling. **PAG**$\mathsf{GCircuit}$ takes the security parameter λ and a circuit C to be garbled as input, and outputs a garbled circuit $\widetilde{C}$ as well as some state k. To garble the circuit, **PAG**$\mathsf{GCircuit}$ first chooses two uniformly random input wire keys and a permutation bit for each input wire, and then traverses C and garbles each gate according to **PAG**Gate for gate type $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$. The garbled circuit $\widetilde{C}$ then consists of garblings $\widetilde{g}_i$ of all gates g_i in the circuit. The procedure also outputs the keys assigned to the input wires of the circuits as well as their permutation bits, for later use when garbling the input. GInput takes input x and the state from **PAG**$\mathsf{GCircuit}$ and outputs the input garbling $\tilde{x}$. Note that $\tilde{x}$ contains the output map which is known to be necessary in the adaptive setting by the lower bound of Applebaum et al. [5].

Evaluation. Fig. 5 shows the evaluation procedure given a garbled circuit $\widetilde{C}$ and a garbled input $\tilde{x}$. **PAG**Eval parses the garbled input $\tilde{x}$ into K containing input wire keys as well as permutation bits for the input wires, and the output map consisting of signal bits ϕ_j for each output wire j. The procedure then traverses the circuit and evaluates each gate. Finally, the output map is applied to learn the output of the circuit evaluation.

Correctness and Security. Correctness and selective security of the scheme follow from an argument similar to that for correctness of the fast garbling technique. We therefore omit the proofs and refer the reader to their work. Security holds under the assumption that $\mathcal{F}$ is a pseudorandom function. In the next section, we are going to show that unlike fast garbling, our new garbling scheme additionally provides adaptive security under the same assumption.

```
**PAG**Eval($\widetilde{C}, \tilde{x}$)

1. Parse $\tilde{x} = (K, (\pi_j)_{j \in [m]})$.
2. Evaluate Circuit:
   • Parse $K = (k_{\mathsf{in}_1} \| \phi_{\mathsf{in}_1}, \ldots, k_{\mathsf{in}_n} \| \phi_{\mathsf{in}_n})$.
   • For each level $j = 1, \ldots, d$, and each $g_i = (\mathsf{Gate}, a, b, c)$ at level $j$:
     – if $\mathsf{Gate} = \mathsf{AND}$:
         Let $\widetilde{g}_i = (T_1, T_2, T_3)$, $\alpha := 2\phi_a + \phi_b$, and $T_0 := 0$.
         Set $k_c \| \phi_c := T_\alpha \oplus F_{k_a}(g \| \phi_a \phi_b) \oplus F_{k_b}(g \| \phi_a \phi_b)$.
     – if $g = \mathsf{XOR}$
         Let $\widetilde{g}_i = [T_{a,1}, T_{b,1}]$, and $T_{a,0} = T_{b,0} = 0$.
         Set $k_c := (T_{a,\phi_a} \oplus F_{k_a}(g \| \phi_a)_{\mathsf{abl}}) \oplus \big(T_{b,\phi_b} \oplus F_{k_b}(g \| \phi_b)_{\mathsf{abl}}\big)$.
         $\phi_c := \phi_a \oplus \phi_b$.
3. Unmask output: For $j \in [m]$,
   • Set $y_j := \pi_j \oplus \phi_j$.
4. **Output** $y_1, \ldots, y_m$.
```

Figure 5: **PAG** Garbling Scheme: Eval function.

5 Security Proof

Our goal is to show indistinguishability of real and simulated garbling towards an adaptive adversary. To this extent, we follow [17]. We will first define a template for hybrid games and then establish rules for transitioning between them.

5.1 Setting up hybrid games

Each gate will be assigned a *gate mode* from $\{\mathsf{RealGate}, \mathsf{InputDepSimGate}, \mathsf{SimGate}\}$ which indicates the way it is garbled. For $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$, the gate is garbled according to **PAG**Gate and **SimPAG**Gate, respectively (cf. Fig. 3 and 6 for AND gates and Fig. 2 and 7 for XOR gates). A *circuit configuration* consists of two sets $(\mathsf{mode}_i)_{i \in [q]}$ and I. The first one specifies the gate mode mode_i of each gate g_i. The second set is a set of *guessed wires*. We will let $((\mathsf{mode}_i)_{i \in [q]}, I)$ denote a circuit configuration and sometimes use mode as a shorthand for $(\mathsf{mode}_i)_{i \in [q]}$. A circuit configuration is called *valid* if the outgoing wire of every gate that is in InputDepSimGate mode is contained in I.

The hybrid game $\mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, I)$ corresponding to valid circuit configuration (mode_i, I) and interacting with adversary $\mathcal{A}$ is described in Fig. 9. When the adversary $\mathcal{A}$ is clear from the context, we will simply write $\mathsf{Hyb}(\mathsf{mode}, I)$ instead. The hybrid game consists of two phases, a guessing phase and a garbling phase. The game starts by setting up a set of guesses Guess that contains a guessed bit for each wire $w_i \in I$. Those guesses are necessary to garble gates in InputDepSimGate as their garbling procedure requires knowledge of the value on the outgoing wire. The game then receives the circuit C and garbles it. Here, each gate is garbled according to its gate mode, possibly using guesses whenever necessary. When the adversary has received $\widetilde{C}$ and returns an input x, the game checks the guesses, and aborts with fixed output 0 if any of them were incorrect. Otherwise, it proceeds by garbling the input and creating the output map. Finally, the game outputs the garbled input $\tilde{x}$ and the output map to the adversary and outputs its output.

