On Treewidth, Separators and Yao's Garbling

Chethan Kamath[1(✉)], Karen Klein[2], and Krzysztof Pietrzak[3]

[1] Tel Aviv University, Tel Aviv-Yafo, Israel
[2] ETH Zurich, Zurich, Switzerland
[3] IST, Klosterneuburg, Austria
pietrzak@ist.ac.at

Abstract. We show that Yao's garbling scheme is adaptively indistinguishable for the class of Boolean circuits of size S and treewidth w with only a $S^{O(w)}$ loss in security. For instance, circuits with constant treewidth are as a result adaptively indistinguishable with only a polynomial loss. This (partially) complements a negative result of Applebaum et al. (Crypto 2013), which showed (assuming one-way functions) that Yao's garbling scheme cannot be adaptively *simulatable*. As main technical contributions, we introduce a new pebble game that abstracts out our security reduction and then present a pebbling strategy for this game where the number of pebbles used is roughly $O(\delta w \log(S))$, δ being the fan-out of the circuit. The design of the strategy relies on separators, a graph-theoretic notion with connections to circuit complexity.

1 Introduction

Suppose that Alice, who holds a function represented as a Boolean circuit C, and Bob, who holds an input x to that function, want to jointly evaluate $y = \mathsf{C}(x)$ such that Alice learns nothing about x while Bob learns nothing about C (except for some side-information that is unavoidable). Yao put forward[1] the following elegant solution:

1. Alice first sends $\tilde{\mathsf{C}}$, a "garbling" of the circuit C, to Bob,

[1] According to [7], the idea was first presented by Yao in oral presentations on secure function-evaluation [42,43] but formally described only in [21].

Chethan, K—Supported by Azrieli International Postdoctoral Fellowship. Most of the work was done while the author was at Northeastern University and Charles University, funded by the IARPA grant IARPA/2019-19-020700009 and project PRIMUS/17/SCI/9, respectively.
Karen, K—Supported in part by ERC CoG grant 724307. Most of the work was done while the author was at IST Austria funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).
Krzysztof, P—Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

K. Nissim and B. Waters (Eds.): TCC 2021, LNCS 13043, pp. 486–517, 2021.
https://doi.org/10.1007/978-3-030-90453-1_17

2. Bob then obtains $\tilde{x}$, a "garbling" of his input x, from Alice via oblivious transfer,
3. Bob finally evaluates $\tilde{\mathsf{C}}$ on $\tilde{x}$ to learn y and sends it over to Alice.

Yao showed how the garbling steps above can be carried out using a symmetric-key encryption (SKE) scheme (and hence one-way functions). This has been ever since referred to as Yao's garbling scheme, and is the focus of this work. We describe it next in slightly more details.

Yao's Garbling Scheme. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a (special) SKE. To garble a circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^\ell$ with fan-in 2 and arbitrary fan-out:

1. Alice first samples a pair of secret keys (k_w^0, k_w^1) for each wire w in C.
2. For every gate $g : \{0,1\}^2 \to \{0,1\}$ with left input wire u, right input wire v, and output wire w, she then computes a *garbling table* $\tilde{g}$ consisting of the four ciphertexts listed in Table 1(a) in random order.
3. Finally, she constructs the *output mapping* μ which, for each output wire w, maps each of the keys (k_w^0, k_w^1) to the bit it "encodes".

The *garbled circuit* $\tilde{\mathsf{C}}$ consists of all the garbling tables $\tilde{g}$ and the output map μ, Alice sends it over to Bob. This constitutes the *offline* phase of the protocol. To garble an input $x = x_1 \| \ldots \| x_n$, Alice simply gives out, for each input wire w_i, the key $k_{w_i}^{x_i}$ corresponding to the bit x_i. This constitutes the *online* phase of the protocol. To evaluate the garbled circuit on the garbled input, the encryption scheme must satisfy a *special correctness property*: for each ciphertext $c \leftarrow \mathsf{Enc}_k(m)$ there should exist a single key (i.e., k) such that decryption passes. Using the keys in the garbling input, Bob can now evaluate C "over the encryption" as follows:

1. Starting from the input level and in some topological order, he progressively decrypts each garbling table in $\tilde{\mathsf{C}}$ by trying the two keys in hand on all the four ciphertexts for each garbling table. Thus, in each step, he learns one of the secret keys corresponding to the output wire of the gate in consideration.
2. At the end of this process, Bob recovers exactly one of the two keys associated with each output wire of the circuit. This allows him to use the output map μ to "decode" the revealed output keys to the output string $y \in \{0,1\}^\ell$.

The scheme as described above is what is regarded to be the original formulation of Yao's garbling scheme [27,33]. A slight variant in which Alice defers sending the output map μ to the online phase (along with $\tilde{x}$) is also of interest [27], although it suffers from a higher online complexity compared to the original formulation. To avoid confusion, we refer to the original scheme as *Yao's offline garbling scheme* and the modified scheme as *Yao's online garbling scheme* or, in short, Online Yao and Offline Yao respectively. Our work concerns the security of Offline Yao.

Table 1. Garbling tables for (a) general gate g (b) constant-0 gate and (c) constant-1 gate. u and v denote the two input wires and w denotes the output wire. The two keys associated with (say) the wire u are denoted by k_u^0 and k_u^1.

$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{g(0,0)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^0))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^1))$
$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{g(1,0)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^0))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^1))$
$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{g(0,1)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^0))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^1))$
$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{g(1,1)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^0))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^1))$
(a)	(b)	(c)

Security. Even though garbling schemes found several applications (see [7]), its security was formally analysed much later in [33]. They consider a simulation-based notion[2] captured by the following experiment:

1. The adversary submits a circuit-input pair (C, x) to the challenger.
2. The challenger responds either with the real garbling $(\tilde{\mathsf{C}}, \tilde{x})$ (i.e., real game or Real) or with a "simulated" garbling where a constant-0 circuit is used instead of C (i.e., simulated game or Sim). The constant-0 circuit has the same topology as C but with all its gates replaced by constant-0 gates.
3. The adversary wins if it guesses which case it is.

Then they gave a reduction from the (special) indistinguishability of the underlying SKE for *offline* Yao. Note that the adversary in the above security game must *select* the garbling input x at the same time as the circuit C. This is in conflict with the online-offline nature of the actual scheme where Bob (a potential adversary) sees $\tilde{\mathsf{C}}$ *before* he commits to x. Hence Bob could have chosen the input, *adaptively*, based on $\tilde{\mathsf{C}}$. In fact, such a scenario does arise in applications such as one-time programs and secure outsourcing [6]. Therefore it is natural to consider strengthening the above selective definition of simulatability to an adaptive definition where A gets to choose the input after it sees the garbling of a circuit of its choice. Unfortunately, this is too strong a notion to attain for Offline Yao: it was shown in [4] that the *online complexity* of a garbling scheme (or, more generally, a randomised encoding scheme) in the adaptive setting *must* exceed the output-size of the circuit (given that one-way functions exist). Jafargholi and Wichs [27] observed that this negative result does not apply to Online Yao since the output map there gets sent in the online phase, and even managed to prove adaptive *simulatability* of Online Yao. Security of other variants of Yao's garbling scheme was also proved [22,26]. However, the case of Offline Yao was largely ignored.

[2] This is an equivalent formulation of the definition in [33] and is taken from [27]. Our overview of the proof in [33] to be discussed in Sect. 1.2 has been adapted accordingly.

1.1 Our Results

Although the negative result in [4] rules out adaptive *simulatability* of Offline Yao, it is not clear if it also applies to its adaptive *indistinguishability* [7], which is defined by the following experiment:

1. The adversary submits a pair of circuits $(\mathsf{C}_0, \mathsf{C}_1)$ of the same topology to the challenger
2. The challenger. flips a coin b and responds with $\tilde{\mathsf{C}}_b$.
3. The adversary then submits a pair of inputs (x_0, x_1) such that $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ and the challenger responds with $\tilde{x}_b$.
4. The adversary wins if it guesses the bit b correctly.

Although it is a weaker notion of security, adaptive indistinguishability suffices for certain applications (e.g, adaptively-indistinguishable symmetric-key functional encryption [26]).

Table 2. Security of Yao garbling and its variants. The (only) negative result is highlighted in red.

	Selective		Adaptive	
	Offline Yao	Online Yao	Offline Yao	Online Yao
Simulatability	[33]		[4]	[27]
Indistinguishability			This work	

Our Results. We help (partially) complete the landscape for security of Yao's garbling (see Table 2). To this end, we characterise the adaptive indistinguishability of Offline Yao in terms of the treewidth[3] of the circuit. Our main results are informally stated below.

Theorem (main). Consider the class of Boolean circuits $\mathcal{C}$ of size S with treewidth $w = w(S)$. Offline Yao is adaptively indistinguishable for $\mathcal{C}$ with $S^{O(w)}$ loss in security.

For Boolean circuits of constant (resp., poly-logarithmic) treewidth, we obtain the following corollary.

Corollary. Offline Yao is adaptively indistinguishable for Boolean circuits of size S and $O(1)$ (resp., $\mathbf{polylog}(S)$) treewidth with a polynomial (resp., quasi-polynomial) in S loss in security.

[3] Since treewidth is defined for undirected graphs, whenever we refer to the treewidth of a directed graph (or a circuit) we refer to the treewidth of the graph obtained by ignoring the direction of its edges.

Interpreting Our Results. Treewidth is a notion from algorithmic graph theory that has found several applications in parametrised and circuit complexity (see Sect. 1.3) Intuitively, it is a (graph) property that measures how "far" the circuit is from a formula (and, more generally, how far a graph is from a tree): in particular, the smaller the treewidth the closer the circuit is to a formula. Therefore, it is not surprising that having a low treewidth limits how powerful a circuit can be. A precise characterisation of this (from above) was given in [19]: every circuit of size S and treewidth $w = w(S)$ can be simulated in *depth* $w \log(S)$. Thus, e.g, circuits of constant treewidth can be simulated in $\mathbf{NC}^1$. Whether the converse is true in general – i.e., whether $\mathbf{NC}^i$ can be simulated using circuits with treewidth $O(\log^{i-1}(S))$ – is an open problem to the best of our knowledge.[4] However it is partially true: namely, $\mathbf{NC}^1$ circuits can be simulated using polynomial-sized Boolean formulae (which, by definition, have treewidth 1) [13,41]. Consequently, the first corollary applies to functions computable in $\mathbf{NC}^1$.

Given the aforementioned negative result from [4], we find any proof of adaptive security for Offline Yao rather surprising. Nevertheless, there are scenarios where our results also lead to improvements in concrete efficiency (even after the loss in security is taken into account). We describe one such scenario next. Recall from the discussion above that for functions computable in $\mathbf{NC}^1$, we show security of Offline Yao at only a polynomial loss. Moreover, the online complexity of garbling such a function using Offline Yao depends only on the input length n (times the security parameter λ). Now, note that PRGs of arbitrary stretch (say n^c for a constant $c \in \mathbb{N}$) exist in $\mathbf{NC}^1$ [15,23]. However, if one were to use Online Yao, then the online complexity is substantial ($n^c \times \lambda$). This example is particularly interesting since Offline Yao for such a function is not simulatable at all as a consequence of the negative result.

Finally, a remark on the optimality of our upper bound: it was recently shown in [30] that any *black-box* reduction that proves indistinguishability of Offline Yao (or, for that matter, Online Yao) must lose security by a factor that is sub-exponential in the depth of the circuit. Therefore, there remains a gap between the lower bound proved there and the upper bound shown here.

