Obfuscation Combiners

Marc Fischlin[1] Amir Herzberg[2] Hod Bin Noon[2]
Haya Shulman[3]

[1]Technische Universität Darmstadt, Germany
[2]Bar Ilan University, Israel
[3]Fraunhofer SIT, Darmstadt, Germany

March 15, 2016

Abstract

Obfuscation is challenging; we currently have practical candidates with rather vague security guarantees on the one side, and theoretical constructions which have recently experienced jeopardizing attacks against the underlying cryptographic assumptions on the other side. This motivates us to study and present *robust combiners for obfuscators,* which integrate several candidate obfuscators into a single obfuscator which is secure as long as a quorum of the candidates is indeed secure.

We give several results about building obfuscation combiners, with matching upper and lower bounds for the precise quorum of secure candidates. Namely, we show that one can build 3-out-of-4 obfuscation combiners where at least three of the four combiners are secure, whereas 2-out-of-3 structural combiners (which combine the obfuscator candidates in a black-box sense) with only two secure candidates, are impossible. Our results generalize to $(2\gamma+1)$-out-of-$(3\gamma+1)$ combiners for the positive result, and to 2γ-out-of-3γ results for the negative result, for any integer γ.

To reduce overhead, we define *detecting combiners*, where the combined obfuscator may sometimes produce an error-indication instead of the desired output, indicating that some of the component obfuscators is faulty. We present a $(\gamma+1)$-out-of-$(2\gamma+1)$ detecting combiner for any integer γ, bypassing the previous lower bound. We further show that γ-out-of-2γ structural detecting combiners are again impossible.

Since our approach can be used for practical obfuscators, as well as for obfuscators proven secure (based on assumptions), we also briefly report on implementation results for some applied obfuscator programs.

1 Introduction

Software obfuscation has a long tradition in protection against reverse engineering. For example, the first International Obfuscated C Code Contest (`www.ioccc.org`) has been organized in 1984 and experienced the 23rd event in this series in 2014. There are obfuscators for all popular programming languages today. For example, for Java, there are several open-source projects like ProGuard, ClassEncrypt, or JavaGuard, and an even larger number of commercial products. These approaches are usually based on heuristics and best practices, ranging from simple renaming of function and variables names, to elaborate schemes, e.g. [CT02]. However, these practical obfuscators do not provide verifiable, *proven* security guarantees.

Provably secure obfuscation, in the sense that it is based on some reasonable cryptographic assumption, has long been a highly desirable yet hard-to-reach goal. Even worse, there have been devastating impossibility results for the natural notion of virtual black-box obfuscation [BGI+12] and only limited positive results for special cases like point functions [Can97]. A significant breakthrough came with the work by

Garg et al. [GGH+13], indicating that the relaxed yet useful notion of indistinguishability obfuscation may be achievable for general circuits. This notion basically says that one cannot distinguish the obfuscated codes of two functionally equivalent circuit programs.

It is fair to say that the underlying cryptographic assumption, on which is security of the construction of Garg et al. [GGH+13] is based upon, is non-standard and not well analyzed (yet). This is also true for the alternative approach to build indistinguishability obfuscators proposed by Pass et al. [PST14]. This is complemented by yet other proposals of Gentry et al. [GLSW15] based on a more standard-like computational assumption about multilinear maps, and of Ananth and Jain [AJ15] based on compact functional encryption. At the same time, recent attacks [CHL+15, CLT14, GHMS14, CGH+15] on multilinear maps, albeit currently not known to break the aforementioned obfuscation candidates, testify that constructions may suddenly turn out to lack the desired security guarantees.

The above leaves us with multiple choices of candidates for building obfuscators, both in practice as well as in theory, and it is currently difficult to determine the best choice in terms of security. For the heuristic, practical obfuscators, it may be even harder to distinguish sound constructions from weak approaches, since the design strategies may be vague. A straightforward idea to boost confidence in obfuscator candidates, both in theory as well as in practice, is to interlock multiple solutions and approaches. This idea of failure-tolerant cryptographic designs has traditionally been subsumed under the notion of robust combiners.

1.1 Robust Combiners for Obfuscation

The notion of robust combiners has been introduced by Harnik et al. [HKN+05] based on the idea of tolerant cryptographic designs by Herzberg [Her02, Her05, Her09]. Such combiners take several candidates for a cryptographic task and provide a secure solution if a quorum of the candidates is indeed secure. The idea has been successfully applied to several cryptographic primitives, including hash functions [BB06, Pie07, FL07, Pie08, Mit13, FLP14, MP14], encryption [DK05, HKN+05], commitments [HKN+05, Her02, Her05], and oblivious transfer [HKN+05, MP06, MPW07].

A robust combiner for obfuscation would take as input a program (abstractly in form of a circuit or a Turing machine[1]) and create an obfuscated version with the help of the candidate obfuscators $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_N$. As long as a sufficient number of candidate obfuscators is indeed secure, the combiner should also provide a secure obfuscator. In order to make formal claims about the robustness of the combiner, due to the lack of rigorous security properties for practical obfuscators, one inevitably needs to base the notion of security for the combiner on the various models in the cryptographic literature, such as virtual black-box obfuscation or indistinguishability obfuscation.[2]

What distinguishes the idea of combiners for obfuscation from the previous scenarios is that obfuscation combiners are higher-order combiners which are closely linked to the functionality of their inputs. Consider for instance the case of hash function combiners where it usually suffices that the combiner preserves the security property only, enabling solutions like the concatenation combiner $\mathsf{Comb}^{H_1,H_2}(x) = H_1(x)||H_2(x)$ with longer output for collision resistance. Devising hash combiners with equal output size as H_1, H_2, retaining this mild functional property, is conceivably hard [BB06, Pie07, Pie08]. An obfuscation combiner, in contrast, must provide a circuit which computes the same function as the input circuit; it cannot implement a different function with a larger output. Indeed, note that functional preservation and input hiding are conflicting requirements for obfuscation and one is easy to achieve without the other.

As a concrete example consider combiners in the context of virtual black-box obfuscation. Herzberg and Shulman [HS10] show that the cascading construction $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2}(\cdot) = \mathcal{O}_2(\mathcal{O}_1(\cdot))$ of two candidate

[1] In our presentation of our formal results we focus on circuits instead of Turing machines, since our approach applies equally well to both settings but the state of the art of solutions is much more advanced in the circuit setting.

[2] There are approaches to define metrics for practical obfuscators [CTL97, AMS+07], but mainly in terms of software complexity. We discuss them in Section 7.

obfuscators $\mathcal{O}_1, \mathcal{O}_2$ is robust for this notion *as long as functional correctness of the candidates is guaranteed.* If this is not granted and the inner obfuscator is corrupt then $\mathcal{O}_1$ may implement an arbitrary function, such that the combiner neither preserves functional correctness nor necessarily input hiding. The latter holds as one usually does not have any security guarantees for input circuits with diverging functionalities, even if obfuscator $\mathcal{O}_2$ is sound. Analogously, if the outer obfuscator is corrupt then the resulting cascade may no longer sustain functionality.

While functional correctness of an obfuscator is usually not based on unproven cryptographic assumptions, unlike the obfuscation property, there are two reasons why certifying functional correctness may still be hard. First, software implementations are error prone, and the complexity of previous theoretical proposals for obfuscation [GGH+13, PST14, GLSW15] seems to be inimical in this regard. Secondly, one may have little control over, or insights into, the actual obfuscation program. This is clearly true for commercial obfuscation programs; in fact, the programs of such obfuscators are often themselves obfuscated. The creation of a corrupt obfuscator, which intentionally leaks some information, is easy; to demonstrate this, we implemented demos of different types of corrupted-obfuscators, including obfuscators which leak information even when used in cascade.

The concern about corrupt-obfuscators may also emerge in theoretical solutions. As an example for the latter, in the universal-parameter generation setting [HJK+14] a *trusted* party publishes an obfuscated program which parties can use to generate common parameters. What if we now prefer to use several potentially *untrusted* authorities and combine their obfuscated programs?

1.2 Our Results

Our goal is to provide a general combiner for obfuscation. It should satisfy the formal requirements in order to allow for sound solutions both in theory and in practice. Ideally, the combiner should tolerate a large number of corrupt obfuscators, be very efficient, and ensure various notions of obfuscation simultaneously. Note that, while virtual black-box obfuscation may be impossible in general, for some functions and attack models [BR14, BGK+14] the notion may still be achievable, such that our combiner should also comply with this notion.

On the positive side we present a 3-out-of-4 combiner which can tolerate a single corrupt combiner out of four candidates. It is depicted in Figure 1 and consists of two layers. In the first layer we insert the input circuit C into three combinations of three of the obfuscators each; in each combination, we output a circuit that produces the majority of the three obfuscated circuits. We only require three of the four combinations of picking three of four obfuscators. Each unit ensures that if at most one candidate is corrupt then functional correctness is still preserved. In the next layer we then run each of the first-layer majority circuits through the complementary fourth obfuscation candidate and again take the majority to ensure correctness. Obfuscation follows as either all three candidates on the first layer are sound and thus hide the input circuit, or the fourth candidate on the second layer ensures this.

Our combiner indeed works for different notions of obfuscation such as virtual black-box and grey box obfuscation,[3] indistinguishability obfuscation, and differing-input obfuscation. In total it requires twelve calls to obfuscators and has depth 2. The latter is important as obfuscation may cause a polynomial blow-up in size. Remarkably, while most theoretical solutions currently induce a significant size expansion, with a few exceptions [BV15, AJS], obfuscators in practice only display a mild increase in code size. Note that devising combiners of depth 1 with a structure as above is impossible as the corrupt obfuscator may then leak information about the input circuit via the output.

We then show an impossibility result for 2-out-of-3 combiners. There are, of course, trivial combiners in this case, such as the combiner which simply uses the sound candidate only, and the (inefficient) combiner

[3] With respect to dependent auxiliary inputs [GK05].

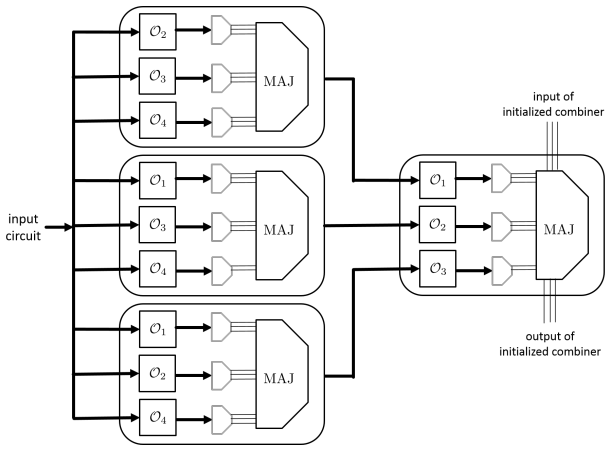

Figure 1: Our 3-out-of-4 combiner. The MAJ circuit has three hardwired circuits C_1, C_2 and C_3 with equal input and output sizes, which also correspond to the input and output size of the MAJ circuit. For input x the MAJ circuit evaluates each of the three circuits for x and returns the bit-wise majority of the circuit's outputs.

for indistinguishability obfuscation that evaluates the input circuit and then outputs the lexicographic smallest equivalent circuit. We thus focus on *structural* combiners that use a fixed pattern, independently of the status of the candidates, and do not semantically interpret the input circuit. Our 3-out-of-4 combiner is structural in this regard. We show that no 2-out-of-3 structural combiner may ensure both functional correctness and obfuscation. This holds for the weaker notion of indistinguishability obfuscation and therefore also for the stronger notions of black-box and grey-box obfuscation. Note that this also applies to any 1-out-of-2 combiner.[4]

We extend the positive result as well as the negative result to the case of $(2\gamma{+}1)$-out-of-$(3\gamma{+}1)$ resp. 2γ-out-of-3γ combiners. That is, we give a construction which can be seen as a less efficient generalization of our basic solution if one can corrupt at most γ out of $3\gamma + 1$ obfuscators. We then argue that one cannot have structural combiners if γ out of the 3γ obfuscators can be corrupt. For both settings we can draw on the the ideas and techniques from the basic cases.