We can now relate the hybrid games to the definition of adaptive security for garbling schemes (cf. Def. 2). The real game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 0)$ is equivalent to $\mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, \emptyset)$ where each $\mathsf{mode}_i = \mathsf{RealGate}$. We will compare the real game to the simulated game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 1)$ for simulator Sim = **SimPAG**GCircuit described in Fig. 8, which is equivalent to $\mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, \emptyset)$ with $\mathsf{mode}_i = \mathsf{SimGate}$ for all $i \in [q]$.

```
HybAnd1 $(g, \{k_a, k_b\}, \phi_a, \phi_b, v_c)$

    1. If $\phi_a = \phi_b = 0$ set
       $k_c^{v_c} \| \phi_c := F_{k_a}(g\|00) \oplus F_{k_b}(g\|00)$ and $T_\alpha \leftarrow \{0,1\}^{\lambda+1}, \alpha \in \{1,2,3\}$.

    2. else set  $k_c^{v_c} \| \phi_c \leftarrow \{0,1\}^{\lambda+1}$
                 $T_{2\phi_a+\phi_b} := F_{k_a}(g\|\phi_a\phi_b) \oplus F_{k_b}(g\|\phi_a\phi_b) \oplus (k_c\|\phi_c)$
                 $T_\alpha \leftarrow \{0,1\}^{\lambda+1}$ for $\alpha \in \{1,2,3\} \setminus 2\phi_a + \phi_b$.

    3. Set $\pi_c := \phi_c \oplus v_c$, and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.

    4. Output $\left(k_c^0, k_c^1, \pi_c, \widetilde{g} = (T_1, T_2, T_3)\right)$.

SimPAGAnd $\left(g, \{k_a^\sigma, k_b^\sigma\}_{\sigma \in \{0,1\}}, \pi_a, \pi_b, v_c\right)$    //HybAnd2

    1. Set:  $\mathsf{K}_0 := \mathcal{F}_{k_a^{\pi_a}}(g\|00) \oplus \mathcal{F}_{k_b^{\pi_b}}(g\|00)$,   $\mathsf{K}_1 := \mathcal{F}_{k_a^{\pi_a}}(g\|01) \oplus \mathcal{F}_{k_b^{\bar{\pi}_b}}(g\|01)$
             $\mathsf{K}_2 := \mathcal{F}_{k_a^{\bar{\pi}_a}}(g\|10) \oplus \mathcal{F}_{k_b^{\pi_b}}(g\|10)$,   $\mathsf{K}_3 := \mathcal{F}_{k_a^{\bar{\pi}_a}}(g\|11) \oplus \mathcal{F}_{k_b^{\bar{\pi}_b}}(g\|11)$.

    2. Set output wire keys $k_c^{v_c} \| \phi_c := \mathsf{K}_0$,   $\pi_c = \phi_c \oplus v_c$ and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.

    3. Compute the ciphertexts, $T_\alpha := \mathsf{K}_\alpha \oplus \mathsf{K}_0$, for $\alpha \in \{1,2,3\}$.

    4. Output $\left(k_c^0, k_c^1, \pi_c, \widetilde{g} = (T_1, T_2, T_3)\right)$.
```

Figure 6: Distributions HybAnd1 and **SimPAG**And for AND gate, defined for adaptive security proof.

5.2 Indistinguishability of hybrid games

In this section, we will provide rules for moving from one circuit configuration to another and prove that the corresponding hybrid games are indistinguishable.

Define $\mathsf{TimeGC}(q)$ to be the time it takes to garble a circuit of size q using our garbling scheme.

Definition 5 (Neighboring hybrids, target gate [17])**.** We say two valid hybrids or configurations $((\mathsf{mode}_i)_{i\in[q]}, I), ((\mathsf{mode}'_i)_{i\in[q]}, I)$ are *neighboring* if the set of guessed wires I is the same in both of them and the garbling modes of all gates except one are the same; i.e. there exists some $j \in [q]$ such that for all $i \neq j$ we have $\mathsf{mode}_i = \mathsf{mode}'_i$. We call g_j the *target gate* of the two hybrids or configurations.

Definition 6 (Predecessor/Successor/Sibling gates [17])**.** Given a circuit C and a gate $j \in [q]$ of the form $g_j = (g, w_a, w_b, w_c)$ with incoming wires w_a, w_b and outgoing wire w_c:

- We define the *predecessors* of j, denoted by $\mathsf{Pred}(j)$, to be the set of gates whose outgoing wires are either w_a or w_b. If w_a, w_b are input wires, then $\mathsf{Pred}(j) = \emptyset$, else $|\mathsf{Pred}(j)| = 2$.
- We define the *successors* of j, denoted by $\mathsf{Succ}(j)$, to be the set of gates that contain w_c as an incoming wire. If w_c is an output wire, then $\mathsf{Succ}(j) = \emptyset$.

We define the *siblings* of j, denoted by $\mathsf{Siblings}(j)$, to be the set of gates that contain either w_a or w_b as an incoming wire.

5.2.1 Rule 1: RealGate $\leftrightarrow$ InputDepSimGate

The following rule allows changing a gate's mode from RealGate to InputDepSimGate if all of its predecessors are in InputDepSimGate mode. That means that if a circuit configurations has a target gate with gate mode RealGate, then it is allowed to move to the neighboring configuration with target gate in InputDepSimGate mode (and vice versa).

```
HybXor1 $(g, \{k_a, k_b\}, \phi_a, \phi_b, v_c)$

  1. Compute the translated keys and the ciphertexts for $\alpha \in \{b, a\}$:
       If $\phi_\alpha = 0$ then  $\tilde{k}_\alpha := F_{k_\alpha}(g\|0)_{\mathsf{abl}}$  and $T_\alpha \leftarrow \{0,1\}^\lambda$
                          else  $\tilde{k}_\alpha \leftarrow \{0,1\}^\lambda$  and $T_\alpha := F_{k_\alpha}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$.
  2. Set output wire keys $k_c^{v_c} = \tilde{k}_a \oplus \tilde{k}_b$,  $\pi_c = \phi_a \oplus \phi_b \oplus v_c$ and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.
  3. Output $\left(k_c^0, k_c^1, \pi_c, \widetilde{g} = (T_a, T_b)\right)$.

SimPAGXor $\left(g, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \phi_a, \phi_b, v_c\right)$    //HybXor2

  1. Compute the translated keys and the ciphertexts for $\alpha \in \{a, b\}$:
       If $\phi_\alpha = 0$ then  $\tilde{k}_\alpha := \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|0)_{\mathsf{abl}}$  and $T_\alpha := \mathcal{F}_{k_\alpha^{\bar{v}_\alpha}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$.
                          else  $\tilde{k}_\alpha := \mathcal{F}_{k_\alpha^{\bar{v}_\alpha}}(g\|0)_{\mathsf{abl}}$  and $T_\alpha := \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$.
  2. Set output wire keys $k_c^{v_c} = \tilde{k}_a \oplus \tilde{k}_b$,  $\pi_c = \phi_a \oplus \phi_b \oplus v_c$ and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.
  3. Output $(k_c, \phi_c, \widetilde{g} = (T_a, T_b))$.
```

Figure 7: Distributions HybXor1 and **SimPAG**Xor for XOR gates, defined for the proof of adaptive security.

Lemma 2. Let $\mathsf{Hyb}(\mathsf{mode}, I)$ and $\mathsf{Hyb}(\mathsf{mode}', I)$ be two neighboring hybrids that differ in target gate g_j, i.e. $\mathsf{mode}_j = \mathsf{RealGate}$ in mode and $\mathsf{mode}'_j = \mathsf{InputDepSimGate}$ in mode'. Furthermore, all $i \in \mathsf{Pred}(j)$ satisfy $mode_i = \mathsf{InputDepSimGate}$. Then for any adversary $\mathcal{A}$, the hybrids $\mathsf{Hyb}_\mathcal{A}(\mathsf{mode}, I)$ and $\mathsf{Hyb}_\mathcal{A}(\mathsf{mode}', I)$ are $(T(\lambda), 2\varepsilon(\lambda))$-indistinguishable if $\mathcal{F}$ is a $(T(\lambda) + \mathsf{TimeGC}(|C|), \varepsilon)$-secure 2PRF.