Implications to Simulatability of Online Yao. It is worth pointing out that our results may also imply tighter reductions for simulatability of Online Yao. The reduction for simulatability of Online Yao from [27] loses a factor that is exponential in the *width* of a circuit: our approach can be seen as an extension of their techniques. Since treewidth is bounded from above by width, in cases where there is a gap between treewidth and width for a circuit class, our approach would lead to a tighter reduction for simulatability of Online Yao compared to [27]. A more detailed explanation follows later in Remark 1.

Comparison with [26]. We conclude the section by comparing our result with [26], which is also concerned with adaptively-indistinguishable garbled circuits. The

[4] See this question (48504) posted on CSTheory, Stack Exchange.

construction in [26] builds on [22] and therefore has Offline Yao as its basis. However, it requires (i) applying an additional layer of *somewhere equivocal encryption* to the garbling table and (ii) modifying the circuit to be garbled in order to make the security proof go through. These modifications lead to their construction being less efficient compared to plain Offline Yao, but it does allow them to prove adaptive indistinguishability. It is not clear if any of the ideas employed there can be used to argue the indistinguishability of Offline Yao (this is, in fact, posed as an open question there).

1.2 Technical Overview

Outline. Our starting point is the reduction provinf adaptive simulatability of Online Yao [27]. The key idea in [27] is to abstract out the hybrid argument using a pebble game on the circuit, which we call the black-gray (BG) pebble game (Definition 8). To be precise, they showed that if a circuit allows a BG pebbling strategy of length τ that uses σ (black) pebbles, then there exists a reduction proving adaptive simulatability of Online Yao with a loss in security at most $O(\tau 2^{\sigma})$. This allows us to shift the focus from security reductions to the conceptually-cleaner task of coming up with "pebble-efficient" strategies. We start off below by describing this connection and then explain why this approach falls short when it comes to arguing adaptive indistinguishability (or simulatability) of Offline Yao. Next we show how this issue can be remedied, key to it is a new pebble game, which we call the black-gray-red (BGR) pebble game (Definition 11). Analogous to [27], we prove that if there exists a BGR pebbling strategy of length τ that uses σ ("grayscale", i.e.black or gray) pebbles, then there exists a reduction for adaptive indistinguishability of Offline Yao where the loss in security is at most $O(\tau 2^{\sigma})$ (Theorem 6). Finally to complete the proof – and as our main technical contribution – we describe a pebble-efficient strategy for the BGR pebble game in which the number of (grayscale) pebbles used grows only with the treewidth of the circuit (Theorem 5). The strategy has a divide-and-conquer flavour and crucially relies on the notion of *separators* from graph theory. We next elaborate on each of the steps above.

Pebble Game and Hybrids. The reduction in [27] builds on the reduction for selective simulatability of Offline Yao [33]. Both these works follow a sophisticated hybrid argument which can be described abstractly using a BG pebbling strategy.

Pebbles and Garbling Modes. The BG pebble game (formally defined in Definition 8), as its name suggests, uses two types of pebbles: black and gray. A pebble configuration $\mathcal{P}$ for a circuit C determines how the garbled circuit $\tilde{\mathsf{C}}$ is simulated in the hybrid $\mathsf{H}_{\mathcal{P}}$. To be more precise, the pebble configuration $\mathcal{P}$ can associate each gate g in C with a black or gray pebble. In order to translate $\mathcal{P}$ to the garbling $\tilde{\mathsf{C}}$, the simulator in hybrid $\mathsf{H}_{\mathcal{P}}$ does the following:

- if g carries no pebble in $\mathcal{P}$, then the corresponding garbling table in $\tilde{\mathsf{C}}$ consists of an honest garbling table of g (Table 1(a))
- if g carries a gray pebble, then the garbling table encodes a constant-0 gate (Table 1(b)).
- if g carries a black pebble then the garbling table encodes either a constant-0 or a constant-1 gate (Table 1(c)) *depending* on the value of (the output wire of) g when C is run on the garbling input x.

The three *modes* above of simulating individual gates are named real, simulated and input-dependent modes respectively or, in short, Real, Sim and Input, respectively (Table 3(a)). Note that the real garbling game corresponds to the empty pebble configuration (since all the gates are honestly garbled), whereas the simulated game will correspond to the all-gray configuration (since all the gates have been replaced by the constant-0 gate).

Pebbling Rules. Note that any arbitrary configuration of pebbles $\mathcal{P}$ describes a valid hybrid $\mathsf{H}_{\mathcal{P}}$. The role of the pebbling rules is to model indistinguishability of neighbouring hybrids. To be more precise, if a pebble configuration $\mathcal{Q}$ can be obtained from another configuration $\mathcal{P}$ by a valid pebbling move (or vice versa) then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ should be indistinguishable. Consequently a BG pebbling strategy $\boldsymbol{\mathcal{P}}$, which must start from an empty configuration and end with the all-gray configuration, leads to a valid sequence of hybrids that establishes that the real garbling game and simulated garbling game are indistinguishable, proving the security of the garbling scheme. In the BG pebble game, the following moves (see Fig. 1) are allowed:

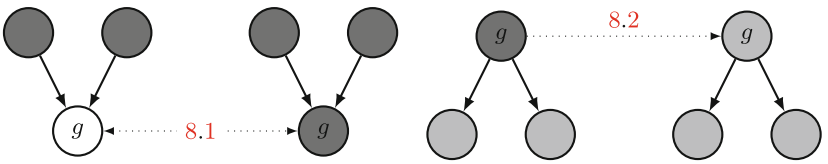

Fig. 1. Rules for BG pebble game.

1. a black pebble can be placed on or removed from a gate g if and only if g's predecessor gates are pebbled black; and
2. a black pebble on a gate g can be replaced by a gray pebble if g's *successor* gates are pebbled, either black or gray.

To understand the rationale behind the two rules, one needs to take a closer look at the structure of a garbling table in Yao's scheme. Since this is not that relevant to the current discussion, we refer the readers interested in more details to Sect. 3.

Selective Simulatability of Offline Yao. Observe that in order to simulate $\tilde{\mathsf{C}}$ in a hybrid $\mathsf{H}_{\mathcal{P}}$, the simulator only needs to know the output value of those gates that are pebbled black in $\mathcal{P}$ (i.e., the gates in Input mode). In the selective setting, since the adversary commits to the garbling input x in the offline phase, the value of *all* the gates is available beforehand. Hence, in this case the simulator has the luxury of using as many pebbles as it needs. Therefore the pebbling strategy (implicitly) employed in [33] is the following:

1. starting from the input gates, pebble the circuit completely black in some topological order, and then
2. starting from the output gates and in reverse topological order, replace each black pebble with a gray pebble.

To complete the description of the hybrid $\mathsf{H}_{\mathcal{P}}$ in the selective setting, one thing remains to be addressed. For concreteness, let's consider the simulated game, which corresponds to the all-gray pebble configuration (the argument for other hybrids is analogous). Note that it is not possible to send the honestly-generated output map μ in $\mathsf{H}_{\mathcal{P}}$ since this will lead to the output being mapped to the all-0 string. However, since x is available in the offline phase, [33] resolved this issue by *programming* the output map to map the zero-keys of the output wires to $\mathsf{C}(x)$. The adversary cannot tell this from the honest output map since the change is information-theoretic.

Since the above pebbling strategy takes at most $2S$ moves (and uses S black pebbles) the corresponding hybrid argument only loses a $2S$ factor. It is possible to further reduce to adaptive simulatability via *random guessing*, but this incurs an additional loss in security that is exponential in the length of x.

Adaptive Simulatability of Online Yao. In order to avoid this exponential loss in the adaptive setting, [27] had to mainly tackle two issues, both arising from the fact that the garbling input x is now only available in the online phase.

1. Firstly, simulating the hybrids could not rely on the knowledge of the values of too many gates in C.
2. Secondly, the output map could no longer be programmed in the offline phase since the output $\mathsf{C}(x)$ is only determined in the online phase.

The first issue was resolved in [27] by employing BG pebbling strategies that were more frugal in terms of the number of black pebbles used. To this end, they proved that if there exists a BG pebbling strategy of length τ that uses σ black pebbles, then the loss in the resulting security is at most $O(\tau 2^{\sigma})$. Here, loosely speaking, the 2^{σ} factor is the cost of randomly guessing the output values of the gates pebbled black, which they require in order to carry out the simulation of the hybrids (as well as the reduction).[5] To complete their proof, [27] described two (generic) pebbling strategies: one where σ grows only with the width of the circuit and another where σ grows only with the depth of the circuit.

[5] This is one of the earliest applications of the piecewise-guessing framework [24].

A consequence of the latter is the adaptive simulatability of log-depth (i.e., $\mathbf{NC}^1$) circuits with a *polynomial* loss in security.

The second issue, on the other hand, was basically side-stepped by *modifying* the garbling scheme to defer the sending of the output map to the online phase, i.e., by resorting to *online* Yao. This tweak allowed [27] to carry out a "deferred programming" of the output map since the garbling input is available in the online phase. The cost is an increased online complexity which is now dependent also on the output size.

Indistinguishability of Offline Yao: Our Approach. Unfortunately, given the negative result from [4], it is unlikely that a result as strong as [27] could be shown for adaptive simulatability of Offline Yao.[6] However, as we will see, relaxing the security requirement to adaptive indistinguishability offers some wiggle room. The key to exploiting this, as we explain next, is to discard the simulated garbling mode (Sim) in the hybrids altogether, which allows us to argue security *without* having to program the output map.

Bypassing the Simulated Mode. A standard way to show that a simulation-based definition implies an indistinguishability-based definition (e.g, think of semantic security and IND-CPA) is to use a two-step hybrid argument where the simulated game acts as an intermediary between the "left" and "right" indistinguishability games. If one attempts to use this approach in our context and use the result from [27] to argue adaptive indistinguishability of *offline* Yao garbling, we immediately run into the issue with programming the output map. Thus it seems that the necessity to program the output map is tied to the simulated game, and hence to the simulated mode of garbling. The main idea behind our reduction is therefore to avoid the simulated mode and instead only work with the real and input-dependent modes, which do not require programming the output map. Thus in all our hybrids, the output map is simply the honestly-generated output map and therefore can be generated in the offline phase itself.

Our Approach. Our idea is to directly replace – gate by gate – the honest garbling table of gates in C_0 (Real_0) with that of gates in C_1 (Real_1). Since the luxury of programming the output map is no longer available, it is crucial to ensure that the evaluation of the garbled circuit in all intermediate hybrids is correct at all times: even though $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ holds (by definition) there is no guarantee that the output of the internal gates of C_0 and C_1 match. An error propagated as a result of one circuit influencing the computation of another may render the hybrids trivially distinguishable to the adversary (via evaluation of the garbling). To this end, we employ the input-dependent modes for (C_0, x_0) and (C_1, x_1) (resp., Input_0 and Input_1). In more details, in all our hybrids, we ensure

[6] Since pseudo-random generators (of arbitrary stretch) exist in $\mathbf{NC}^1$ [15,23], the result in [4] rules out reductions with polynomial loss for *offline* Yao. This is in stark contrast to the aforementioned positive result from [27] for *online* Yao for $\mathbf{NC}^1$ circuits.

that a gate in Real_0 mode is *never* adjacent to another gate in the Real_1 mode. This is accomplished by maintaining a "frontier" of gates in Input_0 and Input_1 mode in between the gates in real mode. This separation of the left (Real_0 and Input_0) and right (Real_1 and Input_1) modes guarantees that the computations belonging to the two circuits do not "corrupt" each other. We point out that this is reminiscent of (circuit) simulation strategies adopted in certain works in circuit complexity [19] (see Sect. 1.3).