The combiners above are correcting in the sense that they guarantee functional correctness if a quorum of input obfuscator candidates is secure. One can also envision a weaker notion of combiners, which output circuits that either compute the correct output of the input circuit, but may also output, instead, a special error indicator $\perp$. This error indicator should be output only when one of the component obfuscator is faulty; if all obfuscators are sound then the combiner must never output $\perp$. In particular, such combiners cannot output false answers. We call them *detecting combiners* in analogy to coding theory.

For detecting combiners we achieve slightly different bounds. That is, we show that one can have $(\gamma + 1)$-out-of-$(2\gamma + 1)$ combiners for any γ. For the case $\gamma = 1$ and 2-out-of-3 combiners we can again provide an optimized version similar to our original 3-out-of-4 combiner. Concerning lower bounds, we can apply the ideas of the other combiners to show that there cannot exist structural γ-out-of-2γ detecting combiners for any γ. The reduced overhead of detecting combiners may make them attractive option for practical implementations, where once detection ability exists, the attack-vector of providing faulty obfuscator appears unlikely.

While our main results follow the common approach in provably secure obfuscation, we stress that we view our approach to be equally well suited for practice. In Section 7 we therefore evaluate performance of our combiner when applied to practical obfuscators, and discuss the implications of our findings in this domain.

[4]Every 1-out-of-2 combiner is also a 2-out-of-3 combiner if it ignores the third obfuscator.

Concurrent Work. Independently of our work, Ananth et al. [AJN+] also discuss the idea of obfuscation combiners, but also extensions to universal obfuscators and witness encryption. Their approach is fundamentally different, as they basically split the input circuit via secret sharing into n shares, and then let each of the n candidates obfuscate the program of one of n parties carrying out a joint multi-party computation to recover the circuit's output. In their solution this requires specific multi-party protocols and additional cryptographic assumptions such as LWE or DDH, and indistinguishability obfuscation against sub-exponential adversaries.

Compared to our approach, Ananth et al. [AJN+] thus provide (non-structural) obfuscation combiners which do not need nested applications of the candidates, but require the obfuscators to handle programs of the multi-party protocol. Their combiners are secure as long as a single candidate is secure (and the underlying cryptographic assumptions hold). This, however, is based on the presumption that none of the candidates maliciously provides incorrect outputs, whereas we discuss robustness with respect to arbitrary behavior of the candidates (and without further assumptions).

2 Preliminaries

We exclusively treat circuits here; the approach can be transfered to the case of Turing machines straightforwardly. When speaking of circuits C from some class $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$ we usually mean some arbitrary (but efficiently computable) description of the circuit. When considering specific encodings with dedicated properties, as required for our lower bounds, we usually write $\langle C\rangle$ for the encoding of the circuit under scheme $\langle\cdot\rangle$. If, on the other hand, we consider the function implemented by the circuit we usually write $C(\cdot)$ instead, and $C(x)$ for the output of circuit C on input x. When writing $C(\cdot) = C'(\cdot)$ or $C \equiv C'$ we refer to functional equality of circuits C and C', comprising input and output length, whereas $C = C'$ or $\langle C\rangle = \langle C'\rangle$ means equal descriptions (under the encoding in question).

2.1 Obfuscators

Barak et al. [BGI+12] defined several notions of obfuscators, with *virtual black-box* (VBB) obfuscators being the strongest one. This notion says that the adversary cannot learn anything from an obfuscated circuit beyond the circuit's outputs for chosen inputs. While they also showed that this notion is in general unachievable, for specific cases such as point functions one may be able to attain this level of obfuscation. Below we mainly consider obfuscation of circuits, and we also consider the possibility that the obfuscator itself may be non-uniform and work specifically for different values of λ. The latter allows corrupt (also called malicious) obfuscators to match the algorithm class of adversaries and distinguishers. All obfuscators here, sound and corrupt ones, are nonetheless considered to be stateless.

Definition 1 (Virtual Black-Box Obfuscation)**.** *A (possibly non-uniform) PPT algorithm $\mathcal{O}$ is a virtual black-box obfuscator for circuit class $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$ if the following holds:*

Functional Correctness: *For any $\lambda \in \mathbb{N}$, any circuit $C \in \mathcal{C}_\lambda$, any obfuscated version $O \leftarrow \mathcal{O}(1^\lambda, C)$ we have $C \equiv O$.*

VBB Obfuscation: *For any (possibly non-uniform) PPT algorithm $\mathcal{A}$ there exists a (possibly non-uniform) algorithm PPT $\mathcal{S}$ and a negligible function $\epsilon(\lambda)$ such that for all circuits $C \in \mathcal{C}_\lambda$ we have*

$$\Big|\mathrm{Prob}\Big[\mathcal{A}(1^\lambda, \mathcal{O}(1^\lambda, C)) = 1\Big] - \mathrm{Prob}\Big[\mathcal{S}^C(1^\lambda) = 1\Big]\Big| \leq \epsilon(\lambda),$$

where the probabilities are over the randomness of $\mathcal{O}$ and $\mathcal{A}$ resp. $\mathcal{S}$.

Virtual grey-box (VGB) obfuscation [BC10] is defined analogously, only that the simulator above is computationally unbounded but can make at most a polynomial number of queries to its oracle circuit. Clearly, VBB obfuscation implies VGB obfuscation. A stronger notion is based on the extension to *(dependent) auxiliary inputs* [GK05] where both the adversary and the simulator receive a random sample aux as additional input, where aux may depend on any circuit $C' \in \mathcal{C}_\lambda$.[5] We will use this version for proving the security of our combiners for VBB and VGB obfuscation.

Another meaningful relaxation, implied by both notions above in the non-uniform setting, is *indistinguishability obfuscation* [BGI+12] which basically says that the obfuscations of two functional equivalent circuits are indistinguishable:

Definition 2 (Indistinguishability Obfuscator)**.** *A (possibly non-uniform) PPT algorithm $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$ if the following conditions hold:*

Functional Correctness: *For any $\lambda \in \mathbb{N}$, any circuit $C \in \mathcal{C}_\lambda$, any obfuscated version $O \leftarrow i\mathcal{O}(1^\lambda, C)$ we have $C \equiv O$.*

Indistinguishability: *For any (possibly non-uniform) PPT distinguisher $\mathcal{D}$, there exists a negligible function $\epsilon(\lambda)$ such that for all circuits $C_0, C_1 \in \mathcal{C}_\lambda$ with $C_0 \equiv C_1$ we have*

$$\Big|\mathrm{Prob}\Big[\mathcal{D}(1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_0)) = 1\Big] - \mathrm{Prob}\Big[\mathcal{D}(1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_1)) = 1\Big]\Big| \leq \epsilon(\lambda),$$

where the probabilities are over the randomness of $i\mathcal{O}$ and $\mathcal{D}$.

There are several variations of the above definitions. For one, we can allow for a negligible error in the functional correctness (over the random choices of the obfuscator). Both our positive and our negative result are robust with respect to such a change. That is, our 3-out-of-4-combiners uses a constant number of obfuscator calls such that the error would remain negligible; obfuscation would still hold, because the leakage due to incorrect obfuscator outputs has negligible probability. Similarly, our impossibility result about 2-out-of-3 combiners would still hold, even if the starting combiners would have perfect functional correctness, but the (fixed-size structural) combiner could have a negligible error. Alternatively, one may use the recent approach in [BV] to eliminate the error first.

Finally, yet another version of obfuscation, called *differing-inputs* obfuscation [BGI+12], demands indistinguishability of two obfuscated circuits, but only if the input circuits C_0, C_1 can be sampled such that finding inputs where C_0 and C_1 differ, is infeasible. More formally, we assume that there is a PPT algorithm $\mathsf{Sampler}$ associated to the circuit family $\mathcal{C}$ such that for any PPT algorithm $\mathcal{A}$ there exists a negligible function $\epsilon(\lambda)$ such that the probability that $C_0(x) \neq C_1(x)$, where $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda)$ and $x \leftarrow \mathcal{A}(1^\lambda, C_0, C_1, \mathsf{aux})$, is at most $\epsilon(\lambda)$. Note that we assume that $\mathsf{Sampler}(1^\lambda)$ only outputs circuits $C_0, C_1 \in \mathcal{C}_\lambda$,

A differing-inputs obfuscator $\mathrm{di}\mathcal{O}$ for $\mathcal{C}$ and $\mathsf{Sampler}$ is now defined analogously to an indistinguishability obfuscator, only that it is infeasible to distinguish outputs $\mathrm{di}\mathcal{O}(1^\lambda, C_0)$ and $\mathrm{di}\mathcal{O}(1^\lambda, C_1)$ for sampled $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda)$, even if given aux as additional input. While the notion is also quite useful for the design of protocols [ABG+13], Garg et al. [GGHW14] argue that the notion may be hard to achieve.

2.2 Combiners for Obfuscators

Roughly, a combiner for obfuscators is a procedure which uses a set of obfuscators $\mathcal{O}_1, \mathcal{O}_2, \ldots$ to turn an input circuit C into an obfuscated one, with the guarantee that if an (unspecified) quorum of the underlying obfuscators is secure, then so is the combiner. In the definition below we abstractly speak of o-obfuscators, leaving open which obfuscation category $o \in \{\text{VBB,VGB,indistinguishability, differing-inputs}\}$ we refer to.

[5] This slightly strengthens the original auxiliary input setting [GK05] where only $C' = C$ is allowed.

For combiners of primitives with multiple properties, such as functional correctness and obfuscation here, there are varying levels of combiners, called weak, mild, and strong [FL08, FLP14]. A strong combiner preserves security "property-wise", i.e., for each property individually if sufficiently many candidates have this property then so does the combiner. A weak combiner only preserves all properties if there are enough candidates which are secure and thus have all properties simultaneously. The mild notion is in between where the candidates must somehow cover all properties but for each property possibly by different candidates. In [FL08, FLP14] it has been discussed that strong robustness implies mild robustness which in turn implies weak robustness, and that the implications are strict in case of hash functions for some properties.