Due to space constraints, the proof of Lemma 2 is deferred to Appendix A.

5.2.2 Rule 2: InputDepSimGate $\leftrightarrow$ SimGate

This rule allows changing a gate mode from InputDepSimGate to SimGate if all of its successors are in InputDepSimGate or SimGate mode.

Lemma 3. Let $\mathsf{Hyb}(\mathsf{mode}, I)$ and $\mathsf{Hyb}(\mathsf{mode}', I)$ be two neighboring hybrids that differ in target gate g_j, i.e. $\mathsf{mode}_j = \mathsf{InputDepSimGate}$ in mode and $\mathsf{mode}'_j = \mathsf{SimGate}$ in mode'. Furthermore, all $i \in \mathsf{Succ}(j)$ satisfy $mode_i \in \{\mathsf{SimGate}, \mathsf{InputDepSimGate}\}$. Then for any adversary $\mathcal{A}$, the hybrids $\mathsf{Hyb}_\mathcal{A}(\mathsf{mode}, I)$ and $\mathsf{Hyb}_\mathcal{A}(\mathsf{mode}', I)$ are identically distributed.

Proof. Let $\mathcal{A}$ be an arbitrary but fixed adversary. Then the two hybrids $H_0 := \mathsf{Hyb}_\mathcal{A}(\mathsf{mode}, I)$ and $H_1 = \mathsf{Hyb}_\mathcal{A}(\mathsf{mode}', I)$ differ only in how the target gate $g_j = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$ is garbled: In the hybrid H_0, gate garbling $\widetilde{g}_j$ is computed as $\widetilde{g}_j \leftarrow \textbf{SimPAG}\mathsf{Gate}\left(j, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, \mathsf{Guess}(c)\right)$, whereas in H_1, $\widetilde{g}_j \leftarrow \textbf{SimPAG}\mathsf{Gate}\left(j, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, 0\right)$. Now there are the following cases to distinguish, depending on where g_j is located in the circuit.

If g_j is not an output gate and all successors are in $\{\mathsf{SimGate}, \mathsf{InputDepSimGate}\}$, then the gate's output keys k_c^0 and k_c^1 are treated symmetrically within the gate, i.e. their roles are interchangeable. Moreover, the permutation bit π_c is independent of the output keys. This holds for both AND and XOR gates.

If g_j is an output gate, then the output keys are only used in the output map and the gate itself. As argued above, the output keys are treated symmetrically within the gate, and the output permutation bit is independent of the output keys. Hence, in the output map, there is no difference between setting $\widetilde{d}_j := \pi_j$ in H_0 and $\widetilde{d}_j := y_j \oplus \pi_j$ in H_1.

Finally, in the case that all successors of an input gate g_i are in SimGate mode, the input gate is garbled in a slightly different way: Instead of setting $K[i] := k_{\mathsf{in}_i}^{x_i} \| (\pi_{\mathsf{in}_i} \oplus x_i)$, it is set to $K[i] := k_{\mathsf{in}_i}^0 \| \pi_{\mathsf{in}_i}$. However, both cases are identically distributed as the keys $k_{\mathsf{in}_i}^{x_i}$ and $k_{\mathsf{in}_i}^0$ are treated symmetrically everywhere in the game and the permutation bit π_{in_i} is uniformly random. $\square$

```
SimPAGGCircuit(λ, C)

  1. Garble Circuit:
     • (Input wires) For $i \in [n], \sigma \in \{0,1\}$,
       $k^{\sigma}_{\mathsf{in}_i} \leftarrow \{0,1\}^{\lambda}, \pi_{\mathsf{in}_i} \leftarrow \{0,1\}$
       $K = (k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i}, \pi_{\mathsf{in}_i})_{i\in[n]}$.
     • (Gates) For $g_i = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$ in $C$:
       $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i) \leftarrow \mathbf{SimPAG}\mathsf{Gate}\big(i, \{k^{\sigma}_a, k^{\sigma}_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, 0\big)$.
     • (Decoding info) For $j \in [m]$: Set $\widetilde{\mathsf{sd}}_j = \pi_j$.
  2. Output $\widetilde{C} = (\widetilde{g}_i)_{i\in[p]}$, $k = \Big(K, (\widetilde{\mathsf{sd}}_j)_{j\in[m]}\Big)$.

SimGInput(y, (K, (sd_j)_{j∈[m]}))

  1. Set the output permutation bits $\pi_j := \widetilde{\mathsf{sd}}_j \oplus y_j$.
  2. Select output keys, $K^x = \big(k^0_{\mathsf{in}_1} \| \pi_{\mathsf{in}_i}, \ldots, k^{x_0}_{\mathsf{in}_n} \| \pi_{\mathsf{in}_i}\big)$.
  3. Output $\tilde{x} = \big(K^x, (\pi_j)_{j\in[m]}\big)$.
```

Figure 8: Simulation for **PAG** Garbling Scheme.

Finally, we will use the following lemma that allows to scale the number of guesses up and down to allow for comparison of hybrids that differ in the wires corresponding to their guesses.

Lemma 4. If $\mathbf{D}_{T(\lambda)}\big[\mathsf{Hyb}(\mathsf{mode}, I), \mathsf{Hyb}(\mathsf{mode}', I)\big] = \varepsilon(\lambda)$ and J is a set of wires disjoint from I, then

$$\mathbf{D}_{T(\lambda)}\big[\mathsf{Hyb}(\mathsf{mode}, I \cup J), \mathsf{Hyb}(\mathsf{mode}', I \cup J)\big] = 2^{-|J|} \cdot \varepsilon(\lambda).$$

The proof is analogous to that of [17, Lemma 3]. The idea is that adding J places the additional burden of guessing the wire values corresponding to J correctly on the adversary.

5.3 Sequences of Hybrid Games

So far, we have established that certain hybrid games with valid neighboring configurations are indistinguishable. What remains to find is a sequence of indistinguishable hybrid games for an arbitrary circuit that goes from the real game $\mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, \emptyset)$ where each $\mathsf{mode}_i = \mathsf{RealGate}$ to the simulated game $\mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, \emptyset)$ with $\mathsf{mode}_i = \mathsf{SimGate}$ for all $i \in [q]$. The pebbling strategy for the graph will then imply a sequence of indistinguishable hybrid games from the real game to the simulation.

5.3.1 Pebbling Game

Consider the following pebble game:

Graph of circuit C: Given a circuit C, the corresponding graph consists of a node for each gate, and a directed edge from node i to node j if the outgoing wire from gate i is an ingoing wire for gate j. Furthermore, we think of each input wire as outgoing wire of a dummy input gate.