The design of our black-gray-red (BGR) pebble game is carried out keeping the above blueprint in mind. Looking ahead, one can think of it as a symmetrised formulation of the BG pebble game. Our proof that a BGR strategy implies a valid sequence of hybrids is mostly similar to that in [27]: we show that if there exists a BGR pebbling strategy of length τ that uses σ grayscale (i.e.black or gray) pebbles, then there exists a reduction to adaptive indistinguishability of Offline Yao with a loss in security at most $O(\tau 2^{\sigma})$ (Theorem 6).[7] The bulk of our technical work goes into coming up with pebble-efficient strategies for the BGR pebble game. This task turns out to be considerably more involved than for the BG pebble game (primarily due to the constraints introduced by the additional rules in the BGR game). The best strategy we could come up with exploits the treewidth w of the circuit, and as a result the number of (grayscale) pebbles used is roughly $\sigma := w\delta \log(S)$, where S is the size of the circuit and δ its fan-out. The strategy has a divide-and-conquer flavour and crucially relies on the notion of separators from graph theory [11,40]. In the remainder of the technical overview, we informally present the BGR pebble game and then briefly explain the treewidth-based BGR strategy.

BGR Pebble Game. Let g denote the location of a gate in $G := \Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$, the directed acyclic graph (DAG) underlying the circuits, and let g_0 (resp., g_1) denote the corresponding gate in C_0 (resp., C_1). The BGR pebble game (formally defined in Definition 11), as its name suggests, uses three types of pebbles: black, gray and red. In order to translate a BGR pebble configuration $\mathcal{P}$ to the garbling $\tilde{\mathsf{C}}$, the simulator in hybrid $\mathsf{H}_{\mathcal{P}}$ does the following for all internal gates g:

- if g carries no pebble in $\mathcal{P}$, then its garbling table in $\tilde{\mathsf{C}}$ will be the honest garbling table of g_0,
- if g carries a black pebble then the honest garbling table will be replaced by that of constant-0 or constant-1 gate *depending* on the output value of g_0 when C_0 is run on x_0,
- if g carries a gray pebble, then the simulation is the same as in previous case except that the garbling *depends* on the output value of g_1 when C_1 is run on x_1,
- if g carries a red pebble, then its garbling table in $\tilde{\mathsf{C}}$ will be the honest garbling table of g_1.

The input is then garbled as follows: For the i-th input gate, if this gate carries no pebble or a black pebble, then the i-th key in $\tilde{x}$ is the key corresponding to the ith

[7] We use the piecewise-guessing framework [24] instead of a direct argument as in [27].

bit of x_0, otherwise it is the key corresponding to the ith bit of x_1. (The pebbles on the output gates are simply ignored.) The four modes of simulation above are real and input-dependent modes for the left and right game respectively or, in short, Real_0, Input_0, Input_1 and Real_1 respectively (see Table 3(b)). Note that the semantics of gates that carry no pebble or a black pebble is the same as in the BG pebble game (if one sets $(\mathsf{C}_0, x_0) = (\mathsf{C}, x)$), but a gray pebble is now interpreted differently. A BGR pebbling strategy starts off with a configuration with all gates empty (i.e., honest garbling of C_0) but the goal is now to pebble them all red (i.e., honest garbling of C_1). Thus the extreme hybrids correspond to the left and right games in the adaptive indistinguishability game. The pebbling rules, listed below (see Fig. 2), are designed keeping the above discussion in mind and so that indistinguishability of neighbouring hybrids can be argued (Lemma 1):

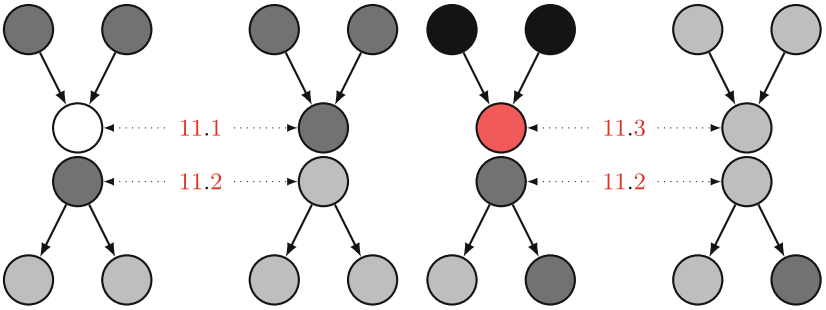

Fig. 2. Rules for BGR pebble game.

1. a black pebble can be placed on or removed from a gate g if and only if g's predecessor gates are pebbled black; and
2. a black pebble on a gate g can be *swapped* with a gray pebble if g's successor gates are pebbled, either black or gray; and
3. a gray pebble on a gate g can be *swapped* with a red pebble if g's predecessor gates are pebbled gray.

Note that the dynamic between no pebbles and black pebbles is similar to the dynamic between red and gray pebbles (hence the reason we consider it to be a symmetric version of the BG pebble game). Since the output values of gates which carry a black or gray pebble in $\mathcal{P}$ need to be known to carry out the simulation of $\mathsf{H}_{\mathcal{P}}$, the goal here is to minimise the number of such "grayscale" pebbles.

Treewidth, Separators and BGR Pebbling Strategies. Compared to the BG pebble game, pebble-efficient strategies for the BGR pebble game are harder to come by. (This is not surprising, in hindsight, given the negative result [4].) In particular, the generic pebbling strategies used in [27] no longer work without incurring

a blow-up in the number of pebbles employed.[8] Below we briefly explain our treewidth-based strategy, the best (generic) strategy we could come up with.

Crucial to our strategy is the notion of *separators*. Informally, a separator for a circuit C of size S is a subset of gates $\mathcal{S}$ such that removing $\mathcal{S}$ (and the edges incident on it) from C partitions C into sub-circuits of "comparable" size. Slightly more formally, $\mathcal{S}$ partitions C into sub-circuits $\mathsf{C}_1, \ldots, \mathsf{C}_p$ such that for every sub-circuit C_i, $|\mathsf{C}_i| \leq 2S/3$ (say). In a classical result from graph theory, it is shown that the size of separator of a graph (and therefore a circuit) is at most its treewidth [11,40]. Since treewidth is a monotonous property – i.e., removing wires or gates from C can only decrease its treewidth – the process of decomposition into sub-circuits using separators can be recursively carried out further (using a different separator each time) till one ends up with constant-size sub-circuits. Such a recursive decomposition is also carried out in the simulation in [19, Theorem 2] (also see [9,35]).

Our pebbling strategy exploits this recursive decomposition to minimise the number of grayscale pebbles used. To this end, the pebbling strategy maintains *long-term* grayscale pebbles only at the separators. These pebbles help reduce the task at hand to that of (recursively) pebbling the resulting sub-circuits, one at a time *reusing* pebbles in that process. Therefore, our pebbling strategy can be recursively described as follows:

- place grayscale pebbles at the separator $\mathcal{S}$ of G,
- recursively, one at a time, place red pebbles on each subcircuit C_i,
- replace the grayscale pebbles on $\mathcal{S}$ with red pebbles.

Since the depth of the recursion is bounded by $O(\log S)$ (thanks to the property of the separator), the hope would be that the number of grayscale pebbles maintained overall does not blow up. We show that this is indeed the case as our main technical contribution (Theorem 5).

Theorem (main). Any circuit C of size S, fan-out δ and treewidth w can be BGR pebbled using $O(\delta w \log(S))$ grayscale pebbles.

Translating the above divide-and-conquer approach into an actual pebbling strategy (Sect. 4) turns out to be tricky due to the intricate nature of the BGR pebbling rules. We refer the readers to Sect. 4 for the details.

Epilogue. It is instructive to review the above pebbling strategy in terms of the actual simulation. The (garbling tables of) circuit C_0 is being progressively, piece by piece, replaced by the (garbling tables of) circuit C_1 as dictated by the recursion, with the bulk of the replacement happening at the base of the recursion. It is exactly those long-term grayscale pebbles placed on the separators which act as the frontier between the pieces of C_0 and C_1. This ensures that computations of the two circuits are insulated from each other.

[8] The width-based BG strategy from [22,27] can be modified to obtain a comparable BGR strategy for *levelled* circuits. However, the resulting security bounds do not yield any advantage over simply guessing the input (which we want to avoid).

Remark 1. We remark that our result on the BGR pebbling complexity can also be used to prove tighter security bounds for simulatability of Online Yao for circuit classes where the treewidth is smaller as the width. This is true since any BGR sequence with complexity σ implies a BG sequence with complexity at most σ: simply consider the BG sequence obtained from a BGR sequence by substituting all the red pebbles with a gray pebble, and note that for BG pebbling only the number of black pebbles is counted.

1.3 Related Work

Garbling. Most of the works on garbling that are relevant to our paper have already been discussed in Sect. 1. In addition to them, [5,20,25,31] pertain to adaptively-secure garbling and are also worth pointing out. Besides, there are several constructions of garbling schemes which aim to exploit structured primitives to improve upon other aspects of garbling like, e.g, online complexity (e.g, [3,12]). We refer the readers to [7] for an excellent exposition on both the historial and technical aspects of Yao's garbling.

Treewidth, Separators and Computational Complexity. Treewidth [40] has its roots in algorithmic graph theory. Many hard graph-theoretical problems become tractable when one restricts to graphs of bounded treewidth. In some cases, this leads to even **NC** algorithms for problems which are otherwise known to be **NP**-complete (e.g, [9]). More often than not, this is because bounding the treewidth leads to divide-and-conquer algorithms, sometimes via separators (see [10] for an instructive survey). Unsurprisingly, this also has several consequences in circuit complexity (e.g, [1,2,34,36]), and perhaps the most relevant to our work are [19,28]. It was shown in [28] that circuits with constant treewidth can be simulated in $\mathbf{NC}^1$; [19] extended this result by using separators to show that circuits of size S and treewidth w can be simulated in depth $w \log(S)$. Both these results can be regarded to be a generalisation of Spira's theorem that Boolean formulae can be simulated by $\mathbf{NC}^1$ circuits [41].

Computing Separators. The problem of computing (balanced) separators in its full generality is **NP**-complete [18,37]; finding *minimal* separators is **NP**-hard [14]. The parameterised complexity of this problem is well-studied and it is **W**[1]-hard (in both the size of the separator and size of the components) [37]. However, when restricted to *constant-degree* graphs, the problem becomes fixed-parameter tractable [37]. For results pertaining to approximation algorithms for computing (balanced) separators, see [16–18].

2 Preliminaries

2.1 Notation

By $[a, b]$, we denote the sequence of integers $a, a+1, \ldots, b-1, b$. All our logarithms are base two.