Definition 3 (Robust Combiner for o-Obfuscation)**.** *Let* Comb *be a PPT oracle algorithm and let* $\mathcal{O}_1, \ldots, \mathcal{O}_N$ *be o-obfuscators candidates. Then* Comb *is called a*

- *strongly robust t-out-of-N combiner if for each of functional correctness and o-obfuscation, if at least t of the N candidates have this property, then so does the combiner* $\mathsf{Comb}^{\mathcal{O}_1,\ldots,\mathcal{O}_N}$*;*
- *mildly robust t-out-of-N combiner if, whenever functional correctness and o-obfuscation are each satisfied by at least t of the N candidates, then the combiner too has both properties;*
- *weakly robust t-out-of-N combiner if the combiner is a functional correct o-obfuscator if there are at least t out of N candidates which are simultaneously functionally correct and o-obfuscators.*

The definition assumes that the obfuscators and combiner all work for the same class $\mathcal{C}$ of obfuscatable circuits. This neglects an important aspect, though: If the combiner calls obfuscators recursively then the candidates need to be able to handle obfuscated circuits, too. We assume that this is indeed the case —and discuss it more explicitly for our structural combiners below— making the implicit assumption that the candidates also allow for a superclass $\mathcal{C}^{\mathsf{Comb}}$ of circuits which is rich enough tor capture intermediate circuits created by the specific combiner. Still, the task for the combiner is to obfuscate the "core" class $\mathcal{C}$ of circuits.

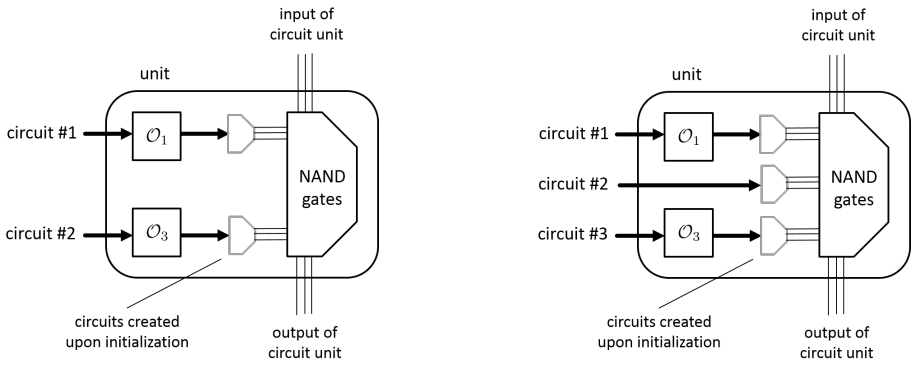

Figure 2: Example of a unit (with pass-through version on the right-hand side).

As usual for combiners in general, there is always a secure obfuscation combiner, namely, the one which "obliviously" uses the secure obfuscator $\mathcal{O}_i$ and ignores the other ones in order to obfuscate the input circuit. However, this only provides an existential proof and says nothing about how to design an actual solution. Even worse, for indistinguishability obfuscation there is a trivial (non-efficient) combiner for obfuscators which can be described effectively [BGI+12]. The combiner takes as input (the description of) a circuit C and finds the (lexicographically) minimal circuit $C_{\min}$ which computes the same functionality as C and outputs this circuit $C_{\min}$. Then any two circuits C, C' with the same functionality yield the

same obfuscated circuit $C_{\min}$. This combiner ignores the candidate obfuscators and already constitutes an unconditionally secure obfuscator itself. It is even efficient relative to a Σ_2^p oracle. Hence, any lower bound for combiners would need to bypass this result and therefore need to implicitly show that $\Sigma_2^p \neq P$.

One option to circumvent the first problem is to require to have an effective mean to turn attacks against the combiner into attacks for the candidate obfuscators. This option of so-called *black-box combiners* has been used for other lower bounds such as for hash function combiners [BB06, Pie07, Pie08]. Still, in our setting such black-box combiners would have to deal with the problem of the inefficient combiner.

An alternative path, which we also take here, is therefore to restrict the way how the combiner works. Whereas the above unconditional combiner approaches the circuit *semantically* by plotting its behavior, we look into what we call *structural* combiners here. Basically, these are combiners which have a prescribed structure with place-holder gates for the obfuscators, and they merely plug in the input circuit and derive the output circuit according to this fixed structure, without evaluating the circuits. It turns out that our 3-out-of-4 combiner is in fact structural.

2.3 Structural Combiners

A structural combiner for obfuscators is a circuit consisting of NAND gates and of obfuscator gates, where each one of the latter is labeled with one of the obfuscators $\mathcal{O}_i$. The layout is independent of the actual obfuscators and should thus work with any concrete obfuscator candidates, i.e., be black-box. The combiner is structured in so-called units. A *unit* is a sub circuit which takes as input the descriptions of circuits and itself describes a circuit. The unit first inserts the input circuits into some of the obfuscators, where we allow multiple appearances of obfuscators in a unit, and then processes the output circuits by a circuit consisting of NAND gates only. An example is given in the left part of Figure 2. If the input circuit is given to obfuscators $i_1, i_2, \dots$ then we call this an $\{i_1, i_2, \dots\}$-unit for the multiset $\{i_1, i_2, \dots\}$. The example in Figure 2 describes a $\{1, 3\}$-unit. Furthermore, we can even let some input circuit be passed to the NAND-circuit completely, saying that the unit is *pass-through* in this case. Since it is irrelevant for our lower bound which circuit is passed through, we do not need to specify the identifier. The right hand side of Figure 2 shows a pass-through version of a $\{1, 3\}$-unit.

The output of a unit can itself serve again as the input for another unit. We can therefore nest units in a tree-like structure as in Figure 3. In particular, we can analogously to the notion of depths of circuits define the depth of a unit, starting with level-1 units, as well as paths from the input circuit to the final unit. We call the path of units form level-1 units to the final unit a full path. A unit which is level-1 always receive the combiner's input circuit as inputs, but potentially also other unit circuits if it is simultaneously a higher level unit. Every unit has at least one input circuit, and a unit can of course serve as multiple inputs to other units.

To complete the description of a structural combiner we need to specify the output of our combiner for some input circuit C, once the obfuscator candidates are determined. We call this the *initialization* of the combiner with C. Basically the output is again a circuit and it is derived by stepwise replacing the obfuscator gates in units (starting with level-1 units which receive C as input) with samples of the output of the corresponding obfuscator. Note that the structure of the combiner circuit remains, only the obfuscator gates are now filled in with concrete circuits. In case of pass-through units we additionally place the code of the unit's input circuit inside the new circuit at the corresponding position. Once a unit has been initialized we can use it as input to a higher-level unit and initialize that unit, till we have eventually initialized the final unit. Instructively, the reader may think of this as a left-to-right pass in Figure 3 to compute the final output circuit, denoted as as $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\cdots}(C)$. Note that this is a random variable, depending on the randomness of the obfuscators. A sample of this random variable can then be fed with inputs x to produce some output y.

The above assumes that the class of obfuscatable circuits for structural circuits is closed under recursive

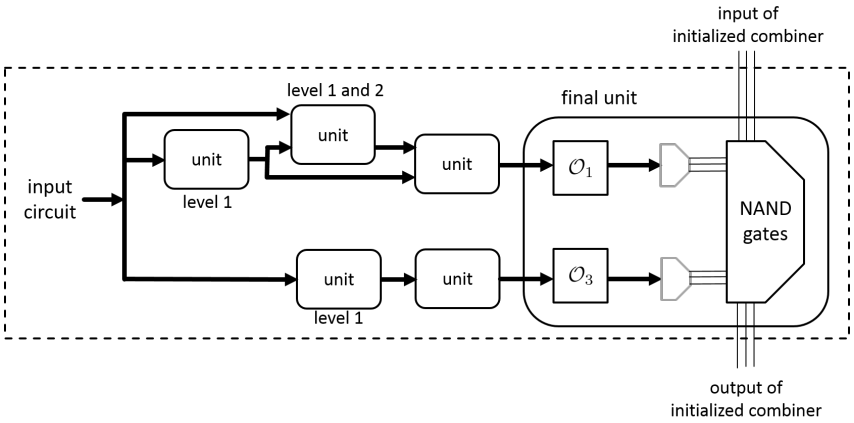

Figure 3: Combiner circuit consisting of units.

constructions of units. We note that for concrete constructions such as our 3-out-of-4 combiner in the next section it suffices that we can also obfuscate level-1 units of the original input circuits. Given a circuit class $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$, some fixed structural combiner Comb, and fixed obfuscators $\mathcal{O}_1, \mathcal{O}_2, \ldots$ we denote by $\mathcal{C}^{\mathsf{Comb}} = (\mathcal{C}_\lambda^{\mathsf{Comb}})_{\lambda\in\mathbb{N}}$ the class of circuits which, besides all circuits $C \in \mathcal{C}_\lambda$, for any C also includes all possible initializations of all units of the combiner (except for the final unit) for the given obfuscators. It is understood that, when considering a specific combiner Comb, all candidate obfuscators $\mathcal{O}_1, \mathcal{O}_2, \ldots$ must be able to handle the class $\mathcal{C}^{\mathsf{Comb}}$, whereas the combiner only works for the "inner" class $\mathcal{C}$. Instructively, one may think of $\mathcal{C}$ as the class one would like to obfuscate, although the candidate obfuscators allow for broader classes.

3 Robust 3-out-of-4 Combiner for Obfuscators

In this section we present a 3-out-of-4 (structural) combiner for obfuscation, depicted in Figure 1 on Page 4.

3.1 Construction

The idea is to first obfuscate the input circuit C by all combinations of 3 out of the 4 given obfuscators $\mathcal{O}_1, \ldots, \mathcal{O}_4$ and for each combination taking the majority of the output of the three obfuscated circuits. Note that since at least 2 of the 3 obfuscators in such a combination work properly, the majority decision provides a functionally correct output. Formally, for the majority circuit MAJ combining three input circuits by evaluating each one for a given input x and taking the bit-wise majority of the outputs, we thus build the circuits

$$O_{i_1,i_2,i_3} \leftarrow \mathrm{MAJ}(\mathcal{O}_{i_1}(C), \mathcal{O}_{i_2}(C), \mathcal{O}_{i_3}(C)), \quad 1 \le i_1 < i_2 < i_3 \le 4$$

for all possible 4 combinations of i_1, i_2, i_3. Since we merely need an arbitrary 3 of these 4 circuits for the next stage, we take the combinations leaving out obfuscators $1, 2$ and 3 (in this order).