Pebbles: Each node can have either *no pebble*, a *black pebble*, or a *gray pebble* on it, corresponding to the gate being in RealGate, InputDepSimGate, and SimGate mode, respectively.

Pebbling rules: The game consists of the following rules:
Pebbling rule A: We can place or remove a black pebble on a node as long as both predecessors of it have black pebbles on them, or it corresponds to an input gate.

```
Game $\mathsf{Hyb}^{\lambda}_{\mathcal{A}}((\mathsf{mode}_i)_{i\in[q]}, I)$
  1. (Guesses) For all $w_i \in I$,
     • Let $\mathsf{Guess}(w_i) \leftarrow \{0,1\}$.
  2. **Receive $C$ from $\mathcal{A}$.**
     Garble circuit $C$:
  3. (Input wires) For $i \in [n]$, $\sigma \in \{0,1\}$ :   $k^{\sigma}_{\mathsf{in}_i} \leftarrow \mathsf{Gen}(1^\lambda)$,   $\pi_{\mathsf{in}_i} \leftarrow \{0,1\}$.
  4. (Gates) For each $g_i = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$, in $C$ :
       If $\mathsf{mode}_i$   $= \mathsf{RealGate}$,
                  $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i) \leftarrow$   $\mathbf{PAG}\mathsf{Gate}\,(i, \{k^{\sigma}_a, k^{\sigma}_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b)$.
       else       I. If  $\mathsf{mode}_i =$   $\mathsf{SimGate}$, let $\mu := 0$ else $\mu := \mathsf{Guess}(c)$.
                  II. $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i)$   $\leftarrow \mathbf{SimPAG}\mathsf{Gate}\,(i, \{k^{\sigma}_a, k^{\sigma}_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, \mu)$.
  5. **Send $\widetilde{C}$ to $\mathcal{A}$ and get back $x$.**
     Garble Input $x$:
  6. (Check the guesses) For each $i \in I$,
       – Let $v(w_i)$ be the bit on the wire $w_i$ during the computation $C(x)$.
       – If $v(w_i) \neq \mathsf{Guess}(w_i)$ **Output 0** and abort.
  7. (Choose the color-bits ) Let $y = C(x)$. For $\ell = 1, \ldots, m$:
       – If $\mathsf{mode}_\ell \neq \mathsf{SimGate}$, set $\widetilde{d}_\ell = \pi_\ell$
       – else, set $\widetilde{d}_\ell := y_\ell \oplus \pi_\ell$.
  8. (Select input keys) For $\ell = 1, \ldots, n$:
       – If all gates $i$ having $\mathsf{in}_\ell$ as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then
         set $K[\ell] := k^0_{\mathsf{in}_\ell} \| \pi_{\mathsf{in}_\ell}$,
       – else set $K[\ell] := k^{x_\ell}_{\mathsf{in}_\ell} \| (\pi_{\mathsf{in}_\ell} \oplus x_\ell)$.
  9. **Send $\widetilde{x} := (K, \{\widetilde{d}_\ell\}_{\ell\in[m]})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.**
```

Figure 9: The Hybrid Game.

Pebbling rule B: We can replace a black pebble by a gray pebble on a node as long as all successors of it have black or gray pebbles on them, or it corresponds to an output gate.

Pebble configuration: A *pebble configuration* specifies for each node if it contains no pebble, a black pebble or a gray pebble.

Pebbling: A pebbling of a graph corresponding to circuit C is a sequence of γ moves that starts with no pebbles on the graph and follows the pebbling rules until it reaches a pebble configuration where all nodes have gray pebbles on them. A pebbling is said to use t black pebbles if t is the maximal number of black pebbles used at any point during the pebble game.

5.3.2 From Pebbling to Sequence of Hybrids

Each pebble configuration specifies a circuit configuration and hence a hybrid game in the following way. For each gate $i \in [q]$, set its gate mode mode_i according to the pebble on its corresponding node:

- $\mathsf{mode}_i = \mathsf{RealGate}$ if it has no pebble,

```
**Reduction** from **PAG**And to HybAnd1
**ReducOne**$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{AND},\mathcal{A}}\left(g, \{k_a^{v_a}, k_b^{v_a}\}, \phi_a, \phi_b, v_a, v_b\right)$

  1. Set: $\mathsf{K}_{2\phi_a+\phi_b} := \mathcal{F}_{k_a^{v_a}}(g\|\phi_a\phi_b) \oplus \mathcal{F}_{k_b^{v_b}}(g\|\phi_a\phi_b)$,
          $\mathsf{K}_{2\phi_a+\bar{\phi}_b} := \mathcal{F}_{k_a^{v_a}}(g\|\phi_a\bar{\phi}_b) \oplus \mathcal{O}_b(g\|\phi_a\bar{\phi}_b)$,
          $\mathsf{K}_{2\bar{\phi}_a+\phi_b} := \mathcal{O}_a(g\|\bar{\phi}_a\phi_b) \oplus \mathcal{F}_{k_b^{v_b}}(g\|\bar{\phi}_a\phi_b)$,
          $\mathsf{K}_{2\bar{\phi}_a+\bar{\phi}_b} := \mathcal{O}_a(g\|\bar{\phi}_a\bar{\phi}_b) \oplus \mathcal{O}_b(g\|\bar{\phi}_a\bar{\phi}_b)$.

  2. Compute $\pi_a = v_a \oplus \phi_a$ and $\pi_b = v_b \oplus \phi_b$.

  3. Set output wire keys:
       If $\pi_a = \pi_b = 1$ then  $k_c^1 := \mathsf{K}_{0\mathsf{abl}}$,  $k_c^0 \leftarrow \{0,1\}^\lambda$,  $\pi_c := \overline{\mathsf{K}_{0\mathsf{lsb}}}$.
                          else  $k_c^1 \leftarrow \{0,1\}^\lambda$,  $k_c^0 := \mathsf{K}_{0\mathsf{abl}}$,  $\pi_c := \mathsf{K}_{0\mathsf{lsb}}$.

  4. let $t^0 := k_c^0\|\pi_c$ and $t^1 := k_c^1\|\bar{\pi}_c$.

  5. Compute the ciphertexts:
       $T_1 := \mathsf{K}_1 \oplus t^{\pi_a \wedge \bar{\pi}_b}$,  $T_2 := \mathsf{K}_2 \oplus t^{\bar{\pi}_a \wedge \pi_b}$,  $T_3 := \mathsf{K}_3 \oplus t^{\bar{\pi}_a \wedge \bar{\pi}_b}$.

  6. **Output** $\left(k_c^0, k_c^1, \pi_c, \widetilde{g} = (T_1, T_2, T_3)\right)$.

**Reduction** from HybAnd1 to **SimPAG**And
**ReducTwo**$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{AND},\mathcal{A}}\left(g, \{k_a^{v_a}, k_b^{v_a}\}, \phi_a, \phi_b, v_c\right)$

  1. Set: $\mathsf{K}_{2\phi_a+\phi_b} := \mathcal{F}_{k_a^{v_a}}(g\|\phi_a\phi_b) \oplus \mathcal{F}_{k_b^{v_b}}(g\|\phi_a\phi_b)$,
          $\mathsf{K}_{2\phi_a+\bar{\phi}_b} := \mathcal{F}_{k_a^{v_a}}(g\|\phi_a\bar{\phi}_b) \oplus \mathcal{O}_b(g\|\phi_a\bar{\phi}_b)$,
          $\mathsf{K}_{2\bar{\phi}_a+\phi_b} := \mathcal{O}_a(g\|\bar{\phi}_a\phi_b) \oplus \mathcal{F}_{k_b^{v_b}}(g\|\bar{\phi}_a\phi_b)$,
          $\mathsf{K}_{2\bar{\phi}_a+\bar{\phi}_b} := \mathcal{O}_a(g\|\bar{\phi}_a\bar{\phi}_b) \oplus \mathcal{O}_b(g\|\bar{\phi}_a\bar{\phi}_b)$.

  2. Set output wire key $k_c^{v_c}\|\phi_c := \mathsf{K}_0$,  $\pi_c := \phi_c \oplus v_c$,  and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.

  3. Compute the ciphertexts, $T_\alpha := \mathsf{K}_\alpha \oplus \mathsf{K}_0$, for $\alpha \in \{1,2,3\}$.

  4. **Output** $\left(k_c^0, k_c^1, \pi_c, \widetilde{g} = (T_1, T_2, T_3)\right)$.
```