Notation for Graphs. For a graph $G = (\mathcal{V}, \mathcal{E})$ and a subset $\mathcal{S} \subseteq \mathcal{V}$, $G_{|\mathcal{S}}$ denotes the subgraph of G obtained by *restricting* to the set of vertices in $\mathcal{S}$. That is $G_{|\mathcal{S}} = (\mathcal{S}, \mathcal{E}_{|\mathcal{S}})$ where $\mathcal{E}_{|\mathcal{S}} := \{(u, v) \in \mathcal{E} : u, v \in \mathcal{S}\}$. For a directed graph G, a vertex $u \in \mathcal{V}$ is a predecessor (resp., successor) of another vertex $v \in \mathcal{V}$ if $(u, v) \in \mathcal{E}$ (resp., $(v, u) \in \mathcal{E}$). We say that u is adjacent to v if it is either a predecessor or a successor of v. These definitions can be naturally extended to a set of vertices $\mathcal{S}$ by taking a union over all the vertices in $\mathcal{S}$. The degree δ of a vertex is the number of vertices adjacent to it. The in-degree δ_{in} (resp., out-degree δ_{out}) of a vertex is its number of predecessors (resp., successors). The degree, in-degree and out-degree of a graph is obtained by taking the corresponding maximum over all its vertices.

Notation for Circuits. We consider Boolean circuits with explicit input and output gates, associated with the input and output wires respectively. For a circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^\ell$ with S gates (including the n input and ℓ output gates) and W wires of which n (resp., ℓ) are input (resp., output) wires, we denote the DAG that represents the topology of the circuit C by $\Phi(\mathsf{C})$. That is, $\Phi(\mathsf{C})$ is a graph with $\mathcal{V} = [1, S]$ obtained by:

1. assigning the input (resp., output) gates to the vertices $[1, n]$ (resp., $[S - \ell + 1, S]$),
2. assigning the internal gates to the vertices $[n + 1, S - \ell]$, and
3. assigning the wires of the circuit to the edges.

The wires are assigned an index from $[1, W]$, with the input (resp., output) wires indexed from $[1, n]$ (resp., $[W - \ell + 1, W]$). An internal gate of a circuit is represented by a four-tuple (g, u, v, w) where $g : \{0,1\}^2 \to \{0,1\}$ denotes the predicate implemented, and u, v and w denote the left input, right input and output wires, respectively. We use $V_0(w)$ (resp., $V_1(w)$) as a short-hand for $V_0(\mathsf{C}_0, x_0, w)$ (resp., $V_1(\mathsf{C}_1, x_1, w)$), the function that returns the *value* of the wire w when the circuit C_0 (resp., C_1) is evaluated on the input x_0 (resp., x_1).

2.2 Garbling

The formal definition of syntax and security of garbling schemes is originally from [7]. Our definitions are taken mostly from [26].

Definition 1 (Indistinguishability). *A function $\epsilon : \mathbb{N} \to [0, 1]$ is* negligible *if for every polynomial $p(\lambda)$ there exists an $\lambda_0 \in \mathbb{N}$ such that $\epsilon(\lambda) \leq 1/p(\lambda)$ for all $\lambda \geq \lambda_0$. Let $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ be two distribution ensembles indexed by a security parameter λ. We say that X_λ and Y_λ are $(\epsilon(\lambda), T(\lambda))$-indistinguishable if for any adversary A of size at most $T(\lambda)$,*

$$|\mathsf{Pr}_{a \leftarrow X_\lambda}[\mathsf{A}(a) = 1] - \mathsf{Pr}_{a \leftarrow Y_\lambda}[\mathsf{A}(a) = 1]| \leq \epsilon(\lambda).$$

Definition 2 (Garbling Scheme). *A garbling scheme* **GC** *is a tuple of PPT algorithms* $(\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{GEval})$ *with syntax and semantics defined as follows.*

$(\tilde{\mathsf{C}},K) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C})$. *On inputs a security parameter* λ *and a circuit* $\mathsf{C} : \{0,1\}^n \to \{0,1\}^\ell$, *the* garble-circuit *algorithm* GCircuit *outputs the* garbled circuit $\tilde{\mathsf{C}}$ *and* key K.

$\tilde{x} \leftarrow \mathsf{GInput}(K, x)$. *On input an input* $x \in \{0,1\}^n$ *and key* K, *the* garble-input *algorithm* GInput *outputs* $\tilde{x}$.

$y = \mathsf{GEval}(\tilde{\mathsf{C}}, \tilde{x})$. *On input a garbled circuit* $\tilde{\mathsf{C}}$ *and a garbled input* $\tilde{x}$, *the* evaluate *algorithm* GEval *outputs* $y \in \{0,1\}^\ell$.

Correctness. There is a negligible function $\epsilon = \epsilon(\lambda)$ such that for any $\lambda \in \mathbb{N}$, any circuit C and input x it holds that

$$\mathsf{Pr}\left[\mathsf{C}(x) = \mathsf{GEval}(\tilde{\mathsf{C}}, \tilde{x})\right] = 1 - \epsilon(\lambda),$$

where $(\tilde{\mathsf{C}}, K) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C})$, $\tilde{x} \leftarrow \mathsf{GInput}(K, x)$.

Definition 3 (Adaptive Indistinguishability). *A garbling scheme* **GC** *is* (ϵ, T)*-adaptively-indistinguishable for a class of circuits* $\mathcal{C}$, *if for any probabilistic adversary* A *of size* $T = T(\lambda)$,

$$|\mathsf{Pr}\left[\mathsf{G}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 0) = 1\right] - \mathsf{Pr}\left[\mathsf{G}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 1) = 1\right]| \leq \epsilon(\lambda).$$

where the experiment $\mathsf{G}_{\mathsf{A},\mathbf{GC},\mathsf{S}}(1^\lambda, b)$ *is defined as follows:*

1. A *selects two circuits* $\mathsf{C}_0, \mathsf{C}_1 \in \mathcal{C}$ *such that* $\Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$ *and receives* $\tilde{\mathsf{C}}_b$ *where* $(\tilde{\mathsf{C}}_b, K) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C}_b)$.
2. A *specifies* x_0, x_1 *such that* $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ *and receives* $\tilde{x}_b \leftarrow \mathsf{GInput}(K, x_b)$.
3. *Finally,* A *outputs a bit* b', *which is the output of the experiment.*

In the selective counterpart of Definition 3, the adversary has to select (along with the circuit) the input also in the first step. For self-containment, we provide the definition of selective indistinguishability in Definition 4.

Definition 4 (Selective Indistinguishability). *A garbling scheme* **GC** *is* (ϵ, T)*-selective-indistinguishable for a class of circuits* $\mathcal{C}$, *if for any probabilistic adversary* A *of size* $T = T(\lambda)$,

$$|\mathsf{Pr}\left[\mathsf{H}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 0) = 1\right] - \mathsf{Pr}\left[\mathsf{H}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 1) = 1\right]| \leq \epsilon(\lambda).$$

where the experiment $\mathsf{G}_{\mathsf{A},\mathbf{GC},\mathsf{S}}(1^\lambda, b)$ *is defined as follows:*

1. A *selects two circuits* $\mathsf{C}_0, \mathsf{C}_1 \in \mathcal{C}$ *and two inputs* x_0, x_1 *such that* $\Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$ *and* $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$. *It receives* $(\tilde{\mathsf{C}}_b, \tilde{x}_b)$ *where* $(\tilde{\mathsf{C}}_b, K) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C}_b)$ *and* $\tilde{x}_b \leftarrow \mathsf{GInput}(K, x_b)$
2. A *outputs a bit* b', *which is the output of the experiment.*

Remark 2. A few remarks concerning Definitions 3 and 4 are in order:

1. We call the experiments corresponding to $b = 0$ and $b = 1$ in Definitions 3 and 4 the "left" and "right" experiments, respectively.
2. When the context is clear, we use the simpler notation F^0 and F^1 to denote the experiments $\mathsf{F}_{\mathsf{A},\mathbf{GC},\mathsf{S}}(1^\lambda, 0)$ and $\mathsf{F}_{\mathsf{A},\mathbf{GC},\mathsf{S}}(1^\lambda, 1)$, respectively. Similarly, we use G^0, G^1, H^0 and H^1 for the experiments in Definitions 3 and 4
3. We use $T_{\mathsf{G}} = T_{\mathsf{G}}(\lambda)$ (resp., $T_{\mathsf{H}} = T_{\mathsf{H}}(\lambda)$) to denote the time taken to run experiment G (resp., H).

Offline Yao. We formally describe Yao's original garbling scheme in the full version of the paper [29]. In addition to satisfying the standard notion of security for SKE (IND-CPA), the SKE needs to satisfy the following property for correctness of the garbling schemes to hold.

Definition 5 (Special Correctness [27]). *We say that an SKE* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *satisfies special correctness if for every security parameter* λ*, every key* $k \leftarrow \mathsf{Gen}(1^\lambda)$*, every message* $m \in \mathcal{M}$*, and encryption* $c \leftarrow \mathsf{Enc}_k(m)$*,* $\mathsf{Dec}_{k'}(c) = \perp$ *holds for all* $k' \neq k$ *with overwhelming probability.*

2.3 Pebble Games

In this section, we formally define the pebble games that are relevant to our discussion.

Definition 6 (Reversible black pebble game [8,38]). *Consider a DAG* $G = (\mathcal{V}, \mathcal{E})$ *with* $\mathcal{V} = [1, S]$ *and let* $\mathcal{X}_B = \{\perp, B\}$*. Let* $\mathcal{T} \subseteq \mathcal{V}$ *denote the sinks of* G*. Consider a sequence* $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ *of pebble configurations for* G*, where* $\mathcal{P}_i \in \mathcal{X}_B^{\mathcal{V}}$ *for all* $i \in [0, \tau]$*. We call such a sequence* a reversible black pebbling strategy[9] *for* G *if (i) every vertex is empty in the initial configuration (i.e.,* $\mathcal{P}_0 = (\perp, \ldots, \perp)$*), (ii) every sink is black-pebbled in the final configuration (i.e.,* $\mathcal{P}_\tau(j) = B$ *for all* $j \in \mathcal{T}$*), and (iii) every configuration is obtained by applying the following rule to its preceding configuration:*

1. $\perp \leftrightarrow B$*: a black pebble can be placed on or removed from a vertex if its predecessors are black-pebbled. In particular, a black pebble can be placed on or removed from a source vertex at any time. More formally,* $\exists! j^* \in \mathcal{V}$ *such that*
   - $(\mathcal{P}_{i+1}(j^*) = B$ *and* $\mathcal{P}_i(j^*) = \perp)$ *or* $(\mathcal{P}_{i+1}(j^*) = \perp$ *and* $\mathcal{P}_i(j^*) = B)$*,*
   - $\forall j \in \mathsf{pre}_G(j^*) : \mathcal{P}_i(j) = B$*, and*
   - $\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$*.*

The space-complexity *of a reversible black pebbling strategy* $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ *for a DAG* G *is defined as the maximum number of black pebbles used at any point in the strategy:*

$$\sigma_G(\boldsymbol{\mathcal{P}}) := \max_{i \in [0,\tau]} |\{j \in [1, S] : \mathcal{P}_i(j) = B\}|.$$

Definition 7. *If* $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ *is a black pebbling strategy of space-complexity* σ *for a graph* G*, we say that* $\boldsymbol{\mathcal{P}}$ *is a* (σ, τ)*-strategy for* G*. We say that a class of graphs* $\mathcal{G}$ *has a* (σ, τ)*-strategy if every graph* $G \in \mathcal{G}$ *has a* (σ, τ)*-strategy. Similarly, we say that a class of circuits* $\mathcal{C}$ *has a* (σ, τ)*-strategy if for every circuit* $\mathsf{C} \in \mathcal{C}$*,* $\Phi(\mathsf{C})$ *has a* (σ, τ)*-strategy.*

[9] To be precise, such a pebbling strategy is said to be *persistent* [38] since the final configuration consists of the sinks pebbled. In this paper, we only deal with persistent strategies.

Remark 3. Similar definitions apply to the rest of the pebble games considered in the paper.