Of course, a corrupt obfuscator among $\mathcal{O}_{i_1}, \mathcal{O}_{i_2}, \mathcal{O}_{i_3}$ in the majority combination could still reveal information about the input circuit C. We hence add another layer where we now combine three of the majority combinations as before, by running each combination O_{i_1,i_2,i_3} through the complementary obfuscator $\mathcal{O}_{i_4}$ and taking the majority of these circuits again. Put differently, we now build the circuit

$$\mathrm{MAJ}(\mathcal{O}_1(O_{2,3,4}), \mathcal{O}_2(O_{1,3,4}), \mathcal{O}_3(O_{1,2,4})).$$

Functional correctness of our combiner is guaranteed because each of the input circuits $O_{2,3,4}, O_{1,3,4}, O_{1,2,4}$ computes the correct function and at least two of the level-2 obfuscators $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ are correct. The obfuscation property holds because if one of the level-2 obfuscators, say, $\mathcal{O}_1^*$, is malicious, then the level-1 obfuscators generating $O_{2,3,4}$ already hide the input circuit. Furthermore, the malicious obfuscator $\mathcal{O}_1^*$ cannot bias the functional correctness of the circuits $O_{1,3,4}$ and $O_{1,2,4}$ in the other branches, such that the sound second-layer obfuscators $\mathcal{O}_2, \mathcal{O}_3$ also hide $O_{1,3,4}$ and $O_{1,2,4}$ and thus the input circuit C, even if $\mathcal{O}_1^*$ on the first level reveals information about C.

3.2 Security

We start by showing that the combiner is (strongly) robust for indistinguishability obfuscation. Recall that strong robustness refers to the fact that each property, functional correctness and obfuscation, is preserved individually. Note that for our combiner (and also the security proof) it suffices that the parties merely have black-box access to all obfuscators.

Theorem 1. *The combiner in Figure 1 is a strongly robust* 3*-out-of-*4 *combiner for indistinguishability obfuscation.*

Proof. Functional correctness is straightforward, given that for each unit at least two obfuscators are functionally correct and since we apply the majority of the outputs.

We next show indistinguishability. Take an arbitrary distinguisher $\mathcal{D}$ against our combiner. We need to show that there exists a negligible function ϵ such that for an arbitrary pair $C_0, C_1 \in \mathcal{C}_\lambda$ of circuits, the distinguishing advantage of $\mathcal{D}$ is smaller than $\epsilon(\lambda)$. The idea is to show that one can gradually replace the input circuits C_0 to the obfuscators in the combiner by circuit C_1, taking some care with the single corrupt obfuscator.

For the gradual replacement fix the order of the nine level-1 obfuscators $\mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \ldots, \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_4$ according to their appearance in Figure 1 from top to down, with one exception: for a parameter $k \in \{1,2,3,4\}$, a reminiscent for the index of the corrupt obfuscator $\mathcal{O}_k^*$, we move all occurrences of this obfuscator to the very end of the list. For instance, for $k = 2$ we would have the order $\mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_1, \ldots, \mathcal{O}_4, \mathcal{O}_2^*, \mathcal{O}_2^*$. Let $K = K(k) \in \{7,8\}$ be the first index of $\mathcal{O}_k^*$ in that list. Define now the random hybrid variables $H_i^k(C_0, C_1)$ for $i = 0, 1, \ldots, 9$ as the output of our combiner if we pass circuit C_0 for the first i obfuscators (according to our order) and C_1 for the remaining $9 - i$ ones. Then, clearly $H_9^k(C_0, C_1)$ corresponds to the distribution of our combiner for input C_0, and $H_0^k(C_0, C_1)$ to the one of our combiner for C_1. It hence suffices to show for any i that $\mathcal{D}$'s probability of distinguishing adjacent H_{i-1}^k, H_i^k is negligible.

To bound the advantage of $\mathcal{D}$ for each pair (H_{i-1}^k, H_i^k) we will wrap the algorithm into a sequence of distinguishers $\mathcal{D}_i^k$ for $i = 1, 2, \ldots, 9$. The distinguisher $\mathcal{D}_i^k$ works in two modes, depending on the status of the i-th obfuscator in our sequence:

- If i is such that the i-th obfuscator is not corrupt, i.e., $i < K(k)$, then $\mathcal{D}_i^k$ expects as input a pair C_0, C_1 and an obfuscated circuit O' generated by the i-th obfuscator in our order for C_b, $b \in \{0,1\}$. Algorithm $\mathcal{D}_i^k$ computes the output of our combiner (with the given obfuscators) but inserts C_1 as input in the first $i - 1$ level-1 obfuscators, O' as the *output* of the i-th level-1 obfuscator, and C_0 as input in the final $9 - i$ slots. It completes the output O of the combiner for these data and lets $\mathcal{D}$ run on C_0, C_1 and O. Algorithm $\mathcal{D}_i^k$ returns whatever $\mathcal{D}$ outputs.

- If the i-th obfuscator is corrupt, i.e., $i \geq K(k)$, then $\mathcal{D}_i^k$ expects as extra auxiliary input a pair C_0, C_1 and a sample O' of one of the sound obfuscator candidates. Here the obfuscator $\mathcal{O}_j$ producing O' is determined by looking at the level-1 unit u in which the i-th (corrupt) obfuscator $\mathcal{O}_k^*$ appears. For this unit, and its three obfuscators, there exists the fourth, complementing obfuscator $\mathcal{O}_j$ to which

the unit's output is fed to on the level-2 unit. For instance, if $k = 2$, $K = 8$, and $i = 8$, then the corresponding level-1 unit u is the top one in Figure 1, and the complementing obfuscator is $\mathcal{O}_1$.

The input to the complementing obfuscator $\mathcal{O}_j$ for deriving O' is either a sample of the level-1 unit where all honest obfuscators are initialized with C_0 and the corrupt one with C_1, or all of them are initialized with C_0. By assumption, both samples are in the class $\mathcal{C}_\lambda^{\mathsf{Comb}}$ such that the sample can be passed to $\mathcal{O}_j$. Algorithm $\mathcal{D}_i^k$ now evaluates our combiner, by replacing inputs to obfuscators up to index i by C_1, for subsequent indices giving input C_0, and replacing the output of the complementing obfuscator $\mathcal{O}_j$ in unit u when evaluating our combiner by O'. Return $\mathcal{D}$'s output bit on input C_0, C_1 and the combiner's output O.

Assume i is such that the i-th obfuscator in order is still different from $\mathcal{O}_k^*$, i.e., $i < K$. Then if O' is the obfuscation of C_1, then $\mathcal{D}_i^k$ runs $\mathcal{D}$ exactly on the distribution of the hybrid variable $H_{i-1}^k(C_0, C_1)$. In particular, for $i = 1$ algorithm $\mathcal{D}_i^k$ runs $\mathcal{D}$ on a sample of our combiner's output for C_1. Analogously, for $i = 9$ and O' stemming from the complementing obfuscator for a sample of the level-1 unit with all C_0 inputs, the input to $\mathcal{D}$ is distributed like a sample of our combiner for C_0 (and thus of $H_9^k(C_0, C_1)$).

Assume that the k-th obfuscator is indeed corrupt. For $i < K$ it follows from the indistinguishability obfuscation of the sound obfuscators that there exist negligible functions $\epsilon_i(\lambda)$ such that for any C_0, C_1, the advantage of $\mathcal{D}_i^k$ in distinguishing the two input cases is at most $\epsilon_i(\lambda)$. For $i \geq K$ this follows as the input circuits to the two sound obfuscators in unit u are already C_0, such that the majority computation of the unit ensures that in both cases the unit circuit computes the function $C_0(\cdot)$. It follows that both input circuits to the complementing obfuscator $\mathcal{O}_j$ compute the same function and we can again conclude from the security of the obfuscator that the advantage must be bounded by some function $\epsilon_i(\lambda)$. Note that here we take advantage of the fact that indistinguishability holds for all circuits and therefore in paticular also for our partly combiner samples.

It therefore also holds for any i that the advantage of $\mathcal{D}$ in distinguishing $H_{i-1}(C_0, C_1)$ and $H_i(C_0, C_1)$ for any C_0, C_1 is at most $\epsilon_i(\lambda)$, too. Hence, the overall advantage of $\mathcal{D}$ is at most $\epsilon(\lambda) := \sum_{i=1}^{9} \epsilon_i(\lambda)$ and thus negligible.[6] □

The claim carries over to the case of differing-inputs obfuscation. Recall that the main difference to indistinguishability obfuscation is that, for the differing-inputs case, the circuits in question are generated by an algorithm $\mathsf{Sampler}$ such that the circuits may compute different functions, but $\mathsf{Sampler}$ ensures that finding differing inputs is infeasible. We can basically apply the same hybrid argument in this case as above. However, for the step $i \geq K$, when using the obfuscation of our level-1 unit, we need to specify sampler $\mathsf{Sampler}'_k$ with oracle access to $\mathcal{O}_1, \ldots, \mathcal{O}_4$ to generate the input circuit for the complementing obfuscator. Algorithm $\mathsf{Sampler}'_k$ first runs $\mathsf{Sampler}$ to get (C_0, C_1, aux), then generates two samples of the level-1 unit (one time using C_0 for the honest obfuscators and C_1 for $\mathcal{O}_k^*$, and the other time using C_0 everywhere), and finally outputs these two samples and $\mathsf{aux}' = (C_0, C_1, \mathsf{aux})$ as auxiliary data. Note that finding an input x where the two level-1 unit samples differ is impossible, as both implement the same function.

We next show that the claim remains true with respect to virtual black-box and grey-box obfuscation. For this we assume that the adversary and the simulator receive some circuit-dependent auxiliary input aux as additional input, as explained in Section 2.

Proposition 1. *The combiner in Figure 1 is a strongly robust* 3*-out-of-*4 *combiner for virtual black-box and grey-box obfuscation with respect to dependent auxiliary input.*

[6] Note that we do not need to know the index k of the obfuscator; unlike the construction it suffices that the proof provides an existential result.

Proof. Functional correctness follows as in the case of indistinguishability obfuscation. We only discuss the VBB property here; the VGB property follows analogously.

Consider an adversary $\mathcal{A}_0$ against VBB obfuscation. This adversary receives an output sample O' of our combiner as input and some auxiliary input $\mathsf{aux}[0] = \mathsf{aux}[0](C)$. Let k be again the index of the malicious obfuscator and this time define $L = L(k) \in \{3, 5\}$ as follows. For $k = 4$ we would have the malicious obfuscator $\mathcal{O}_4^*$ only on first-level units and we only need to look at the $L = 3$ second-level obfuscators. For $k \in \{1, 2, 3\}$, on the other hand, the malicious combiner appears in a second-level unit and we thus consider the $L = 5$ sound obfuscators, consisting of the 3 obfuscators leading to the second-level appearance of $\mathcal{O}_k^*$ and the remaining 2 honest level-two obfuscators.

Assume now that we change the auxiliary input to include the obfuscator results of our combiner for all L sound obfuscators defined above. Denote these intermediate results, ordered according to the obfuscator application, by $O[1..L] = (O_{i_1}, O_{i_2}, O_{i_3}, \ldots, O_{i_L})$, and let $O[1..i]$ denote the first i entries in $O[1..L]$. Let $\mathsf{aux}[0..i]$ denote the sample given by a sample of first i obfuscator outputs, together with the (independent) sample $\mathsf{aux}[0]$ of $\mathcal{A}_0$.