Figure 10: Reductions between AND hybrid distributions. We use **Reduc**$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{AND},\mathcal{A}}$, to refer to the two reductions combined, which shows **PAG**And and **SimPAG**And are indistinguishable, based on PRFs.

- $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ if it has a black pebble,
- $\mathsf{mode}_i = \mathsf{SimGate}$ if it has a gray pebble,

The set I is the set of output wires of those gates that have black pebbles on them. By definition, this is a valid circuit configuration. A pebbling in γ moves consists of a sequence of $\gamma+1$ pebble configurations, starting with no pebbles and ending with a gray pebble on each node, where each pebble configuration is obtained following rule A or B. Each pebbling defines a sequence of hybrid games $H_\alpha = \mathsf{Hyb}(\mathsf{mode}^\alpha, I^\alpha)$ for $\alpha = 0, \ldots, \gamma$, starting from the real game H_0 (circuit configuration $\mathsf{Hyb}(\mathsf{mode}, \emptyset)$ with $\mathsf{mode}_i = \mathsf{RealGate}$) and ending in the ideal game H_γ (circuit configuration $\mathsf{Hyb}(\mathsf{mode}, \emptyset)$ with $\mathsf{mode}_i = \mathsf{SimGate}$).

What remains to show is that the hybrids in the sequence are indeed indistinguishable, i.e. a connection between rules 1 and 2 and pebbling rules A and B.

Theorem 1. *Assume that $\mathcal{F}$ is a $(T + \mathsf{TimeGC}(|C|), \varepsilon(\lambda))$-secure 2PRF. If there is a pebbling of circuit C in γ moves, using t pebbles, then $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 0)$ and $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 1)$ are $(T'(\lambda), \varepsilon'(\lambda))$-indistinguishable with*

- $\varepsilon'(\lambda) \leq \sum_{i=1}^{\gamma} 2^{r_i+1} \cdot \varepsilon(\lambda) \leq \gamma \cdot 2^{t+1} \cdot \varepsilon(\lambda)$ *and*

```
Reduction from PAGXor to HybXor1
ReducOne$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{XOR},\mathcal{A}}\left(g,\{k_a^{v_a},k_b^{v_b}\},\phi_a,\phi_b,v_a,v_b\right)$

  1. $\Delta \leftarrow \{0,1\}^\lambda$.
  2. Compute the translated keys and the ciphertexts for $\alpha \in \{a,b\}$:
       If $\phi_\alpha = 0$ then $\tilde{k}_\alpha = \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|0)_{\mathsf{abl}}$,   and $T_\alpha = \mathcal{O}_\alpha(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha \oplus \Delta$
                      else $\tilde{k}_\alpha = \mathcal{O}_\alpha(g\|0)_{\mathsf{abl}} \oplus \Delta$,   and $T_\alpha = \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$.
  3. Set output wire keys: $k_c^{v_c} = \tilde{k}_a \oplus \tilde{k}_b$,  $\pi_c = \phi_a \oplus \phi_b \oplus v_c$ and $k_c^{\bar{v}_c} := k_c^{v_c} \oplus \Delta$.
  4. Output $\left(k_c^0, k_c^1, \pi_c, \tilde{g} = (T_a, T_b)\right)$.

Reduction from HybXor1 to SimPAGXor
ReducTwo$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{XOR},\mathcal{A}}\left(g,\{k_a^{v_a},k_b^{v_b}\},\phi_a,\phi_b,v_c\right)$

  1. Compute the translated keys and the ciphertexts for $\alpha \in \{a,b\}$:
       If $\phi_\alpha = 0$ then $\tilde{k}_\alpha = \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|0)_{\mathsf{abl}}$,   and $T_\alpha = \mathcal{O}_\alpha(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$
                      else $\tilde{k}_\alpha = \mathcal{O}_\alpha(g\|0)_{\mathsf{abl}}$,   and $T_\alpha = \mathcal{F}_{k_\alpha^{v_\alpha}}(g\|1)_{\mathsf{abl}} \oplus \tilde{k}_\alpha$.
  2. Set output wire keys: $k_c^{v_c} = \tilde{k}_a \oplus \tilde{k}_b$,  $\pi_c = \phi_a \oplus \phi_b \oplus v_c$ and $k_c^{\bar{v}_c} \leftarrow \{0,1\}^\lambda$.
  3. Output $\left(k_c^0, k_c^1, \pi_c, \tilde{g} = (T_a, T_b)\right)$.
```

Figure 11: Reductions for XOR hybrid distributions. We use **Reduc**$^{\mathcal{O}_a,\mathcal{O}_b}_{\mathsf{XOR},\mathcal{A}}$, to refer to the two reductions combined, which shows **PAG**Xor and **SimPAG**Xor are indistinguishable, based on PRFs.

- $T'(\lambda) = T(\lambda) - \mathsf{TimeGC}(|C|)$,

where $r_i = \max(s_{i-1}, s_i)$ *and* s_i *is the number of black pebbles used in the ith pebbling step.*

The complete proof can be found in Supplementary material A.