Definition 8 (Black-gray (BG) pebble game [22,27]). *Consider a DAG $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = [1, S]$ and let $\mathcal{X}_{BG} = \{\bot, \mathtt{B}, \mathtt{G}\}$ denote the set of colours of the pebbles. Consider a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of pebble configurations for G, where $\mathcal{P}_i \in \mathcal{X}_{BG}^{\mathcal{V}}$ for all $i \in [0, \tau]$. We call such a sequence* a black-gray pebbling strategy *for G if (i) every vertex is empty in the initial configuration (i.e., $\mathcal{P}_0 = (\bot, \ldots, \bot)$), (ii) every vertex is gray-pebbled in the final configuration (i.e., $\mathcal{P}_\tau = (\mathtt{G}, \ldots, \mathtt{G})$) and (iii) every configuration is obtained by applying one of the following rules to its preceding configuration (see Fig. 1):*

1. *$\bot \leftrightarrow \mathtt{B}$: a black pebble can be placed on or removed from a vertex if its predecessors are black-pebbled. In particular, a black pebble can be placed on or removed from a source vertex at any time. More formally, $\exists! j^* \in \mathcal{V}$ such that*
   - *$(\mathcal{P}_{i+1}(j^*) = \mathtt{B}$ and $\mathcal{P}_i(j^*) = \bot)$ or $(\mathcal{P}_{i+1}(j^*) = \bot$ and $\mathcal{P}_i(j^*) = \mathtt{B})$,*
   - *$\forall j \in \mathsf{pre}_G(j^*) : \mathcal{P}_i(j) = \mathtt{B}$, and*
   - *$\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$.*
2. *$\mathtt{B} \mapsto \mathtt{G}$: a black pebble on a vertex $v \in \mathcal{V}$ can be replaced with a gray pebble if v's* successors *are pebbled (either black or gray). In particular, a black pebble on a sink can be replaced by a gray pebble at any time. More formally, $\exists! j^* \in \mathcal{V}$ such that*
   - *$\mathcal{P}_{i+1}(j^*) = \mathtt{G}$ and $\mathcal{P}_i(j^*) = \mathtt{B}$,*
   - *$\forall j \in \mathsf{suc}_G(j^*) : \mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}$, and*
   - *$\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$.*

The space-complexity *of a BG pebbling strategy $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ for a DAG G is defined as the maximum number of black pebbles used at any point in the strategy:*

$$\sigma_G(\boldsymbol{\mathcal{P}}) := \max_{i \in [0,\tau]} |\{j \in [1, S] : \mathcal{P}_i(j) = \mathtt{B}\}|.$$

Remark 4. The rule to place or remove a black pebble in Definition 8 (Rule 8.1) is the same as in Definition 6 (Rule 6.1). Therefore the BG pebble game can be thought of as an extension of the RB pebble game (with a different goal).

2.4 Graph Theory

We recall the definition of treewidth and graph separators, and then state a crucial theorem connecting them, which will be exploited in our pebbling strategy. We emphasise that understanding the definition of treewidth is *not* essential to understanding our pebbling strategies: it is the notion of separators, along with Theorem 1, which is key.

Definition 9 ([11,40]). *A tree decomposition of a graph $G = (\mathcal{V}, \mathcal{E})$ is a tree, T, with nodes $\mathcal{X}_1, \ldots, \mathcal{X}_p$, where each $\mathcal{X}_i \subseteq \mathcal{V}$, satisfying the following properties:*

1. *Each graph vertex is contained in at least one tree node (i.e.,* $\cup_{i\in[1,p]}\mathcal{X}_i = \mathcal{V}$*).*
2. *For every edge* $(v, w) \in \mathcal{E}$*, there exists a node* $\mathcal{X}_i$ *that contains both* v *and* w*.*
3. *The tree nodes containing a vertex* v *form a connected subtree of* T*.*

The width of a tree decomposition is the size of its largest node $\mathcal{X}_i$ *minus one. Its* treewidth $w(G)$ *is the minimum width among all possible tree decompositions.*

Definition 10 ([40])**.** *For a graph* $G = (\mathcal{V}, \mathcal{E})$*, a set* $\mathcal{S} \subseteq \mathcal{V}$ *is said to be a separator if the graph* $G_{|\overline{\mathcal{S}}}$ *has at least two components, and each of these components has size at most* $2|\mathcal{V}|/3$*.*[10]

Theorem 1 ([11,40])**.** *A graph* G *with treewidth* $w(G)$ *has a separator of size at most* $w(G)$*.*

3 Hybrid Argument and the BGR Pebble Game

In this section, we formally show that black-gray-red (BGR) pebbling strategies lead to security reductions for Offline Yao. We start off in Sect. 3.1 by formally defining the BGR pebble game and then explain the semantics of its pebbles, described already (albeit informally) in Sect. 1.2. This enables us to define a hybrid $\mathsf{H}_{\mathcal{P}}$ in terms of a pebble configuration $\mathcal{P}$. Then, in Sect. 3.2, we justify the pebble rules by proving that neighbouring pebble configurations can indeed be proved indistinguishable (Lemma 1). Finally, we put these two steps together in Sect. 3.3 and show that BGR strategies imply adaptive indistinguishability of Offline Yao (Theorem 4) using the piecewise-guessing framework [24]. Since most of the ideas in Sects. 3.2 and 3.3 are similar to pre-existing works [24,27], we skip detailed proofs and resort to high-level sketches.

3.1 Pebble Configurations and Hybrids

The BGR pebble game is a symmetric version of the BG pebble game. In addition to the ones in BG pebble game (Rules 8.1 and 8.2), there are additional rules (Rules 11.2 and 11.3) which govern how the red pebbles interact with the gray pebbles. Intuitively speaking, the dynamic between no pebbles and black pebble (Rule 11.1) is similar to the dynamic between red pebbles and gray pebbles (Rule 11.3): see Remark 5. A more formal definition of the game is given next.

Definition 11 (Black-Gray-Red (BGR) pebble game). *Consider a DAG* $G = (\mathcal{V}, \mathcal{E})$ *with* $\mathcal{V} = [1, S]$ *and let* $\mathcal{X}_{BGR} = \{\bot, \mathtt{B}, \mathtt{G}, \mathtt{R}\}$ *denote the set of colours of the pebbles. A pebble is called* grayscale *if it is black or gray. Consider a sequence* $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ *of pebble configurations for* G*, where* $\mathcal{P}_i \in \mathcal{X}_{BGR}^{\mathcal{V}}$ *for all* $i \in [0, \tau]$*. We call such a sequence* a BGR pebbling strategy *for* G *if (i) every vertex is empty in the initial configuration (i.e.,* $\mathcal{P}_0 = (\bot, \ldots, \bot)$*), (ii) and every vertex is red-pebbled in the final configuration (i.e.,* $\mathcal{P}_\tau = (\mathtt{R}, \ldots, \mathtt{R})$*) and (iii) every configuration is obtained by applying one of the following rules to its preceding configuration (see Fig. 2):*

[10] To be precise, such a separator is called "balanced" [19]. In this paper, we only consider balanced separators.

1. $\bot \leftrightarrow \mathtt{B}$: *a black pebble can be placed on or removed from a vertex if its predecessors are black-pebbled. In particular, a black pebble can be placed on or removed from a source vertex at any time. More formally,* $\exists! j^* \in \mathcal{V}$ *such that*
   - $(\mathcal{P}_{i+1}(j^*) = \mathtt{B}$ *and* $\mathcal{P}_i(j^*) = \bot)$ *or* $(\mathcal{P}_{i+1}(j^*) = \bot$ *and* $\mathcal{P}_i(j^*) = \mathtt{B})$,
   - $\forall j \in \mathsf{pre}_G(j^*) : \mathcal{P}_i(j) = \mathtt{B}$, *and*
   - $\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$.
2. $\mathtt{B} \leftrightarrow \mathtt{G}$: *a black pebble on a vertex* $v \in \mathcal{V}$ *can be swapped with a gray pebble if* v*'s* successors *carry grayscale pebbles (i.e., either black or gray). In particular, a black pebble on a sink vertex can be swapped with a gray pebble at any time. More formally,* $\exists! j^* \in \mathcal{V}$ *such that*
   - $(\mathcal{P}_{i+1}(j^*) = \mathtt{G}$ *and* $\mathcal{P}_i(j^*) = \mathtt{B})$ *or* $(\mathcal{P}_{i+1}(j^*) = \mathtt{B}$ *and* $\mathcal{P}_i(j^*) = \mathtt{G})$,
   - $\forall j \in \mathsf{suc}_G(j^*) : \mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}$, *and*
   - $\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$.
3. $\mathtt{G} \leftrightarrow \mathtt{R}$: *a gray pebble can be swapped with a red if its predecessors are gray-pebbled. In particular, a gray pebble on a source vertex can be swapped with a red pebble at any time. More formally,* $\exists! j^* \in \mathcal{V}$ *such that*
   - $(\mathcal{P}_{i+1}(j^*) = \mathtt{G}$ *and* $\mathcal{P}_i(j^*) = \mathtt{R})$ *or* $(\mathcal{P}_{i+1}(j^*) = \mathtt{R}$ *and* $\mathcal{P}_i(j^*) = \mathtt{G})$,
   - $\forall j \in \mathsf{pre}_G(j^*) : \mathcal{P}_i(j) = \mathtt{G}$, *and*
   - $\forall j \in \mathcal{V} \setminus \{j^*\} : \mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$.

The space-complexity *of a BGR pebbling strategy* $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ *for a DAG* G *is defined as the maximum number of* grayscale *pebbles used at any point in the strategy:*

$$\sigma_G(\boldsymbol{\mathcal{P}}) := \max_{i \in [0,\tau]} |\{j \in [1, S] : \mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}\}|.$$

Remark 5. A few remarks on the BGR pebble game are in order:

1. Note that Rules 11.1 and 11.2 (the $\mathtt{B} \mapsto \mathtt{G}$ part) correspond to the rules in the BG pebble game. The end goals in the two games are however different.
2. When restricted to either black and empty (Rule 11.1) or gray and red pebbles (11.3), the BGR pebble game simplifies to the reversible black pebble game of Bennett [8] defined in Definition 6. This is obvious for the black pebbles since the BGR pebble game is an extension of the BG pebble game which, in turn, is an extension of the reversible black pebble game. To see why this is the case for gray and red pebbles, simply think of vertices with red pebbles as being empty (i.e., $\mathtt{R} = \bot$) and gray pebbles as black pebbles (i.e., $\mathtt{G} = \mathtt{B}$), and note that Rule 11.3 is now the same as Rule 11.1. Therefore, if one starts with an all-red (i.e., empty) configuration, the gray pebbles can be placed using reversible pebbling rules. Some of the reversible pebbling strategies will serve as crucial subroutines in the BGR pebbling strategies in the coming sections.
3. When restricted to black and gray pebbles, the BGR pebble game again simplifies to the reversible pebble game played on the graph with the direction of the edges *flipped*. However, we do not make use of this observation.
4. Only black pebbles can be placed on empty vertices. Gray (resp., red) pebbles have to replace black or red (resp., gray) pebbles, respectively.

5. By the pebbling rules, in any strategy a vertex that is empty can *never* end up adjacent to another vertex with red pebble in any BGR pebbling strategy. Moreover, a vertex with gray (resp., black) pebble cannot be a predecessor of a vertex with no (resp.red) pebble; the converse is however possible. These properties will turn out to be important sanity checks in ensuring the validity of BGR pebbling strategies in the later sections. Moreover, they ensure correctness of the simulations they represent.