Instead of considering $\mathcal{A}_0(1^\lambda, O', \mathsf{aux}[0])$ we construct an algorithm $\mathcal{A}_1$ which receives 1^λ and $\mathsf{aux}[0..L]$ as input, assembles a combiner output O' from $\mathsf{aux}[1..L]$ by possibly evaluating the (level-2) malicious obfuscator, and runs adversary $\mathcal{A}_0(1^\lambda, O', \mathsf{aux}[0])$. Then, clearly, the output distribution of both algorithms are identical. We can now view $\mathcal{A}_1$ as an algorithm which receives $\mathsf{aux}[0..L-1]$ as auxiliary input, and the obfuscated circuit $\mathsf{aux}[L]$ together with 1^λ as regular input.[7] For this algorithm $\mathcal{A}_1$, by assumption about the security of $\mathcal{O}_{i_L}$ producing O_{i_L}, there exists a simulator $\mathcal{S}_1^C(1^\lambda, \mathsf{aux}[0..L-1])$ with negligibly close output distribution.

Given $\mathcal{S}_1$ we construct an adversary $\mathcal{A}_2$ which receives auxiliary input $\mathsf{aux}[0..L-2]$, and 1^λ and $\mathsf{aux}[L-1]$ as regular input. It runs $\mathcal{S}_1(1^\lambda, \mathsf{aux}[0..L-2])$ and uses $\mathsf{aux}[L-1]$ to answer oracle calls. Note that, by the functional correctness of $\mathcal{O}_{i_{L-1}}$, using $\mathsf{aux}[L-1]$ to simulate the oracle C of $\mathcal{S}_1$ is sound as both circuits compute the same function. We can set this argument forth to eventually obtain a simulator $\mathcal{S}_L^C(1^\lambda, \mathsf{aux}[0])$, producing some output distribution which is negligibly close to the one of our initial adversary $\mathcal{A}_0(1^\lambda, O', \mathsf{aux}[0])$. This shows VBB obfuscation. □

4 Lower Bounds for Combiners

To illustrate how we use the two required security properties, function preservation and indistinguishability, against each other to derive our general result, it is useful to demonstrate our technique for some toy examples. In the examples we use an unspecified notion of indistinguishability of the obfuscators as we merely highlight the issues; the reader may think for sake of concreteness of the notion of indistinguishability obfuscation.

4.1 Simple Attempts That Fail

The first attempt to build a secure combiner consists of a single unit and is given in the left hand part of Figure 4. It uses three obfuscators $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ and runs the input circuit through each of them. Then it combines the three obfuscated circuits by a majority circuit. Note that this means that this part takes the circuits and has some input wires for the input x, and it evaluates each circuit on x and outputs y as the bit-wise majority of the answers. While we cannot break the functional correctness of the combiner with a single corrupt obfuscator $\mathcal{O}_i^*$, we can easily break the indistinguishability property. To this end we take

[7] By construction, if we shift the input of a level-2 obfuscator then this is a sample of a level-1 unit, whereas the other auxiliary inputs are based on the original and functional equivalent circuit C.

control of obfuscator $\mathcal{O}_1^*$ and let it simply output the input circuit in clear. Note that this means that the unit, after having been initialized, reveals the input circuit in clear as well, and this easy to distinguish.

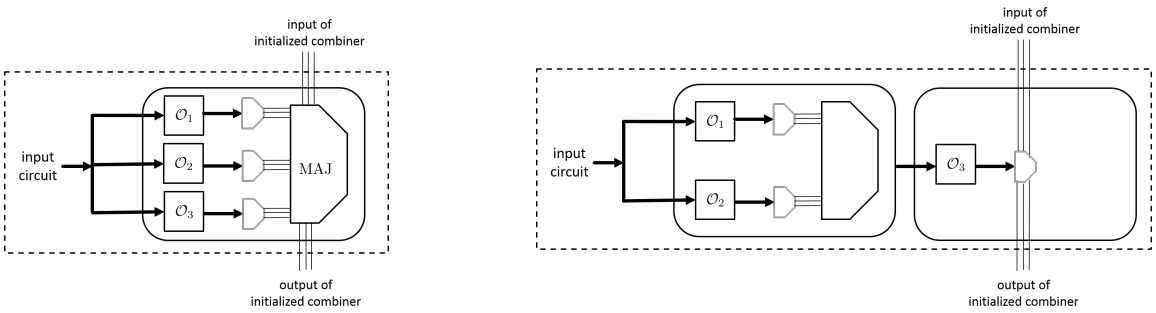

Figure 4: Examples of (insecure) structural combiners

In our second example we have two obfuscators $\mathcal{O}_1, \mathcal{O}_2$, let the output of them be combined arbitrarily, and then input the derived circuit into obfuscator $\mathcal{O}_3$. Note that in this case it is unclear how to break the indistinguishability property by corrupting a single obfuscator only. If it is $\mathcal{O}_3$ then the obfuscators of the first unit already hide the input circuit; if we corrupt one of the obfuscators $\mathcal{O}_1, \mathcal{O}_2$ then the final obfuscation hides the actual circuit.

We can, nonetheless, in the second example break the functional preservation property. Namely, assume that both $\mathcal{O}_1$ and $\mathcal{O}_2$ are secure and that there are two potential input circuits D_0, D_1 computing *different* functions for the same input and output length. If we control $\mathcal{O}_1$, then we let it on any input circuit rather obfuscate D_1. Vice versa, if we control $\mathcal{O}_2$ then we let it always obfuscate D_0, independently of the actual input. It follows that the initialized combiners for D_0 (with our malicious $\mathcal{O}_1^*$ and with genuine $\mathcal{O}_2$) and for D_1 (with genuine $\mathcal{O}_1$ and our malicious $\mathcal{O}_2^*$) have the same distribution. For at least one of the two cases the computed function must then be incorrect, as the initialization samples for both input circuits D_0, D_1 have the same distributions in both cases.

4.2 The General Case of 2-out-of-3 Combiners

The attacks in the simple case show the path for our general impossibility result for 2-out-of-3 structural combiners. If one of the three obfuscators appears in all units on the path from some level-1 unit to the final unit, then it can pass on information about the input circuit C to the final unit. This is done by forwarding some information about the input circuit in the output of the obfuscator. This would clearly violate the indistinguishability property. Hence, on all paths there must be a unit which only uses (at most) the same two obfuscators. But then we can "confuse" the combiner as we did in the second example above. The argument, however, requires some care to deal with the fact that we have many paths. Note also that the confusion strategy fails in our 3-out-of-4 construction because the majority of the three combiners yields the correct function.

For sake of concreteness we use the notion of indistinguishability obfuscation for the obfuscators and the combiners. Recall that this means that for any functionally equivalent circuits C_0, C_1 from class $\mathcal{C}$ the combiners initialization with these two circuits must be computationally indistinguishable. To avoid trivial cases we assume that the class $\mathcal{C}$ contains at least two distinct but functionally *e*quivalent circuits E_0, E_1, and that it also contains two circuits D_0, D_1 computing *d*ifferent functions. We call such classes *non-trivial*.

Since we pass on circuits as inputs we need to fix some encoding. Let $\langle \cdot \rangle$ denote such a function mapping circuits from the class $\mathcal{C}$ to strings. We assume that the encoding is such that given an encoding of a unit (after initialization) one can reconstruct the circuits output by the obfuscators. That is, there

exists an efficient algorithm $\mathsf{reconstruct}$ such that given any initialization $V \leftarrow U(C_1, C_2, C_3, \dots)$ of a unit U, including obfuscated circuits $O_i \leftarrow \mathcal{O}_{i_j}(C_i)$ and possibly pass-through circuits $O_i = C_i$, we have $\mathsf{reconstruct}(\langle V \rangle) = (\langle O_1 \rangle, \langle O_2 \rangle, \langle O_3 \rangle, \dots)$. Furthermore, we assume that the encoding of no unit coincides with the encoding of our equivalent circuits E_0 or E_1 such that it is clear if each $\langle O_i \rangle$ is the result of a unit initialization or rather one of the circuits E_0 or E_1. We call such encodings $\langle \cdot \rangle$ *admissible*. Note that this means that the combiner itself cannot apply any obfuscation techniques beyond the ones provided by the obfuscators placed inside the unit.

We first show that on any full path (from level-1 units to the final unit) each of the three combiners does not appear in a unit (and that a unit in which it does not appear is not pass-through). This holds for any structural combiner, independently of the total number of obfuscators and the number of malicious ones:

Lemma 1. *Let $\mathsf{Comb}^{\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_N}$ be a structural combiner for a non-trivial circuit class $\mathcal{C}$ with admissible encoding $\langle \cdot \rangle$. Then for any full path of units of the combiner and for any $i \in \{1, 2, \dots, N\}$ there must be a unit which is not pass-through and which is not an $\{i, \dots\}$-unit, or else the combiner cannot be an indistinguishable obfuscator.*

Proof. Assume that there exists a full path of units and an $i \in \{1, 2, \dots, N\}$ such that each unit on the path is an $\{i, \dots\}$-unit or that it is pass-through (or both). Then we show how to break indistinguishability obfuscation as follows. Let E_0, E_1 be some functional equivalent circuits in the class with distinct encodings under $\langle \cdot \rangle$. We corrupt obfuscator $\mathcal{O}_i$ and for each input circuit let it, for each call, simply output the input circuit in clear, by duplicating the input description.

Since each unit on the pass includes the i-th obfuscator (or is pass through) a distinguisher can distinguish between a combiner obfuscation of E_0 and E_1 as follows. The distinguisher receives as input the initialization of the final unit U, and runs $\mathsf{reconstruct}(\langle U \rangle)$ to recover all (obfuscated or pass-through) input circuits $O_1, O_2, \dots$. Since the distinguisher knows the layout of the combiner it can recursively apply the reconstruction algorithm to outputs of the i-th combiner resp. to passed circuits; both are initialized units. Following the full path in question, the distinguisher eventually obtains either E_0 or E_1 as the input circuit, and can thus distinguish the two cases easily. □ □

We next show that, given that each full path contains a unit in which, say, obfuscator $\mathcal{O}_3$ does not appear, we can confuse the combiner. This time, the claim only holds for 2-out-of-3 combiners:

Lemma 2. *Let $\mathsf{Comb}^{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3}$ be a structural combiner for a non-trivial circuit class $\mathcal{C}$ with admissible encoding $\langle \cdot \rangle$. Then the combiner cannot be perfectly correct.*

In particular, if u denotes the number of units in the structural combiner and m the maximal number of obfuscator gates in a unit, then with probability at least $2^{-u}(mu)^{-mu}$ (over the random choices of the obfuscators) the combiner's function is different from the one of the input circuit. If u and m are constant, for instance, this means a constant error in functional preservation.

Proof. By Lemma 1 for each path from level-1 units to the final unit there exists a unit which does not contain, say, the obfuscator $\mathcal{O}_3$ and which is neither pass-through. Put differently, such a unit contains (at most) the obfuscators $\mathcal{O}_1, \mathcal{O}_2$, each one possibly multiple times Let $U_1, U_2, \dots$ be the corresponding units which we call *confusion units*. In the example in Figure 5 the confusion units on the three paths are marked by dotted lines.