Finally, we are going to instantiate Theorem 1 with a result of Hemenway et al. [15] stating that for any circuit with q gates and depth d, there exists a pebbling in at most $\gamma = q \cdot 2^{2d}$ moves using $t = 2d$ black pebbles.

Corollary 1. *The* **PAG** *garbling scheme is adaptively secure for all circuits of depth* $d = O(\log \lambda)$ *assuming that pseudorandom functions exist.*

References

[1] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[2] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In *TCC 2018, Part II*, LNCS, pages 455–472. Springer, Heidelberg, March 2018.

[3] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Heidelberg, January 2016.

[4] Benny Applebaum. Garbling XOR gates "for free" in the standard model. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 162–181. Springer, Heidelberg, March 2013.

[5] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[6] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 398–415. Springer, Heidelberg, August 2013.

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

[8] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[9] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017.

[10] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.

[11] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 535–565. Springer, Heidelberg, April / May 2018.

[12] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010.

[13] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2008.

[14] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 567–578. ACM Press, October 2015.

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

[16] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 40–71. Springer, Heidelberg, November 2017.

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

[18] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

[19] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

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

[21] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 2–20. Springer, Heidelberg, September 2008.

[22] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, Heidelberg, August 2014.

[23] Yehuda Lindell and Ben Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 579–590. ACM Press, October 2015.

[24] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.

[25] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 250–267, 2009.

[26] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

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

[28] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

Supplementary material

A Adaptive Security Proof (continued)

Lemma 2. Let $\mathcal{A}$ be an arbitrary but fixed adversary. The two hybrids $H_0 := \mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}, I)$ and $H_1 = \mathsf{Hyb}_{\mathcal{A}}(\mathsf{mode}', I)$ differ only in how the target gate $g_j = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$ is garbled: In the hybrid H_0, the garbled gate $\widetilde{g}_j$ is computed as $\widetilde{g}_j \leftarrow \mathbf{PAG}\mathsf{Gate}\left(i, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b\right)$, whereas in H_1, $\widetilde{g}_j \leftarrow \mathbf{SimPAG}\mathsf{Gate}\left(j, \{k_a^\sigma, k_b^\sigma\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, \mathsf{Guess}(c)\right)$.

Intermediate hybrid game: Intuitively, we would like to argue that an adversary $\mathcal{A}$ cannot distinguish whether the garbling of the target gate contains a single output key or two different ones. However, the $2PRF$ assumption provides only guarantees about indistinguishability of PRF evaluations from random. In order to show indistinguishability of H_0 and H_1, we will therefore define another hybrid game $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}((\mathsf{mode}_i)_{i\in[q]}, I)$ as intermediate step that replaces those PRF evaluations that correspond to ciphertexts that will not be openend by random strings. The game is almost the same as H_0 and H_1 except for the target gate g_j which is garbled according to HybGate1 as described in Fig. 6 and 7, and its surrounding gates.

HybAnd1: If the color bits ϕ_a, ϕ_b are both 0, i.e. if the ciphertext the evaluator can open is the omitted one, then it computes the active output wire key and color bit using $\mathcal{F}$ and sets the three ciphertexts to be random strings. Otherwise, a fresh random active output wire key and color bit are chosen, the ciphertext corresponding to the color bits is computed using $\mathcal{F}$, and the remaining ciphertexts are chosen as random strings. Finally, the permutation bit is computed and a fresh random string is sampled as inactive output wire key.

HybXor1: If the color bit is 0, then compute the translated key using $\mathcal{F}$ and set the ciphertext to be random. Otherwise, choose a random translated key and compute the corresponding ciphertext using $\mathcal{F}$. Finally, compute the active output wire key as XOR of the translated active input wire keys, set the inactive one to be a random string, and compute the permutation bit of the output wire. Note that the gate garbling procedure does not choose a wire offset Δ.

We will first show indistinguishability between H_0 and $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$, and then between $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$ and H_1. In each case, we reduce distinguishing the hybrids to breaking the 2PRF assumption. To this extent, we provide an adversary $\mathcal{B}$ in Fig. 12 that uses a distinguisher $\mathcal{A}$ between two hybrids as subroutine. The adversary $\mathcal{B}$ garbles a circuit while embedding oracle evaluations into the garbling of the target gate. Note that the description of the adversary in Fig. 12 actually contains two adversaries, one for each case. Therefore, the adversary $\mathcal{B}$ is parameterized by the case step it is in that specifies which gate garbling procedure to use for the target gate. Furthermore, the target gate garbling depends on the gate type. The gate garbling procedures can be found in Fig. 10 for AND gates and Fig. **??** for XOR gates.

From H_0 to $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$: Consider the adversary $\mathcal{B}$ (Fig. 12) on input mode, I, target gate index j, $\mathsf{step1}$, and 2PRF oracles $(\mathcal{O}^{(1)}, \mathcal{O}^{(2)}, \mathcal{O}^{(3)}, \mathcal{O}^{(4)})$ that breaks the 2PRF assumption for $\mathcal{F}$ if $\mathcal{A}$ can distinguish H_0 and $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$. The garbling that $\mathcal{B}$ outputs is exactly the same as in H_0, except for the target gate g_j as well as those gates g_i that share an input wire with it. For the latter, it is easy to see that garbling gate g_i using the oracles $\mathcal{O}^{(2)}, \mathcal{O}^{(4)}$ is indistinguishable from garbling g_i according to either of the three other gate modes $\mathsf{RealGate}, \mathsf{InputDepSimGate}, \mathsf{SimGate}$ as the oracles provide PRF evaluations and the key they define implicitly is never revealed as it is not the active input wire key. For the target gate, consider the two gate types separately:

Case 1 - target gate is AND gate: As all predecessors of the target gate are in $\mathsf{InputDepSimGate}$ mode, the input wires to the target gate were generated using $\mathbf{SimPAG}\mathsf{Gate}$ and hence $\mathcal{B}$ holds one wire key on each input wire. Denote these keys by k_a and k_b. It is easy to see that when the oracle outputs evaluations of $\mathcal{F}$, then the result is exactly the same as in the real garbling H_0. On the other hand, if the oracle outputs random strings, then the result will contain three ciphertexts out of which all but the one that the color bits are pointing to are random. The last ciphertext (which may be implicitly defined) is computed using $\mathcal{F}$. Hence, the result is exactly the same as in $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$.

Case 2 - target gate is XOR gate: Again, all predecessors of the target gate are in $\mathsf{InputDepSimGate}$ mode and $\mathcal{B}$ holds one wire key on each input wire. If the oracles' output is according to $\mathcal{F}$, then it is easy to see that the result is exactly the same as in the real garbling H_0. Otherwise, the oracles output random strings and the result is exactly the same as in $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$.