Table 3. (a) Garbling modes in [27]. The gate is denoted by g and the value of its output wire w when run on input x is denoted by $V(w)$. (b) Garbling modes in our case. The gates g_0 and g_1 are the gates in the same position in the circuits C_0 and C_1, respectively. The value $V_0(w)$ (resp., $V_1(w)$) denotes the bit going over the wire w in the computation $\mathsf{C}_0(x_0)$ (resp., $\mathsf{C}_1(x_1)$).

	Real ($\perp$)	Input ($\mathtt{B}$)	Sim ($\mathtt{G}$)
$c_{0,0}$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{g(0,0)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{V(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^0))$
$c_{0,1}$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{g(0,1)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{V(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^0))$
$c_{1,0}$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{g(1,0)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{V(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^0))$
$c_{1,1}$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{g(1,1)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{V(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^0))$

(a)

	Real_0 ($\perp$)	Input_0 ($\mathtt{B}$)	Input_1 ($\mathtt{G}$)	Real_1 ($\mathtt{R}$)
$c_{0,0}$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{g_0(0,0)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{V_0(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{V_1(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^0}(k_w^{g_1(0,0)}))$
$c_{0,1}$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{g_0(0,1)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{V_0(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{V_1(w)}))$	$\mathsf{E}_{k_u^0}(\mathsf{E}_{k_v^1}(k_w^{g_1(0,1)}))$
$c_{1,0}$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{g_0(1,0)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{V_0(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{V_1(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^0}(k_w^{g_1(1,0)}))$
$c_{1,1}$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{g_0(1,1)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{V_0(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{V_1(w)}))$	$\mathsf{E}_{k_u^1}(\mathsf{E}_{k_v^1}(k_w^{g_1(1,1)}))$

(b)

Template for Hybrids. A pebble configuration $\mathcal{P} \in \mathcal{X}^{\mathcal{V}}_{BGR}$ is used to encode a selective hybrid $\mathsf{H}_{\mathcal{P}}$. For an *internal gate* v, the translation is carried out as described below:

- if v carries no pebble ($\perp$) in $\mathcal{P}$ then g is garbled as in the left game (Real_0),
- a black pebble ($\mathtt{B}$) on v indicates that the garbling of g is input-dependant on x_0 and C_0, (Input_0)
- a gray pebble ($\mathtt{G}$) on v indicates that the garbling of g is input-dependant on x_1 and C_1 (Input_1)
- a red pebble ($\mathtt{R}$) on v indicates g is garbled as in the right game (Real_1).

The distributions corresponding to the four garbling modes – Real_0, Input_0, Input_1 and Real_1 – are formally defined in Table 3(b). (Note that the semantics of gray

pebbles is different from that in the BG pebble game.) This information is sufficient to construct the garbled circuit $\tilde{\mathsf{C}}$. What remains to complete the description of $\mathsf{H}_{\mathcal{P}}$, is describing how to generate the input garbling $\tilde{x}$ and the output map. If an input gate carries no (resp., a red) pebble then the garbling key for x_0 (resp., x_1) is selected in that hybrid. The output map, on the other hand, is simply the default one prescribed in the scheme and therefore the pebbles on the output gates are ignored. We refer the readers to the full version of the paper [29] for a formal definition of $\mathsf{H}_{\mathcal{P}}$.

Sequence of Hybrids. A pebbling strategy $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_0, \ldots, \mathcal{P}_\tau\}$ will give rise to a sequence of selective hybrids

$$\mathsf{H}^0 = \mathsf{H}_{\mathcal{P}_0}, \ldots, \mathsf{H}_{\mathcal{P}_\tau} = \mathsf{H}^1, \tag{1}$$

Note that the extreme games correspond to the left selective experiment $\mathsf{H}^0 = \mathsf{H}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 0)$ (since $\mathcal{P}_0 = (\bot, \ldots, \bot)$) and right selective experiment $\mathsf{H}^1 = \mathsf{H}_{\mathsf{A},\mathbf{GC}}(1^\lambda, 1)$ (since $\mathcal{P}_\tau = (\mathtt{R}, \ldots, \mathtt{R})$), respectively. The exact pebbling strategy will be discussed later in Sect. 4. In the next section, we prove the indistinguishability of two neighbouring hybrids in such a sequence.

3.2 Indistinguishability of Neighbouring Hybrids

Lemma 1 (neighbouring indistinguishability). *Let $\mathcal{P}$ and $\mathcal{Q}$ denote two neighbouring configurations in a BGR pebbling strategy. If the underlying encryption scheme* **SKE** *is (ϵ, T)-IND-CPA secure, then $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are $(3\epsilon, T - T_{\mathsf{H}})$-indistinguishable, i.e., for any adversary A of size at most $T - T_{\mathsf{H}}$*

$$|\mathsf{Pr}\left[\langle \mathsf{A}, \mathsf{H}_{\mathcal{P}} \rangle = 1\right] - \mathsf{Pr}\left[\langle \mathsf{A}, \mathsf{H}_{\mathcal{Q}} \rangle = 1\right]| \leq 3\epsilon.$$

Proof. Recall that hybrids correspond to pebble configurations and that two neighbouring hybrids differ by a single pebble. We split the proof into three cases which correspond to the pebbling moves $\bot \leftrightarrow \mathtt{B}$, $\mathtt{B} \leftrightarrow \mathtt{G}$ and $\mathtt{R} \leftrightarrow \mathtt{G}$ respectively. The reduction in the first and last cases is similar, and relies on the indistinguishability of the underlying encryption scheme (similar to [27, Lemma 1]). Therefore in the claim below we focus on the first case. In the second case, we argue that the hybrids are identically distributed (similar to [27, Lemma 2]). Moreover, since the proofs are similar to those in [27], we refer the reader to the full version of the paper [29].

Claim (Rule 11.1: $\bot \leftrightarrow \mathtt{B}$). If the underlying encryption scheme **SKE** is (ϵ, T)-IND-CPA secure, and if $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 11.1 then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are $(3\epsilon, T - T_{\mathsf{H}})$-indistinguishable.

Claim (Rule 11.2: $\mathtt{B} \leftrightarrow \mathtt{G}$). If $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 11.2 then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are identically distributed.

□

Selective indistinguishability. Combining Lemma 1 with the semantics of the pebbles (Table 3) yields (via the standard hybrid argument) selective indistinguishability of Offline Yao.

Theorem 2. *Suppose that a class of circuits $\mathcal{C}$ has a (σ, τ)-BGR pebbling strategy. If the encryption scheme* **SKE** *is (ϵ, T)-secure then* $\mathbf{YGC}_{\mathbf{SKE}}$ *is $(3\tau\epsilon, T - T_{\mathsf{H}})$-selectively-indistinguishable for $\mathcal{C}$.*

3.3 Adaptive Indistinguishability via Piecewise Guessing

Observe that in the hybrid $\mathsf{H}_{\mathcal{P}}$, the knowledge of the committed garbling inputs x_0 and x_1 is used to compute the output value of gates that carry grayscale pebbles in the configuration $\mathcal{P}$. So, in principle, the simulation of $\mathsf{H}_{\mathcal{P}}$ can be carried out if this information is available as an "advice". Moreover, the indistinguishability of two successive hybrids can be shown (Lemma 1) if such advice for *both* the hybrids is available. In case the number of grayscale pebbles is small, the size of this advice could potentially be smaller than the size of garbling inputs x_0 and x_1. This means that it is possible to apply the piecewise-guessing framework [24]. We explain this in detail next.

Applying the Piecewise-Guessing Framework. The main theorem in [24] is stated below in Theorem 3 after having been simplified and tailored for our application to circuit garbing. The result of applying Theorem 3 to Offline Yao is stated in Theorem 4. Furthermore, exploiting the properties of the pebbling strategies we design, we provide an optimised version of Theorem 4 later in Sect. 4.2 (Theorem 6).

Theorem 3 (Theorem 2 in [24] tailored to Definitions 3 and 4). *Let G^0, G^1, H^0 and H^1 be as in Definitions 3 and 4. Furthermore, let $\mathsf{H}^0 = \mathsf{H}_{\mathcal{P}_0}, \ldots, \mathsf{H}_{\mathcal{P}_\tau} = \mathsf{H}^1$ be the sequence of hybrids from Equation (1) and suppose that every pebbling configuration $\mathcal{P}_i$ in the strategy $\mathcal{P}_0, \ldots, \mathcal{P}_\tau$ can be computed in time T_p. Assume that for each $i \in [0, \tau - 1]$, there exists a function $\alpha_i \colon \{0,1\}^{1=*} \to \{0,1\}^\sigma$ such that the hybrids $\mathsf{H}_{\mathcal{P}_i}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are (ϵ, T)-indistinguishable when A commits to $\alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1)$ as advice at the beginning of the experiment (instead of (x_0, x_1)). Then G^0 and G^1 are $(\epsilon \cdot \tau \cdot 2^\sigma, T - (T_\sigma + T_p))$-indistinguishable where T_σ denotes the time to sample a string in $\{0,1\}^\sigma$ uniformly at random.*

Theorem 4. *Suppose that a class of circuits $\mathcal{C}$ has a (σ, τ)-BGR pebbling strategy. If the encryption scheme* **SKE** *is (ϵ, T)-secure then* $\mathbf{YGC}_{\mathbf{SKE}}$ *is $(\tau 2^\sigma \cdot 3\epsilon, T - (T_{\mathsf{H}} + T_\sigma + T_p))$-adaptively-indistinguishable for $\mathcal{C}$.*

Proof (Sketch). As already observed, the advice function α_i should return the values of the output wires of all those gates that carry grayscale pebbles in $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$. Therefore, in Theorem 3 we set

$$\begin{aligned} \alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1) := (V_0(w) : (g, u, v, w) \in \mathsf{C}_0 \text{ and } \mathcal{P}(g) = \mathtt{B}) \| \\ (V_1(w) : (g, u, v, w) \in \mathsf{C}_1 \text{ and } \mathcal{P}(g) = \mathtt{G}) \end{aligned} \tag{2}$$

where $\mathcal{P} := \mathcal{P}_{i+1}$ if $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by adding a grayscale pebble; and $\mathcal{P} := \mathcal{P}_i$ otherwise.[11] The length of the advice is therefore smaller than in the selective hybrid in case the pebbling complexity of $G = \Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$ is smaller than the input length. What remains is to show that indistinguishability of two consecutive hybrids can be shown relying only on $\boldsymbol{\alpha}_i := \alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1)$. To see this note that the knowledge of the committed garbling inputs x_0 and x_1 is used to compute the output value of gates that carry grayscale pebbles in the configuration $\mathcal{P}$. Since these are already present in the hint, the reduction algorithm can simply extract these values from $\boldsymbol{\alpha}_i$ and use them instead of explicitly computing $V_0(\cdot)$ and $V_1(\cdot)$. Following the arguments in Lemma 1, we get that if the encryption scheme is (ϵ, T)-secure then the experiments $\mathsf{H}_{\mathcal{P}_i}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are $(3\epsilon, T - T_{\mathsf{H}})$-indistinguishable when A commits to $\boldsymbol{\alpha}_i$, and the proof now follows Theorem 3.

□

4 BGR Pebbling Strategy

In this section, we describe our main strategy for the BGR pebble game. Then, we discuss the implications of our pebbling strategy to the security of Offline Yao (Sect. 4.2).