We consider two cases, one time corrupting obfuscator $\mathcal{O}_1$, the other time corrupting obfuscator $\mathcal{O}_2$. Let us first consider the case that we corrupt obfuscator $\mathcal{O}_1$. Our version $\mathcal{O}_1^*$ of the obfuscator will internally hold, and formally attributed to the non-uniformity, an initialization sample of $\mathsf{Comb}^{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3}(D_1)$ with the genuine obfuscators for input circuit D_1. In particular, for each confusion unit U_i it will include the

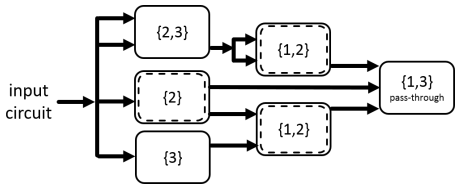

Figure 5: Confusion units in this example are marked by dotted lines.

$j \leq m$ circuit codes of $O_i^j[D_1]$ which the original obfuscator $\mathcal{O}_1$ output in unit U_i in this sample. In order to make our obfuscator state-free we will guess the right insertion positions and injected circuits. That is, for each call (about some input circuit) our malicious obfuscator $\mathcal{O}_1^*$ tosses a coin. If it comes out as head, then the obfuscator proceeds as the genuine obfuscator would. If it is tail, then it picks one of the at most mu circuits $O_i^j[D_1]$ at random, and returns this circuit. An example of a run with good guesses is given in the left part of Figure 5.

For the other case we corrupt $\mathcal{O}_2$ and now include a sample of $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(D_0)$ of the genuine obfuscators, this time for input circuit D_0. Analogously to the other case denote the output of $\mathcal{O}_2$ in the confusion unit U_i by $O_i^j[D_0]$. When called, the malicious obfuscator $\mathcal{O}_2^*$ also generates an honest answer with probability $\frac{1}{2}$, and inserts one of the pre-sampled circuits $O_i^j[D_0]$, the choice made at random, in the other case.

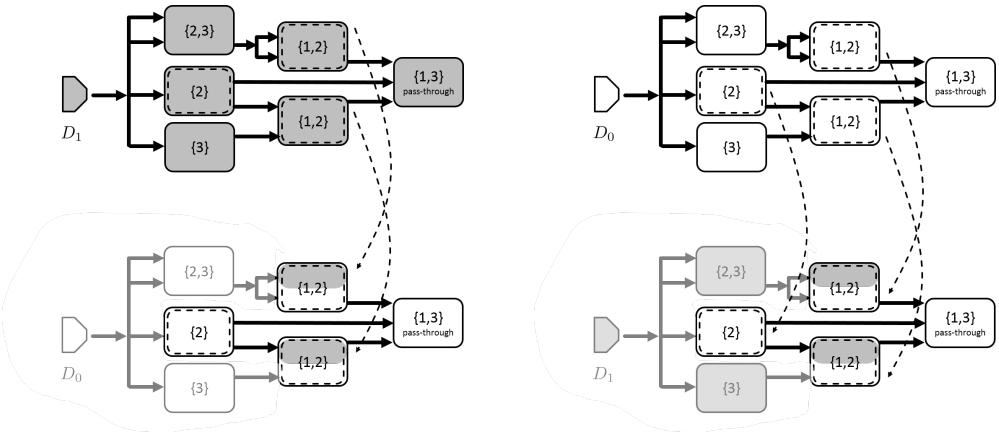

Figure 6: Confusion strategy with malicious obfuscator $\mathcal{O}_1^*$ injecting parts of the upper D_1 initialization sample into the lower D_0 initialization (left), and malicious obfuscator $\mathcal{O}_2^*$ injecting parts of the upper D_0 initialization sample into the lower D_1 initialization (right).

For the analysis we start with the case of a malicious obfuscator $\mathcal{O}_1^*$. Note that, if we let u denote the number of units in the combiner, then with probability 2^{-u} we overwrite the obfuscator's behavior exactly for the confusion units, since we predict the status of each unit (confusion or not) exactly with probability $\frac{1}{2}$. If so, then we also inject the hardwired circuits $O_i^j[D_1]$ "correctly" in confusion unit U_i with probability at least $(mu)^{-mu}$, since we have at most u units with at most m obfuscator gates and need to guess for each unit correctly among the at most mu possibilities among all $O_i^j[D_1]$'s. If this happens, and the combiner receives circuit D_0 as input, then in the confusion units we have consistent samples for $\mathcal{O}_1$ gates (if present), as if the combiners input had been D_1. See the left part of Figure 6 for an example. Simultaneously, in the same unit, we have consistent samples for $\mathcal{O}_2$ gates (if present), as if the overall input had been circuit D_0.

By symmetry, the same is true if we control obfuscator $\mathcal{O}_2^*$ and the combiner's input is D_1. Hence, with probability at least $2^{-u}(mu)^{-mu}$ either case creates the same output distribution. If this happens, then

on each path to the final unit the corresponding confusion unit produces the same distribution upon the single initialization in both cases. It follows that the combiner must implement an incorrect function in one of the cases, showing that functional preservation is not satisfied. It follows that the combiner cannot be perfectly correct. □

Noting that the combiner cannot work even if 2 of the 3 obfuscators both have both properties simultaneously, the previous lemmas imply that there are not even weakly robust 2-out-of-3 combiners for indistinguishability obfuscation. It follows that there cannot exist stronger forms of structural combiners either, such as 1-out-of-2 combiners, strong combiner, or virtual grey-box combiners.

Theorem 2. *For any $o \in \{$ VBB, VGB, indistinguishability, differing-inputs$\}$ there is no structural weakly robust 2-out-of-3 o-obfuscation combiner $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}$ for non-trivial circuit classes $\mathcal{C}$ with admissible encoding $\langle\cdot\rangle$.*

5 The General Case of $(2\gamma[+1])$-out-of-$(3\gamma[+1])$ Combiners

In this section we present a generalization of our 3-out-of-4 combiner to the case of $(2\gamma+1)$-out-of-$(3\gamma+1)$ combiners for any fixed integer γ. In fact, our combiner for $\gamma = 1$ in Section 3 can be seen as a special parallelized version of the general approach here. We then discuss that our lower bound for 2-out-of-3 structural combiners also carries over to the more general case of 2γ-out-of-3γ combiners, showing that our general combiner here is optimal in this regard.

5.1 Robust $(2\gamma+1)$-out-of-$(3\gamma+1)$ Combiners

Consider all sets I of subsets of $\{1, 2, \ldots, 3\gamma+1\}$ of size $2\gamma+1$. For each such set I form the unit which, similar to our 3-out-of-4 case, first in parallel obfuscates the input circuit with each obfuscator $\mathcal{O}_i$ for $i \in I$, and then compute the majority circuit over all these $2\gamma+1$ obfuscated circuits. We write

$$O_I(\cdot) = \mathrm{MAJ}\,\{\mathcal{O}_i(\cdot) \mid i \in I\}$$

for this unit. To obfuscate a circuit C compose each of these units for all the I's sequentially, in arbitrary order. Let us denote this process by

$$(\prod_I O_I)(\cdot) = O_{I_\ell}(\cdots O_{I_3}(O_{I_2}(O_{I_1}(\cdot)))\cdots)$$

for constant $\ell = \binom{3\gamma+1}{2\gamma+1}$. Call this the *sequential-subset combiner* for γ.

Intuitively, the sequential-subset combiner guarantees robustness as there exists a subset I such that this subset only uses the $2\gamma+1$ uncorrupt obfuscators. At the same time each circuit O_I computes the correct function as the majority of the $2\gamma+1$ obfuscators faithfully computes the correct function.

Theorem 3. *For any constant γ the sequential-subset combiner is a strongly robust $(2\gamma+1)$-out-of-$(3\gamma+1)$ combiner for indistinguishability obfuscation, for differing-inputs obfuscation, for virtual black-box obfuscation, and for grey-box obfuscation, the latter ones for dependent auxiliary inputs.*

The proof is similar to our 3-out-of-4 combiner. Functionally correctness follows from the fact that the majority computation in each O_{I_i} ensures that the at most γ corrupt obfuscators cannot bias the outcome. Obfuscation follows as before because there must exist one set I which exclusively contains non-malicious obfuscators.

5.2 Impossibility for 2γ-out-of-3γ Combiners

In this section we discuss that our lower bound for 2-out-of-3 structural combiners carries over to the more general case of 2γ-out-of-3γ combiners.

Theorem 4. *There is no structural weakly robust 2γ-out-of-3γ combiner $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\ldots,\mathcal{O}_{3\gamma}}$ for non-trivial circuit classes $\mathcal{C}$ with admissible encoding $\langle\cdot\rangle$.*

Proof. Recall the proof for the 2-out-of-3 case. There, in the first step we have shown that on each path from a level-1 unit to the output unit there must be a unit in which obfuscator $\mathcal{O}_3$ does not appear and which is not pass-through. We called these units confusion units.

The same argument now applies here as well for the γ obfuscators with indices $2\gamma+1,\ldots,3\gamma$. Else, if there was a path in which one of these obfuscators appears in each unit (or if the unit is pass-through), then we could easily corrupt these obfuscators and forward information about the input circuit through the admissible encoding $\langle\cdot\rangle$. Hence, in the case here there must be a confusion unit on each path, which only uses circuits with indices $1,2,\ldots,2\gamma$ and which are not pass-through.

In the second step of the proof for the 2-out-of-3 case we then show that in the confusion units with obfuscators $\mathcal{O}_1$ and $\mathcal{O}_2$ we can confuse the combiner. One time we corrupt $\mathcal{O}_1$ and let it insert samples of circuit D_1, and the other time we corrupt $\mathcal{O}_2$ and insert samples for D_0, where D_0, D_1 compute different functions. Then the combiner's view when run on input D_0 in the first case, and on D_1 in the second case, has the same distribution and the combiner cannot provide functional correctness.

We apply the same argument here, one time corrupting the first γ obfuscators with indices $1,\ldots,\gamma$ and inserting a sample for D_1, and the other time corrupting obfuscators with indices between $\gamma+1,\ldots,2\gamma$ and using a sample for D_0. Then the combiner's views in both cases (for input circuit D_0 in the first case, and for D_1 in the second case) are identical again such that it cannot provide a correct combiner.

As in the 2-out-of-3 case the malicious obfuscators above insert the confusion samples at random positions, such that it only achieves confusion with the same bound as in the previous case. Note also that we took advantage of the fact that corrupt combiners are coordinated centrally by the adversary. □

6 Detecting Combiners

The combiners in the previous section were correcting in the sense that they guaranteed functionality correctness if a quorum if obfuscator candidates is secure. Here we consider combiners which should create circuits which either output the correct value, but may give some error output $\perp$. We call them detecting combiners.

For detecting combiners we require a weaker correctness property, namely that for any circuit $C \in \mathcal{C}$, for any $O \leftarrow \mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\cdots}(C)$ we have that $O(x) \in \{C(x), \perp\}$ for all $x \in \{0,1\}^*$ in the domain of C. This means that the combiner may sometimes fail to compute the correct function value but then it signals this by outputting a special symbol $\perp$. To prevent trivial solutions like the combiner which outputs the circuit that always returns $\perp$ we assume that $C \equiv O$ if all obfuscators are secure. Note that our assumption about the obfuscators $\mathcal{O}_1, \mathcal{O}_2, \ldots$ being able to deal with (intermediate) combiner outputs in $\mathcal{C}^{\mathsf{Comb}}$ implies that the obfuscators may now also receive circuits which occasionally output $\perp$.