In both cases, H_0 and $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$ are $(T(\lambda), \varepsilon(\lambda))$-indistinguishable.

From $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$ to H_1: This step is very similar to the previous one. We will use adversary $\mathcal{B}$ (Fig. 12) on input mode, I, target gate index j, $\mathsf{step2}$, and 2PRF oracles $(\mathcal{O}^{(1)}, \mathcal{O}^{(2)}, \mathcal{O}^{(3)}, \mathcal{O}^{(4)})$ that breaks the 2PRF assumption for $\mathcal{F}$ if $\mathcal{A}$ can distinguish H_0 and $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$.

The garbling that $\mathcal{B}$ outputs is exactly the same as in H_1, except for the target gate g_j as well as those gates g_i that share an input wire with it. For the latter, it is easy to see that garbling gate g_i using the oracles $\mathcal{O}^{(2)}, \mathcal{O}^{(4)}$ is indistinguishable from garbling g_i according to either of the three other gate modes $\mathsf{RealGate}, \mathsf{InputDepSimGate}, \mathsf{SimGate}$ as the oracles provide PRF evaluations and the key they define implicitly is never revealed as it is not the active input wire key. For the target gate, consider the two gate types separately:

Case 1 - target gate is AND gate: As all predecessors of the target gate are in $\mathsf{InputDepSimGate}$ mode, the input wires to the target gate were generated using **SimPAG**Gate and hence $\mathcal{B}$ holds one wire key on each input wire. Denote these keys by k_a and k_b. It is easy to see that when the oracle outputs evaluations of $\mathcal{F}$, then the result is exactly the same as in the simulated garbling H_1. On the other hand, if the oracle outputs random strings, then the result will contain three ciphertexts out of which all but the one that the color bits are pointing to are random. The last ciphertext (which may be implicitly defined) is computed using $\mathcal{F}$. Hence, the result is exactly the same as in $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$.

Case 2 - target gate is XOR gate: Again, all predecessors of the target gate are in $\mathsf{InputDepSimGate}$ mode and $\mathcal{B}$ holds one wire key on each input wire. If the oracles' output is according to $\mathcal{F}$, then it is easy to see that the result is exactly the same as in the simulated garbling H_1. Otherwise, the oracles output random strings and the result is exactly the same as in $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$.

In both cases, H_1 and $\mathsf{HybInterm}_{\mathcal{A}}^{\lambda}$ are $(T(\lambda), \varepsilon(\lambda))$-indistinguishable. Combining the two steps yields that H_0 and H_1 are $(T(\lambda), 2\varepsilon(\lambda))$-indistinguishable. □

Theorem 1. We are going to show that for any $\alpha \in \{0, \dots, \gamma\}$,

$$\mathbf{D}_{T'(\lambda)}\left[\mathsf{Hyb}(\mathsf{mode}^0, I^\alpha), \mathsf{Hyb}(\mathsf{mode}^\alpha, I^\alpha)\right] \leq \sum_{i=1}^{\alpha} 2^{r_i - s_\alpha + 1} \cdot \varepsilon \leq \alpha \cdot 2^{t - s_\alpha + 1} \cdot \varepsilon.$$

where $s_\alpha = |I^\alpha|$. Observe that $I^\gamma = \emptyset$, and that $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 0) = \mathsf{Hyb}(\mathsf{mode}^0, \emptyset)$ and $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 1) = \mathsf{Hyb}(\mathsf{mode}^\gamma, \emptyset)$. The statement then follows immediately by letting $\alpha = \gamma$.

The proof proceeds by induction on the number $\alpha \in \{0, \dots, \gamma\}$ of pebbling steps taken so far. Denote by H_α the hybrid game $\mathsf{Hyb}(\mathsf{mode}^\alpha, I^\alpha)$ induced by the pebble configuration in step α. Note that we set up the index set I^α such that in each move, rule A (B) applies to the graph of the circuit if and only if rule 1 (2) can be applied to the circuit.

Base case: Let $\alpha = 0$. Then $\mathbf{D}_{T'(\lambda)}[H_0, H_0] = 0$.

Induction hypothesis: Assume that

$$\mathbf{D}_{T'(\lambda)}\left[\mathsf{Hyb}(\mathsf{mode}^0, I^\alpha), \mathsf{Hyb}(\mathsf{mode}^\alpha, I^\alpha)\right] \leq \sum_{i=1}^{\alpha} 2^{r_i - s_\alpha + 1} \cdot \varepsilon \leq \alpha \cdot 2^{t - s_\alpha + 1} \cdot \varepsilon.$$

holds for some arbitrary but fixed $\alpha \in \{0, \dots, \gamma - 1\}$.