4.1 BGR Pebbling via Separators

The strategy we describe, BGRSwitch, is implicit in the simulation in [19]. As a consequence of Theorem 1, a graph G with treewidth $w(G)$ can be recursively decomposed using separators of size at most $w(G)$ into smaller and smaller "component" sub-graphs till the sub-graph is of a manageable (constant) size. As a result, one gets a "component tree" out of the graph, starting with the whole graph at the root and ending with manageable-sized sub-graphs as leaves. For a graph with S vertices and degree δ, the depth of the component tree is at most $O(\log S)$ and its out-degree is at most $\delta \cdot |\mathcal{S}|$ (since each vertex in $\mathcal{S}$ can be connected to at most δ components). The pebbling strategy using separators exploits this recursive structure to minimise the number of grayscale pebbles employed.

Remark 6. Note that Theorem 1 does not provide any guarantees on whether such a sequence of separators can be found *efficiently*. This becomes crucial when simulating the hybrids since it determines the factor T_p. We address this question at the end of this section.

[11] Recall from the proof of Lemma 1 that for pebbling configurations $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ that differ by a pebbling move $\mathtt{B} \leftrightarrow \mathtt{G}$, the corresponding hybrids $\mathsf{H}_{\mathcal{P}_1}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are *identically* distributed.

RB Pebbling via Separators. We first describe RBTreewidth, a space-efficient RB pebbling strategy that will be used as a subroutine in BGRSwitch. RBTreewidth places a black pebble on any vertex on a graph G of size S and treewidth w using $\sigma := O(w \log(S))$ pebbles. To the best of our knowledge, this strategy is new and might be of independent interest.[12] Since the strategy is reversible, by $\mathsf{RBTreewidth}^{-1}$ we denote the reverse strategy that *removes* a black pebble. This strategy, thanks to the observation in Remark 5.2, will also be used to both place or remove a gray pebble on an all-red-pebbled graph (RGRTreewidth).

Lemma 2. *Every node in a DAG G with S vertices, in-degree δ_{in} and treewidth w can be black-pebbled following the RB pebbling rules using at most $\sigma := O((\delta_{in} + w)\log(S))$ black pebbles in at most $\tau := (\delta_{in} w)^{O(\log(S))}$ steps.*

Lemma 3. *Starting from the all-red configuration, every node in a DAG G with S vertices, in-degree δ_{in} and treewidth w can be gray-pebbled following the BGR pebbling rules using at most $\sigma := O((\delta_{in} + w)\log(S))$ gray(scale) pebbles in at most $\tau := (\delta_{in} w)^{O(\log(S))}$ steps.*

Proof (of Lemma 2). We denote the pebbling strategy by RBTreewidth and it takes as input a graph (component) C and a vertex v^* to be pebbled. It uses the same recursive decomposition into components as will be in BGRSwitch (i.e., the component tree). The base case is when the graph C is of small enough size (i.e., with $O(1)$ vertices) and here RBTreewidth simply places a black pebble on v^* using as many black pebbles as needed; i.e.:

1. place black pebbles on all vertices in C in topological order (Rule 6.1); and then
2. remove the black pebbles on the ancestors of v^* in reverse topological order (Rule 6.1).

Otherwise, RBTreewidth splits C into smaller components using its separator, recursively places a black pebble on every vertex in the separator in *topological order*, places a black pebble on v^* by recursing on the component C^* that contains v^*. Finally, it recursively removes the black pebbles on the separator in *reverse* topological order. The details are given below.

1. Decomposes C into its components $C_1, \ldots, C_p$ using its separator $\mathcal{S} \subseteq \mathcal{C}$, where $\mathcal{C} = \mathcal{S} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_p$ and $C_i := C_{|\mathcal{C}_i}$.

[12] It is possible to bound the space-complexity of RB pebbling on DAGs of treewidth w using existing results. First, use the fact that the RB pebbling number of a graph of size S is upper bounded by the *plain* black pebbling [39] number with a multiplicative $\log(S)$ factor [32]. Second, use the fact that the black pebbling number is upper bounded by treewidth w (via so-called pathwidth) with another multiplicative $\log(S)$ factor [11, Theorem 2, Corollary 24]. Consequently, we get that the RB pebbling number is at most $w \log^2(S)$. But this is a worse bound compared to what we show directly in Lemma 2.

2. Recursively place black pebbles on the vertices in $\mathcal{S}$ in topological order. That is, for each vertex $s \in \mathcal{S}$ chosen in topological order:
   (a) recursively place a black pebble on each predecessor of s (unless it already carries a black pebble) in topological order,
   (b) place a black pebble on s (Rule 6.1), and
   (c) recursively remove the black pebbles on each predecessor of s that is not in $\mathcal{S}$ in reverse topological order.
3. Recursively pebble the component $C^* \in C_1, \ldots, C_p$ which contains v^*.
4. Undo Item 2 by recursively removing the black pebbles on the separator in reverse topological order. That is, for each vertex $s \in \mathcal{S}$ chosen in reverse topological order:
   (a) recursively place a black pebble on each predecessor of s in topological order,
   (b) remove the black pebble on s (Rule 6.1), and
   (c) recursively remove the black pebbles on each predecessors of s in reverse topological order.

As we explain next, carrying out Items 2 and 4 in topological and reverse topological order, respectively, is crucial for the efficiency (and correctness) of RBTreewidth. Recall that the property of the separator $\mathcal{S}$ guarantees that the components $C_1, \ldots, C_p$ are themselves of small enough size (see Definition 10). Therefore, once $\mathcal{S}$ is pebbled black RBTreewidth can be called on all the resulting components as there are no edges between the components. However, pebbling a vertex s in $\mathcal{S}$ itself is tricky: the predecessors of s could very well be in different components (since there are no guarantees for the vertices in the separator). However, we do have the guarantee that all the predecessors of a predecessor (outside $\mathcal{S}$) of s belong to the same component or the separator, and are reachable via either source vertices *or* vertices belonging to $\mathcal{S}$. Therefore, as long as the vertices in $\mathcal{S}$ are black-pebbled in topological order, $\mathcal{S}$ can be completely pebbled in Item 2 by recursing on small enough components. A similar argument applies when the black pebbles on the separator are removed in Item 4.

With this in mind, we now analyse RBTreewidth. The reason this strategy requires at most $\sigma := O(w \log(S))$ black pebbles is similar to what we will see in the proof of Theorem 5. The number of pebbles is governed by the expression

$$\sigma(i) \leq (w + \delta_{in}) + \sigma(i+1), \tag{3}$$

where the index i is the depth of the recursion of RBTreewidth. The factor $(w + \delta_{in})$ is the cost of black pebbles placed on the separator in Item 2 and the factor $\sigma(i+1)$ is the cost of recursions in Items 2 to 4. Note that since the size of the components in each these recursive calls is at most 2/3 of the size of the original component C, the overall depth of the recursion remains $O(\log(S))$. The upper bound on the number of pebbles claimed in the lemma follows on solving Eq. (3).

As for the number of steps, it is governed by the expression

$$\tau(i) \leq \tau(i+1) O(\delta_{in} w). \tag{4}$$

since RBTreewidth is recursively called at most $O(\delta_{in} w)$ times on the (sub-) components. As in the case of space-complexity, since we end up with constant-size components at the end of the recursion, the base cost is $O(1)$. The lemma follows on solving Eq. (4). □

Recursive Switching. We are now primed to describe BGRSwitch. It takes as input:

1. the original graph $G = (\mathcal{V}, \mathcal{E})$ that is to be pebbled
2. the vertices $\mathcal{C} \subseteq \mathcal{V}$ that define the graph component $C = G_{|\mathcal{C}}$ being currently considered
3. the "higher" separator $\mathcal{U}$, which is the union of all the separators in the "higher" recursive calls that resulted in the creation of the current component C.

Note that $\mathcal{C}$ and $\mathcal{U}$ are disjoint sets by definition. Throughout the execution of BGRSwitch, we maintain a few pebbling properties as invariants:

– At the start of the execution of BGRSwitch on the current component C, it is guaranteed that the vertices in $\mathcal{U}$ are all black-pebbled. This, in some sense, "isolates" C from the rest of the graph and, as a result, it can be pebbled independently of the rest.
– At the end of the execution of BGRSwitch, we guarantee that the vertices $\mathcal{C}$ in C are red-pebbled (via black and then gray), *except* for the children of the higher separator $\mathcal{U}$, which will be left gray-pebbled.

Next, let's see what happens in BGRSwitch when called on $(G, \mathcal{C}, \mathcal{U})$ (in the first call $\mathcal{C} = \mathcal{V}$ and $\mathcal{U} = \emptyset$). The base case is when the current component $C := G_{|\mathcal{C}}$ is of small enough size (i.e., with $O(1)$ vertices). Here BGRSwitch simply switches C to red by using as many pebbles as needed; i.e.:

1. place black pebbles on all vertices in C (Rule 11.1),
2. replace them with gray pebbles (Rule 11.2), and
3. replace the gray pebbles with red pebbles (Rule 11.3) *except* if the vertex is a child of the upper separator $\mathcal{U}$.

Otherwise, BGRSwitch does the switching from no pebbles to red pebbles for C by recursively splitting into smaller components using the separator for C as follows .

1. Decompose $C = G_{|\mathcal{C}}$ into its components $C_1, \ldots, C_p$ using its separator $\mathcal{S} \subseteq \mathcal{C}$, where $\mathcal{C} = \mathcal{S} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_p$ and $C_i := G_{|\mathcal{C}_i}$. (Midpoint-separator analogy)
2. Place black pebbles on the vertices in $\mathcal{S}$ using RBTreewidth. Note that this is possible only because all the vertices that are required to carry this out are either empty or belong to $\mathcal{U}$ and therefore are black-pebbled.
3. Recursively switch each component $C_1, \ldots, C_p$ using BGRSwitch. After all component are switched, all vertices in $\mathcal{C}$, except the ones that are children of $\mathcal{S}$, are red-pebbled; the children of $\mathcal{S}$ are left gray-pebbled.

4. Replace the black pebbles on the separator $\mathcal{S}$ with gray pebbles by using Rule 11.2.
5. Replace the gray pebbles on $\mathcal{S}$ and its adjacent vertices with red pebbles (except if the vertex is a child of the upper separator) using Rule 11.3 and RGRTreewidth.

Note that during the whole strategy, we maintain as invariant a black-gray frontier between the empty and red-pebbled vertices, and this frontier is exactly at the separators. That is, at any point of the pebbling *no* two vertices such that one is empty and the other is red-pebbled are related (see Remark 5.5.). As pointed out in the technical overview (Sect. 1.2), it is this frontier that insulates the computations in the two circuits and help ensure correctness at all times. In the following theorem we formally analyse its space- and time-complexity.

Theorem 5 (Main theorem). *Every DAG G with S vertices, degrees $\delta_{in}, \delta_{out} \leq \delta$ and treewidth w can be BGR-pebbled using at most $\sigma = O((\delta_{in} + w\delta_{out}) \log(S))$ grayscale pebbles in at most $\tau := (\delta w)^{O(\log(S))}$ steps.*

Proof. To bound the space-complexity of BGRSwitch, first note that the algorithm indeed maintains the invariants stated above:

1. At the start of the execution of BGRSwitch on the current component C, it is guaranteed that the vertices in $\mathcal{U}$ are all black-pebbled. Hence, on input a component at depth of recursion $i \in [0, O(\log(S))]$, there are $|\mathcal{U}(i)| = O(iw)$ pebbles that remain black-pebbled.
2. At the end of the execution of BGRSwitch, all the vertices $\mathcal{C}$ in C are red-pebbled, *except* for children of the higher separator $\mathcal{U}$, which will be left gray-pebbled. Hence, after the execution of BGRSwitch on a component at depth $i > 0$ there are up to $\delta_{out} \cdot |\mathcal{U}(i)| = O(i\delta_{out}w)$ many gray pebbles on the graph.