6.1 Robust $(\gamma+1)$-out-of-$(2\gamma+1)$ Detecting Combiners

To build our $(\gamma+1)$-out-of-$(2\gamma+1)$ combiner we follow the approach of our sequential-subset combiner. We can also straightforwardly give the optimized version for the case of a 1-out-of-3 combiner, akin to our 1-out-of-4 combiner, but omit this step here. To build the sequential-subset combiner consider here all sets I of subsets of $\{1,2,\ldots,2\gamma+1\}$ of size $\gamma+1$. For each such set I we first obfuscate the input circuit

with each obfuscator $\mathcal{O}_i$ for $i \in I$. But now instead of completing the computation by adding a majority sub circuit, we now use the detecting version which (a) either outputs the string on which all circuits agree upon as output (even if it is $\perp$), or (b) returns $\perp$ is there is no such unanimous decision. Let

$$O_I(\cdot) = \text{UNAN}\,\{\mathcal{O}_i(\cdot) \mid i \in I\}$$

denote this unit with the unanimity circuit at the end. For obfuscation of C now compute the sequential-subset combiner

$$(\prod_I O_I)(\cdot) = O_{I_\ell}(\cdots O_{I_3}(O_{I_2}(O_{I_1}(\cdot)))\cdots)$$

as before for constant $\ell = \binom{2\gamma+1}{\gamma+1}$.

Theorem 5. *For any constant γ the sequential-subset combiner is a strongly robust $(\gamma+1)$-out-of-$(2\gamma+1)$ detecting combiner for indistinguishability obfuscation, for differing-inputs obfuscation, for virtual black-box obfuscation, and for grey-box obfuscation, the latter ones for dependent auxiliary inputs.*

The proof is similar to the case of correcting combiners, except that we only guarantee the weaker functional correctness. This property is given since in each unit for index set I there is at least one honest obfuscator among the $\gamma+1$ ones, the unanimity circuit either outputs the function value computed by the honest obfuscator (if all other circuits agree), which may either be the correct function value for some x or $\perp$, or it returns the error message $\perp$. It follows that the overall output of the combiner circuit can only comply with the circuit's output, or returns $\perp$. The obfuscation properties follow as before noting that the obfuscators are able to handle input circuits with output $\perp$, and that there must exist an index set I which only contains good obfuscators.

6.2 Impossibility of γ-out-of-2γ Detecting Combiners

The idea for the lower bound for correcting combiners carries over to detecting combiners, with

Theorem 6. *There is no structural weakly robust γ-out-of-2γ combiner $\mathsf{Comb}^{\mathcal{O}_1,\mathcal{O}_2,\ldots,\mathcal{O}_{2\gamma}}$ for non-trivial circuit classes $\mathcal{C}$ with admissible encoding $\langle\cdot\rangle$.*

Proof. As in the case of 2-out-of-3 combiners and 2γ-out-of-3γ combiners, here, there must be also (non-pass-through) confusion units in each path from input units to the final unit, where none of the obfuscators with indices $\gamma+1,\ldots,2\gamma$ appears. Assume now that we corrupt the obfuscators with indices $1,\ldots,\gamma$ and let these obfuscators insert intermediate samples of a circuit D_0 of the combiner's obfuscation in the confusion units, independently of the input. If the insertions happen at the right position with significant probability, then the combiner must output an obfuscated circuit *as if the combiner has been run on D_0 for honest obfuscators.* In particular, the combiner's circuit must then compute the function D_0 on every input. This holds even if the original input circuit was D_1, computing a different function than D_0, i.e., $D_0(z) \neq D_1(z)$ for some string z. But then the combiner's circuit produces a false output $D_0(z) \neq \perp$ for input z and cannot be detecting. □

7 Implementation and Evaluation

Our formal results have been stated in terms of the common notion of circuit obfuscation. In practice, however, programs are usually considered to be better modeled for Turing machines. We stress that our results, especially for the majority-based combiner, hold for such Turing machine programs as well. Namely, our 3-out-of-4 combiner would then output the program implementing the nested majority implementations.

Concerning provably secure instantiations for Turing machine obfuscation, we note that if the running time and the input length of the Turing machine are bounded then one can in principle transform such machines into corresponding circuits, albeit at the cost of increasing the complexity significantly. A more efficient solution is to use obfuscation techniques for Turing machines directly. Given the current state of constructions this is possible if the input length can be bounded [KLW15] and, for other constructions, if the space is also bounded beforehand [CHJV15, BGL$^+$15].

To evaluate the suggested combiners for typical obfuscation programs in practice we implemented the PyObf python package [Bin16] that can be used to wrap existing obfuscators and to implement new combiners. Even though the conceptual construction of a combiner is not related to the concrete implementation of the underlying obfuscators, implementations for different programming languages might differ, since some constructions introduce new run-time parts (e.g., as the MAJ circuit in our case) to the program. We chose to use JavaScript as the implementation programming language because of the relatively high number of available obfuscators.

7.1 Performance Evaluation

For performance evaluation we used Yahoo!'s YUICompressor v2.4.8[8] as $\mathcal{O}_1$, a slightly randomized version of it as $\mathcal{O}_2$, Google's Closure Compiler v20151015[9] as $\mathcal{O}_3$, and jsPacker.pl v1.00b[10] as $\mathcal{O}_4$. Note that there is no essential difference between $\mathcal{O}_1$ and $\mathcal{O}_2$, especially in terms of obfuscation overhead, since the latter only uses different and randomized symbol selection routine. Security of combiners usually relies on somewhat independent components but since we are mainly interested in performance evaluation here we opted for using the related choice. The evaluated combiners are:

$$
\begin{aligned}
C_1(.) &= \mathcal{O}_4(\mathcal{O}_3(\mathcal{O}_2(\mathcal{O}_1(.)))) \\
C_2(.) &= \mathcal{O}_2(\mathcal{O}_1(\mathcal{O}_4(\mathcal{O}_3(.)))) \\
C_3(.) &= \mathrm{MAJ}(\mathcal{O}_1(O_{2,3,4}), \mathcal{O}_2(O_{1,3,4}), \mathcal{O}_3(O_{1,2,4})) \\
C_4(.) &= \mathrm{MAJ}(\mathcal{O}_3(O_{4,1,2}), \mathcal{O}_4(O_{3,1,2}), \mathcal{O}_1(O_{3,4,2}))
\end{aligned}
$$

The evaluated programs (with varying input size, ranging from a few thousand bytes to a roughly million bytes) are Cookies.js v1.2.2[11] (6,637 bytes), Highlight.js v9.0.0[12] (22,604 bytes), jCarousel v0.3.4[13] (46,007 bytes), Backbone.js v1.2.3[14] (71,415 bytes), Chart.js v1.0.2[15] (109,612 bytes), Epoch v0.8.4[16] (115,940 bytes), Swig v1.4.2[17] (143,975 bytes), PhysicsJS v0.7.0[18] (171,847 bytes), jQuery v1.6.4[19] (238,166 bytes), Raphaël v2.1.4[20] (304,254 bytes), Dojo v1.10.4[21] (629,481 bytes), Video.js v5.4.4[22] (675,527 bytes) and AngularJS v1.4.5[23] (1,052,336 bytes).

[8] https://github.com/yui/yuicompressor
[9] https://github.com/google/closure-compiler
[10] http://dean.edwards.name/download/
[11] https://github.com/ScottHamper/Cookies
[12] https://highlightjs.org
[13] http://sorgalla.com/jcarousel/
[14] http://backbonejs.org/
[15] http://www.chartjs.org/
[16] http://epochjs.github.io/epoch/
[17] http://paularmstrong.github.io/swig/
[18] http://wellcaffeinated.net/PhysicsJS/
[19] https://jquery.com/
[20] https://github.com/DmitryBaranovskiy/raphael
[21] https://dojotoolkit.org/
[22] http://videojs.com/
[23] https://angularjs.org/

Note that the above circuit model describes a program as a function with input and output, in contrast to the common software design of JavaScript libraries that heavily depends on the JavaScript context (e.g., the window object). But this difference is irrelevant to performance evaluation.

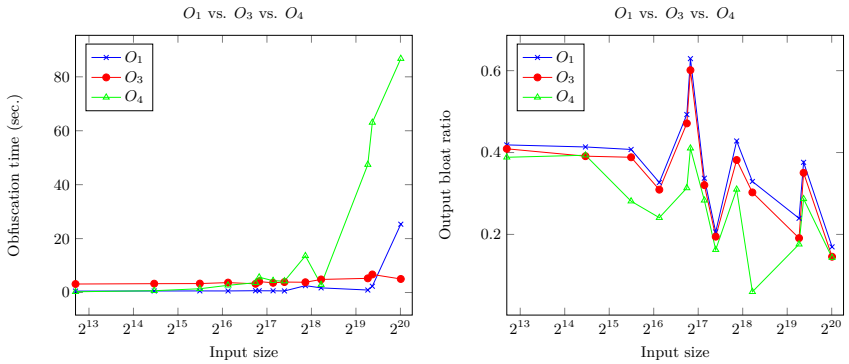

Figure 7: Overhead of individual obfuscators (time and output bloat ratio) for the various programs (in relation to their input sizes for the obfuscator). Note that we do not display obfuscator $\mathcal{O}_2$ here as its performance is essentially identical to the one of $\mathcal{O}_1$.

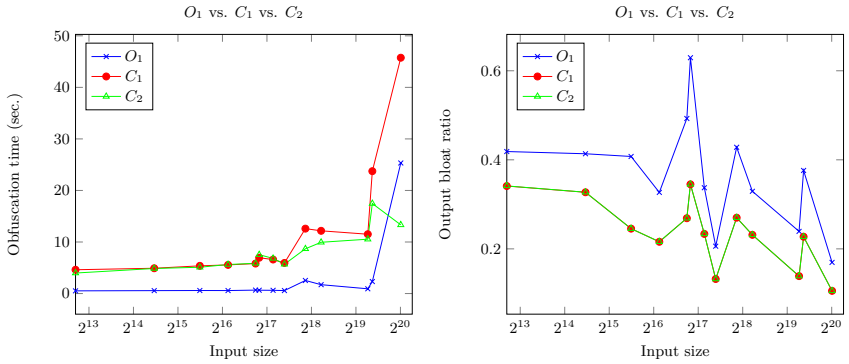

Figure 8: Overhead of cascaded combiner (time and output bloat ration).

Figure 7 gives the effectiveness of the obfuscators in terms of obfuscation time and of output-bloat ratio. Here we show the figures in relation of the sizes of the various programs (from 6,637 bytes to 1,052,336 bytes) given as input to the obfuscators. Note that the results may depend heavily on the specific input programs such that we cannot expect perfectly monotonic behavior in the graphs. Also, as mentioned before, many practical obfuscators come with techniques for code size reduction such that the output bloat ratio can be—and often is—smaller than 1. Next, we compare these figures to the results of the suggested combiners, first to the cascade combiners in Figure 8 and then to the 3-out-of-4 combiners in Figure 9.