Inductive step: We show that the claim holds for $\alpha + 1$. In each move following rule A or B, a black pebble is either added or removed.

```
Adversary $\mathcal{B}$ (Reduction)
Input $\mathsf{mode}$, $I$, target gate index $j$, $\mathsf{step}$, and oracles $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4)$.
 1. (Guesses) For all $w_i \in I$, let $\mathsf{Guess}(w_i) \leftarrow \{0,1\}$.
 2. Receive $C$ from $\mathcal{A}$
    Garble circuit $C$:
 3. (Input Wires) For $i \in [n]$, $\sigma \in \{0,1\}$, $k^{\sigma}_{\mathsf{in}_i} \leftarrow \mathsf{Gen}(1^\lambda)$ and $\pi_{\mathsf{in}_i} \leftarrow \{0,1\}$.
 4. Let $g_j = (\mathsf{Gate}, a^*, b^*, c^*)$, $\alpha = \mathsf{Guess}(a^*)$, $\phi_{a^*} = \alpha \oplus \pi_{a^*}$,
    $\beta = \mathsf{Guess}(b^*)$, $\phi_{b^*} = \beta \oplus \pi_{b^*}$, $\mu^* = \mathsf{Guess}(c^*)$
 5. (Gates) For each $g_i = (\mathsf{Gate}, a, b, c)$, $\mathsf{Gate} \in \{\mathsf{AND}, \mathsf{XOR}\}$, in $C$, except for $g_j$, starting from the input layer
    moving toward the output, do the following:
    (a) If $g_i$ has $a^*$ or $b^*$ as input
        run step 5b but use $\mathcal{O}_2$ (and $\mathcal{O}_4$) instead of running the PRF on $k^{1-\alpha}_{a^*}$ (and $k^{1-\beta}_{b^*}$), otherwise:
    (b) If $\mathsf{mode}_i$ $= \mathsf{RealGate}$,
                 $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i) \leftarrow$    $\mathbf{PAG}\mathsf{Gate}\left(i, \{k^\sigma_a, k^\sigma_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b\right)$.
        else     I. If  $\mathsf{mode}_i =$      $\mathsf{SimGate}$, let $\mu := 0$ else $\mu := \mathsf{Guess}(c)$
                 II. $(k^0_c, k^1_c, \pi_c, \widetilde{g}_i)$  $\leftarrow \mathbf{SimPAG}\mathsf{Gate}\left(i, \{k^\sigma_a, k^\sigma_b\}_{\sigma\in\{0,1\}}, \pi_a, \pi_b, \mu\right)$.
 6. $(k^0_{c^*}, k^1_{c^*}, \pi_{c^*}, \widetilde{g}_j) \leftarrow \begin{cases} \mathbf{ReducOne}^{\mathcal{O}_1,\mathcal{O}_3}_{\mathsf{Gate},\mathcal{B}}\left(j, \{k^\alpha_{a^*}, k^\beta_{b^*}\}, \phi_{a^*}, \phi_{b^*}, \alpha, \beta\right) & \text{if } \mathsf{step} = \mathsf{step1} \\ \mathbf{ReducTwo}^{\mathcal{O}_1,\mathcal{O}_3}_{\mathsf{Gate},\mathcal{B}}\left(j, \{k^\alpha_{a^*}, k^\beta_{b^*}\}, \phi_{a^*}, \phi_{b^*}, \mu^*\right) & \text{if } \mathsf{step} = \mathsf{step2} \end{cases}$.
 7. Send $\widetilde{C}$ to $\mathcal{A}$. Obtain $x$ from $\mathcal{A}$.
    Garble Input $x$:
 8. (Check the guesses) For $\forall\ i \in I$,
    – Let $v(w_i)$ be the bit on the wire $w_i$ during the computation $C(x)$.
    – if $v(w_i) \neq \mathsf{Guess}(w_i)$ Output 0 and abort.
 9. (Choose the colorbits ) Let $y = C(x)$. For $\ell = 1, \ldots, m$:
    Let $i$ be the index of the gate with output wire $\ell$,
    – If $\mathsf{mode}_i \neq \mathsf{SimGate}$, set $\widetilde{d}_\ell = \pi_\ell$
    – else, set $\widetilde{d}_\ell := y_\ell \oplus \pi_\ell$.
10. (Select input keys) For $\ell = 1, \ldots, n$:
    – If all gates $i$ having $\mathsf{in}_\ell$ as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then set $K[\ell] := k^0_{\mathsf{in}_\ell} \| \pi_{\mathsf{in}_\ell}$,
    – else set $K[\ell] := k^{x_\ell}_{\mathsf{in}_\ell} \| (\pi_{\mathsf{in}_\ell} \oplus x_\ell)$.
11. Send $\widetilde{x} := (K, \{\widetilde{d}_\ell\}_{\ell\in[m]})$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.
```

Figure 12: Adversary $\mathcal{B}$ for the $2PRF$ experiment $\mathsf{Exp}^{2PRF}_{\mathcal{F},\mathcal{B}}$ with black-box access to distinguisher $\mathcal{A}$.

Case 1: Move $\alpha + 1$ adds a black pebble. In this case, $s_{\alpha+1} = s_\alpha + 1$ and $r_{\alpha+1} = s_{\alpha+1}$.

$$\begin{aligned}
& \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha+1}), \mathsf{Hyb}(\mathsf{mode}^{\alpha+1}, I^{\alpha+1})\right] \\
\leq\ & \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha+1}), \mathsf{Hyb}(\mathsf{mode}^{\alpha}, I^{\alpha+1})\right] \\
& + \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^\alpha, I^{\alpha+1}), \mathsf{Hyb}(\mathsf{mode}^{\alpha+1}, I^{\alpha+1})\right] && (4) \\
\leq\ & \frac{1}{2} \sum_{i=1}^{\alpha} 2^{r_i - s_\alpha + 1} \cdot \varepsilon + 2\varepsilon && (5) \\
\leq\ & \frac{1}{2} \cdot \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha}), \mathsf{Hyb}(\mathsf{mode}^{\alpha}, I^{\alpha})\right] + 2\varepsilon && (6) \\
=\ & \sum_{i=1}^{\alpha} 2^{r_i - s_{\alpha+1} + 1} \cdot \varepsilon + 2^{r_{\alpha+1} - s_{\alpha+1} + 1} \cdot \varepsilon \\
=\ & \sum_{i=1}^{\alpha+1} 2^{r_i - s_{\alpha+1} + 1} \cdot \varepsilon
\end{aligned}$$

Line 4 follows from the triangle inequality. To get to line 6, use Lemma 4 for

$$\mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha+1}), \mathsf{Hyb}(\mathsf{mode}^{\alpha}, I^{\alpha+1})\right] \leq \varepsilon'$$

for some ε' and $|J| = 1$, as well as Lemma 2 or 3. Applying the induction hypothesis yields line 5. Finally, substitute s_α, rearrange the equation and use that $r_{\alpha+1} = s_{\alpha+1}$ to conclude the case.

Case 2: Move $\alpha + 1$ removes a black pebble. In this case, $s_{\alpha+1} = s_\alpha - 1$ and $r_{\alpha+1} = s_\alpha$.

$$\begin{aligned}
& \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha+1}), \mathsf{Hyb}(\mathsf{mode}^{\alpha+1}, I^{\alpha+1})\right] \\
\leq\ & 2 \cdot \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha}), \mathsf{Hyb}(\mathsf{mode}^{\alpha+1}, I^{\alpha})\right] && (7) \\
\leq\ & 2 \cdot \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^0, I^{\alpha}), \mathsf{Hyb}(\mathsf{mode}^{\alpha}, I^{\alpha})\right] \\
& + 2 \cdot \mathbf{D}_{T'(\lambda)} \left[\mathsf{Hyb}(\mathsf{mode}^\alpha, I^{\alpha}), \mathsf{Hyb}(\mathsf{mode}^{\alpha+1}, I^{\alpha})\right] && (8) \\
\leq\ & 2 \cdot \sum_{i=1}^{\alpha} 2^{r_i - s_\alpha + 1} \cdot \varepsilon + 2 \cdot 2\varepsilon && (9) \\
=\ & \sum_{i=1}^{\alpha} 2^{r_i - s_{\alpha+1} + 1} \cdot \varepsilon + 2^{r_{\alpha+1} - s_{\alpha+1} + 1} \cdot \varepsilon \\
=\ & \sum_{i=1}^{\alpha+1} 2^{r_i - s_{\alpha+1} + 1} \cdot \varepsilon
\end{aligned}$$

Line 6 follows from the previous line by Lemma 4. Line 8 follows from the triangle inequality. Applying the induction hypothesis and Lemma 2 or 3 yields line 9. Finally, substituting s_α, rearranging the equation and using that $r_{\alpha+1} = s_{\alpha+1}$ concludes the case and the proof. □