Now, in Item 3 there are up to δw many components, among which some are already switched, some not, and one is currently processed. For the former set of components, all nodes within these which are children of $\mathcal{U} \cup \mathcal{S}$ are pebbled gray. Hence, by the above, there are up to $\delta_{out} \cdot (|\mathcal{U}(i)| + 1) + |\mathcal{U}(i)| = O(i\delta_{out}w)$ nodes that remain gray- or black-pebbled while BGRSwitch is processed on a lower component. Now, while some node on the separator $\mathcal{S}'$ in the currently-processed component is pebbled using RBTreewidth or RGRTreewidth (cf. Item 2), there are up to $|\mathcal{S}'| \leq w$ additional nodes that remain pebbled. By Lemmas 2 and 3, the space-complexity of RBTreewidth/RGRTreewidth is bounded by $O((\delta_{in} + w) \log(S))$. Thus, we arrive at

$$\sigma(i) \leq O(i\delta_{out}w) + w + O((\delta_{in} + w) \log(S)) = O((\delta_{in} + w\delta_{out}) \log(S)).$$

As for the number of steps, on input a component at depth of recursion i, the time-complexity of BGRSwitch is governed by the expression

$$\tau(i) \leq (\delta_{in}w)^{O(\log(S))} \cdot w + \tau(i+1)\delta w. \tag{5}$$

The first factor is the cost of the subroutines used to pebble the separator black: the subroutine is called at most $|\mathcal{S}| \leq w$ times, each time incurring a cost of at most $(\delta_{in} w)^{O(\log(S))}$ (Lemma 2). The second factor is the cost of recursively calling BGRSwitch on at most δw (sub-)components. Since we end up with constant-size components at the end of the recursion, the base cost is $O(1)$. On solving Equation (5), the theorem follows. □

Computing the Separators. Finally, let us return to the question of computing the sequence of separators underlying our pebbling strategy. While we are not aware of an efficient algorithm for computing balanced separators (see discussion in Sect. 1.3), it suffices for our purpose that a separator of size w can be found in time at most $S^{O(w)}$: since we anyway lose a similar factor in the distinguishing advantage, the overall (asymptotic) loss that the reduction incurs remains similar. Therefore, we simply enumerate all w-sized subsets of vertices till we find a balanced separator – note that given a separator it is easy to verify that it is indeed one, i.e., the problem lies in **NP**. Since computing any BGR pebbling configuration requires knowledge of at most $O(\log(S))$ many separators, the total time required to compute a pebbling configuration is at most $T_p = O(\log(S) S^w)$.

4.2 Optimised Piecewise Guessing

Recall that in Theorem 4, the loss in adaptive security is exponential in the BGR pebbling complexity. This is because the reduction requires as advice the value of the output wire of all the gates that are grayscale pebbled. Therefore when Theorem 4 is used in conjunction with Theorem 5, the loss is exponential in the treewidth *as well as* degree. First, note that for Yao's garbling scheme, we only consider Boolean circuits with fan-in 2. We argue next that the dependence on *out-degree* can be removed thanks to the structure of the configurations in the BGRSwitch pebbling strategy. The resulting theorem is stated in Theorem 6.

Let's return to the recursive step in Item 3 which is the cause of dependence on the degree. At the start of this step, all the vertices in the separator $\mathcal{S}$ have been pebbled. Then each component C_i is recursively switched to red one at a time. At the end of switching C_i, each vertex in C_i is pebbled red, *except* for those vertices that are children of $\mathcal{S}$ (or $\mathcal{U}$) which are left gray. Therefore we can restrict our focus on those vertices that have its predecessors in the separator – let's consider one such vertex v^*. Note that in any configuration where v^* carries a gray pebble, it is guaranteed that its predecessors in the separator are black-pebbled. Therefore, instead of requiring the value of the gate g^* corresponding to v^* as an advice, it can simply be computed as a function of the values of its predecessor gates (which *are* included in the advice). To sum up, instead of providing as advice the values of the output wires of *all* the gates that are grayscale pebbled as in Eq. (2), it suffices to provide a much smaller advice as outlined above. As a result of this observation, we get the following optimised version of Theorem 4. This leads to the corollaries stated in Sect. 1.1.

Theorem 6. *Suppose that a class of circuits $\mathcal{C}$ of size S, fan-in 2 has degree δ and treewidth w. If the encryption scheme* **SKE** *is (ϵ, T)-secure then* $\mathbf{YGC}_{\mathbf{SKE}}$ *is $(3\tau 2^{\sigma}\epsilon, T - (T_{\mathsf{H}} + T_{\sigma} + T_p))$-adaptively-indistinguishable for $\mathcal{C}$ where*

$$\tau := (\delta w)^{O(\log(S))}, \quad \sigma := O(w \log(S)) \text{ and } T_p := O(\log(S) S^w).$$

5 Conclusion and Open Problems

Yao's garbling scheme is one of the most fundamental cryptographic constructions. In this work, we took another step towards completing the landscape of its security. Our result leads to several interesting questions, the most natural being whether the upper bound on loss in security can be improved. To this end, one could look at other (orthogonal) graph properties. Another pressing question is whether there are other applications of treewidth in cryptography (which seems relatively overlooked compared to other fields such as circuit complexity or algorithmic graph theory). This closely concerns the divide-and-conquer approach employed in our security reduction: it seems that the approach of surgically replacing one circuit with another should find use in other scenarios. Our hope is that this work spurs further research in this direction.

Acknowledgements. We are grateful to Daniel Wichs for helpful discussions on the landscape of adaptive security of Yao's garbling. We would also like to thank Crypto 2021 and TCC 2021 reviewers for their detailed review and suggestions, which helped improve presentation considerably.

References

1. Alekhnovich, M., Razborov, A.: Satisfiability, branch-width and tseitin tautologies. Comput. Complex. **20**(4), 649–678 (2011)
2. Allender, E., Chen, S., Lou, T., Papakonstantinou, P.A., Tang, B.: Width-parametrized SAT: time-space tradeoffs. Theory Comput. **10**, 297–339 (2014)
3. Ananth, P., Lombardi, A.: Succinct garbling schemes from functional encryption through a local simulation paradigm. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11240, pp. 455–472. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03810-6_17
4. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 166–184. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_10
5. Bellare, M., Hoang, V.T., Keelveedhi, S.: Instantiating random oracles via UCEs. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 398–415. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_23
6. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_10

7. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012: 19th Conference on Computer and Communications Security, pp. 784–796. ACM Press, Raleigh, NC, USA, Oct. 16–18 (2012)
8. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989)
9. Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: van Leeuwen, J. (ed.) WG 1988. LNCS, vol. 344, pp. 1–10. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-50728-0_32
10. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybern. **11**(1–2), 1–21 (1993)
11. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**(1), 1–45 (1998)
12. Boneh, D., et al.: Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_30
13. Brent, R.P.: The parallel evaluation of general arithmetic expressions. J. ACM **21**(2), 201–206 (1974)
14. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is np-hard. Inf. Process. Lett. **42**(3), 153–159 (1992)
15. Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC^0. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 272–284. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44683-4_24
16. Even, G., Naor, J.S., Rao, S., Schieber, B.: Fast approximate graph partitioning algorithms. SIAM J. Comput. **28**(6), 2187–2214 (1999)
17. Feige, U., Hajiaghayi, M.T., Lee, J.R.: Improved approximation algorithms for minimum-weight vertex separators. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, pp. 563–572. ACM Press, Baltimore, MA, USA, May 22–24 (2005)
18. Feige, U., Mahdian, M.: Finding small balanced separators. In: Kleinberg, J.M. (ed.) 38th Annual ACM Symposium on Theory of Computing, pp. 375–384. ACM Press, Seattle, WA, USA, May 21–23 (2006)
19. Gál, A., Jang, J.: A generalization of Spira's theorem and circuits with small segregators or separators. Inf. Comput. **251**, 252–262 (2016)
20. Garg, S., Srinivasan, A.: Adaptively secure garbling with near optimal online complexity. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 535–565. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_18
21. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed) 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press, New York City, NY, USA, May 25–27 (1987)
22. Hemenway, B., Jafargholi, Z., Ostrovsky, R., Scafuro, A., Wichs, D.: Adaptively secure garbled circuits from one-way functions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 149–178. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_6
23. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptol. **9**(4), 199–216 (1996)

24. Jafargholi, Z., Kamath, C., Klein, K., Komargodski, I., Pietrzak, K., Wichs, D.: Be adaptive, avoid overcommitting. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 133–163. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_5
25. Jafargholi, Z., Oechsner, S.: Adaptive security of practical garbling schemes. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) INDOCRYPT 2020. LNCS, vol. 12578, pp. 741–762. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65277-7_33
26. Jafargholi, Z., Scafuro, A., Wichs, D.: Adaptively indistinguishable garbled circuits. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 40–71. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_2
27. Jafargholi, Z., Wichs, D.: Adaptive security of yao's garbled circuits. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 433–458. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_17
28. Jansen, M.J., Sarma, J.: Balancing bounded treewidth circuits. Theory Comput. Syst. **54**(2), 318–336 (2014)
29. Kamath, C., Klein, K., Pietrzak, K.: On treewidth, separators and yao's garbling. Cryptology ePrint Archive, Report 2021/926 (2021)
30. Kamath, C., Klein, K., Pietrzak, K., Wichs, D.: Limits on the adaptive security of yao's garbling. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 486–515. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_17
31. Kitagawa, F., Nishimaki, R., Tanaka, K., Yamakawa, T.: Adaptively secure and succinct functional encryption: improving security and efficiency, simultaneously. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 521–551. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_17
32. Levine, R.Y., Sherman, A.T.: A note on Bennett's time-space tradeoff for reversible computation. SIAM J. Comput. **19**(4), 673–677 (1990)
33. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)
34. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
35. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**(3), 615–627 (1980)
36. Lokshtanov, D., Mikhailin, I., Paturi, R., Pudlák, P.: Beating brute force for (quantified) satisfiability of circuits of bounded treewidth. In: Czumaj, A. (ed.) 29th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 247–261. ACM-SIAM, New Orleans, LA, USA, Jan. 7–10 (2018)
37. Marx, D.: Parameterized graph separation problems. In: Downey, R., Fellows, M., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 71–82. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28639-4_7
38. Nordström, J.: New Wine into Old Wineskins: A Survey of Some Pebbling Classics with Supplemental Results (2015)
39. Paterson, M.S., Hewitt, C.E.: Record of the project mac conference on concurrent systems and parallel computation. Chapter Comparative Schematology, pp. 119–127. ACM, New York, NY, USA (1970)
40. Robertson, N., Seymour, P.D.: Graph minors. II. algorithmic aspects of tree-width. J. Algorithms **7**(3), 309–322 (1986)
41. Spira, P.: On time-hardware complexity of tradeoffs for boolean functions. In: Proceedings of the 4th Hawaii Symposium System Sciences, pp. 525–527. North Hollywood and Western Periodicals (1971)

42. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, pp. 160–164. IEEE Computer Society Press, Chicago, Illinois, Nov. 3–5 (1982)
43. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada, Oct. 27–29 (1986)