In summary, the proposed obfuscation combiners do not add significant run time overhead compared to a single obfuscator. The factor is roughly proportional to the number of invoked instances, with some gains presumably due to the intermediate code optimization. Due to the advanced compression techniques the code size of our cascaded combiners is in the same order as the individual obfuscators. For the 3-out-of-4 combiner we of course get an increased output size because of the tripling for each majority step, potentially also hampering some code reductions.

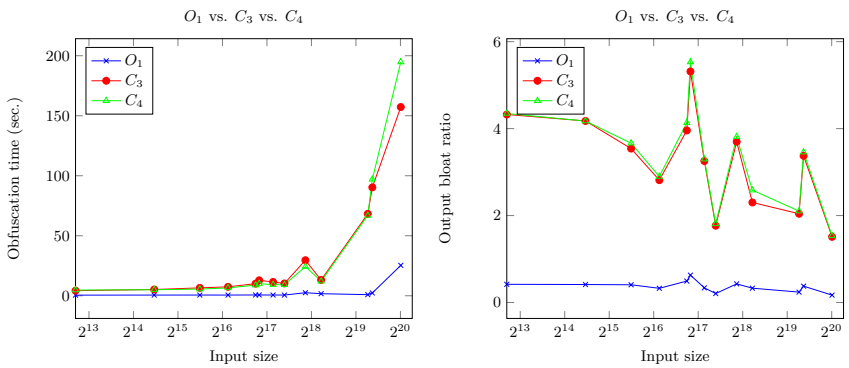

Figure 9: Overhead of 3-out-of-4 combiner (time and output bloat ration).

7.2 Security Evaluation

Due to the unclear situation about security properties of practical obfuscators we have proven robustness of our combiners with respect to the common theoretical notions of obfuscation in the literature. There are approaches to define metrics for practical obfuscators, though. A first approach is by Collberg at al. [CTL97] who define notions for *potency* (the incomprehensibility of the transformed program for humans), *resilience* (the hardness of undoing the transformation through the joint effort of engineers and deobfuscation techniques), and *cost* (the overhead caused by the obfuscator). The measure of quality of an obfuscator is then given by a vector of these three metrics.

While the notion of cost in [CTL97] even distinguishes the full range between exponential and constant overhead in execution resources, the metrics for potency and resilience in [CTL97] are less rigorous. They are accompanied by suitable software complexity measures such as program length or cyclomatic complexity [McC76]. Anckaert et al. [AMS+07] later used software complexity measures, too, for establishing a benchmarking system for obfuscators for binary executables.

While it is beyond the scope of our work here, it may be interesting to benchmark our obfuscation combiners according to the metrics in [AMS+07]. Note that their metrics focus on resilience and somewhat neglect the overhead. Since our combiners in principle increase the software complexity at the cost of incurring additional steps, one should expect that combinations of benchmarked obfuscators yield better values in this regard.

It should be noted, however, that the theoretical notions also supply some form of guarantees for the usual security requirements for practical obfuscators. Take for example the obfuscation of variable names (sometimes called lexicographic obfuscation) or of loop structures (also called control flow obfuscation), which is typically aspired by such obfuscators. One can either view this as some property $\pi(C)$ of the program C, and then VBB and VGB obfuscation even guarantee that *any* predicate $\pi(C)$ is infeasible to compute —this is an equivalent formulation to the given one here— or, taking the viewpoint of indistinguishability obfuscation, one may argue that different starting variable names or equivalent structures cannot be distinguished after obfuscation anymore, also supplying some form of hiding.

7.3 Experiments with Malicious Obfuscators

In order to evaluate the proposed combiners in terms of practical leakage, some malicious obfuscators needed to be tested. The only demand from a malicious obfuscator is that its output will leak some information about the original program, we call this information the *leak object*. In our evaluations the leak object was chosen to be the encrypted original program along with the wrapped encryption key. There are several leakage methods that can be considered:

In-code leakage: The in-code leakage method simply embeds the leak object in the obfuscator's output in some way the adversary can detect later. This leakage method is for example prevented by the cascade combiner.

Run-time leakage: This method adds a leaking code stub to the obfuscator's output. The stub outputs the leak object if some "magic" input values are in place. This leakage method is not prevented by the cascade combiner but is prevented for instance by the 3-out-of-4 combiners.

External leakage: In contrast to the previous methods, with this method the leak object is leaked using an external output mechanism. As before, this leakage method is prevented by the 3-out-of-4 combiners with the restriction of disabling of external interfaces.

Side-channel leakage This leakage method can use some side-channel (e.g. computation time, power) to leak the leak object. At first, it seems that timing side-channel can be prevented by fixing a constant computation time but fixing a limit to the computation time might introduce a new, distinguishable, error flow. This error flow can be abused by the adversary to leak the information by choosing whether to delay the computation above the limit or not. Also, it seems that the power side-channel cannot be prevented easily. Indeed, such side-channel attacks are due to imperfections of running over practical computers; in particular, such attacks are not relevant for the circuits model.

We have implemented the in-code, runtime and external leakage methods for YUICompressor as part of PyObf and confirmed the expected results. The cascade combiner only mitigates the in-code leakage method whereas the 3-out-of-4 combiner is able to protect the source code from run-time leakage as well. These (simple) implementations demonstrate the challenges of combining obfuscators. (A reference will be added in non-anonymized version.)

8 Conclusion

Our positive results about combiners, and also our lower bounds, indicate how to proceed both in theory and practice. If you only have two available candidates then the best solution appears to be the sequential composition $\mathcal{O}_2(\mathcal{O}_1(\cdot))$, *if one can somehow guarantee that the inner obfuscator provides functional correctness*. For three candidates (out of which at least two are sound) then our 2-out-of-3 *detecting* combiner should be the primary choice. To ensure correct output, our 3-out-of-4 combiner provides a secure solution.

Acknowledgments

We are grateful to Christian Collberg for his feedback and encouragement. Marc Fischlin is supported by the Heisenberg grant Fi 940/3-2 and the SPP 1736 grant Fi 940/5-1 of the German Research Foundation (DFG). Amir Herzberg is support by the Israeli Ministry of Science and Technology.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/2013/689`. (Cited on page 6.)

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in*

Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. (Cited on page 2.)

[AJN+] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal obfuscation and witness encryption: Boosting correctness and combining security. `http://eprint.iacr.org/2016/281`. (Cited on page 5.)

[AJS] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation with constant size overhead. IACR Cryptology ePrint Archive, Report 2015/1023. `http://eprint.iacr.org/2015/1023`. (Cited on page 3.)

[AMS+07] Bertrand Anckaert, Matias Madou, Bjorn De Sutter, Bruno De Bus, Koen De Bosschere, and Bart Preneel. Program obfuscation: a quantitative approach. In *Proceedings of the 3th ACM Workshop on Quality of Protection, QoP 2007, Alexandria, VA, USA, October 29, 2007*, pages 15–20. ACM, 2007. (Cited on pages 2 and 21.)

[BB06] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 570–583, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. (Cited on pages 2 and 8.)

[BC10] Nir Bitansky and Ran Canetti. On strong simulation and composable point obfuscation. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 520–537, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. (Cited on page 6.)

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012. (Cited on pages 1, 5, 6, and 7.)

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. (Cited on page 3.)

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press. (Cited on page 19.)

[Bin16] Hod Bin Noon. Pyobf. `https://github.com/hodbn/pyobf`, 2016. (Cited on page 19.)

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany. (Cited on page 3.)

[BV] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: from approximate to exact. `http://eprint.iacr.org/2015/704`. (Cited on page 6.)

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190. IEEE Computer Society, 2015. (Cited on page 3.)

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany. (Cited on page 1.)

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 247–266, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. (Cited on page 2.)

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 429–437, Portland, OR, USA, June 14–17, 2015. ACM Press. (Cited on page 19.)

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. Lecture Notes in Computer Science, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 2.)

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2014/032, 2014. `http://eprint.iacr.org/2014/032`. (Cited on page 2.)

[CT02] Colberg and Thomborson. Watermarking, tamper-proofing, and obfuscation–tools for software protection. *IEEETSE: IEEE Transactions on Software Engineering*, 28, 2002. (Cited on page 1.)

[CTL97] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report #148, Department of Computer Science, The University of Auckland, New Zealand, 1997. (Cited on pages 2 and 21.)

[DK05] Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 188–209, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany. (Cited on page 2.)

[FL07] Marc Fischlin and Anja Lehmann. Security-amplifying combiners for collision-resistant hash functions. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 224–243, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany. (Cited on page 2.)

[FL08] Marc Fischlin and Anja Lehmann. Multi-property preserving combiners for hash functions. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 375–392, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany. (Cited on page 7.)

[FLP14] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust multi-property combiners for hash functions. *Journal of Cryptology*, 27(3):397–428, July 2014. (Cited on pages 2 and 7.)

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press. (Cited on pages 2 and 3.)

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on page 6.)

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 2014. `http://eprint.iacr.org/2014/929`. (Cited on page 2.)

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *46th Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PA, USA, October 23–25, 2005. IEEE Computer Society Press. (Cited on pages 3 and 6.)

[GLSW15] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption, 2015. (Cited on pages 2 and 3.)

[Her02] Amir Herzberg. Folklore, practice and theory of robust combiners. Cryptology ePrint Archive, Report 2002/135, 2002. `http://eprint.iacr.org/2002/135`. (Cited on page 2.)

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 172–190, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany. (Cited on page 2.)

[Her09] Amir Herzberg. Folklore, practice and theory of robust combiners. *Journal of Computer Security*, 17(2):159–189, 2009. (Cited on page 2.)

[HJK+14] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. Cryptology ePrint Archive, Report 2014/507, 2014. `http://eprint.iacr.org/2014/507`. (Cited on page 3.)

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 96–113, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. (Cited on page 2.)

[HS10] Amir Herzberg and Haya Shulman. Robust combiners for software hardening. In *Trust and Trustworthy Computing, Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings*, volume 6101 of *Lecture Notes in Computer Science*, pages 282–289. Springer, 2010. (Cited on page 2.)

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press. (Cited on page 19.)

[McC76] Thomas J. McCabe. A complexity measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976. (Cited on page 21.)

[Mit13] Arno Mittelbach. Cryptophia's short combiner for collision-resistant hash functions. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 136–153, Banff, AB, Canada, June 25–28, 2013. Springer, Heidelberg, Germany. (Cited on page 2.)

[MP06] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 555–569, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. (Cited on page 2.)

[MP14] Bart Mennink and Bart Preneel. Breaking and fixing cryptophia's short combiner. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14: 13th International Conference on Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 50–63, Heraklion, Crete, Greece, October 22–24, 2014. Springer, Heidelberg, Germany. (Cited on page 2.)

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 404–418, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany. (Cited on page 2.)

[Pie07] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 23–33, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany. (Cited on pages 2 and 8.)

[Pie08] Krzysztof Pietrzak. Compression from collisions, or why CRHF combiners have a long output. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 413–432, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany. (Cited on pages 2 and 8.)

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on pages 2 and 3.)