Boolean Circuit Camouflage: Cryptographic Models, Limitations, Provable Results and a Random Oracle Realization

Giovanni Di Crescenzo*
Vencore Labs
Basking Ridge, NJ, 07920, USA
gdicrescenzo@vencorelabs.com

Jeyavijayan Rajendran†
Texas A&M University
College Station, TX, 777840, USA
jv.rajendran@tamu.edu

Ramesh Karri
NYU Tandon School of Engineering
Brooklyn, NY, 11201, USA
rkarri@nyu.edu

Nasir Memon
NYU Tandon School of Engineering
Brooklyn, NY, 11201, USA
memon@nyu.edu

ABSTRACT

Recent hardware advances, called *gate camouflaging*, have opened the possibility of protecting integrated circuits against reverse-engineering attacks. In this paper, we investigate the possibility of provably boosting the capability of physical camouflaging of a single Boolean gate into physical camouflaging of a larger Boolean circuit. We first propose rigorous definitions, borrowing approaches from modern cryptography and program obfuscation areas, for circuit camouflage. Informally speaking, gate camouflaging is defined as a transformation of a physical gate that appears to mask the gate to an attacker evaluating the circuit containing this gate. Under this assumption, we formally prove two results: a limitation and a construction. Our limitation result says that there are circuits for which, no matter how many gates we camouflaged, an adversary capable of evaluating the circuit will correctly guess all the camouflaged gates. Our construction result says that if pseudo-random functions exist (a common assumptions in cryptography), a small number of camouflaged gates suffices to: (a) leak no additional information about the camouflaged gates to an adversary evaluating the pseudo-random function circuit; and (b) turn these functions into random oracles. These latter results are the *first* results on circuit camouflaging *provable in a cryptographic model* (previously, construction were given under no formal model, and were eventually reverse-engineered, or were argued secure under specific classes of attacks). Our results imply a *concrete and provable realization of random oracles*, which, even if under a hardware-based assumption, is applicable in many scenarios, including public-key infrastructures. Finding special conditions under which provable realizations of random oracles has been an open problem for many years, since a software-only provable implementation of random oracles was proved to be (almost certainly) impossible.

CCS CONCEPTS

• **Security and privacy → Hardware reverse engineering**; *Public key encryption*; *Cryptanalysis and other attacks*; *Mathematical foundations of cryptography*; *Privacy-preserving protocols*; *Hardware-based security protocols*; *Malicious design modifications*;

KEYWORDS

Hardware security, Camouflaging, Intellectual Property, Piracy, Cryptography, Program Obfuscation

1 INTRODUCTION

Reverse engineering of an integrated circuit (IC) entails identifying its structure, design and functionality. Reverse engineering can identify the device technology used in an IC [11], extract the circuit of the design [29], and infer its functionality [17]. Several techniques and tools have been developed to enable reverse engineering of an IC (see, e.g., [27] for a tutorial and [10, 15] for software products). An attacker can reverse-engineer an IC to steal and/or pirate a circuit design. One can use the readily available tools and techniques for reverse engineering. IC reverse engineering was listed as one of the serious threats to semiconductor industry [26].

To prevent reverse engineering, SypherMedia [28] provides IC camouflaging services for this purpose. IC camouflaging involves designing a Boolean logic gate in hardware that can implement a plurality of functions. The actual function implemented by the gate is known only to the designer. To a reverse engineer, the camouflaging gate can look as any of the functions that can be implemented by it [25, 28].

Intrigued by recent hardware advances in camouflaging as a possible avenue for protection of integrated circuits against reverse-engineering attacks [28], we start a comprehensive and rigorous approach to the study of circuit camouflaging. We borrow from research approaches in the modern cryptography and program obfuscation areas, by showing formal definitions, the first provable limitation result, the first provable positive result, and an interpretation of the positive result as a construction of a circuit that behaves like a random oracle.

*Part of work done while visiting NYU Tandon School of Engineering.
†Part of work done while at NYU Tandon School of Engineering.

ASHES'17, November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5397-7/17/11…$15.00
https://doi.org/10.1145/3139324.3139331

Our problem: In general terms, the *circuit camouflaging* problem can be described as follows. Let c be a Boolean circuit realizing a function f. The camouflaging problem for c asks to transform c into a related Boolean circuit that computes an equivalent function, critically building on *gate camouflaging* techniques, based on a class of hardware techniques [28], so to limit an adversary's efforts to reverse-engineer c or obtain some information about c or f, even when given enough computational resources and while being capable to read and evaluate the camouflaged version of c. As stated, the problem bears some resemblance to the well-studied problem of program obfuscation, recently receiving much attention in the security and cryptography literature. One main difference between the two areas is that camouflaging mainly relies on hardware-based circuit design techniques while obfuscation mainly relies on software-based cryptographic computation techniques.

Our contribution: In this paper we first propose rigorous definitions for the stated circuit camouflage problem. Briefly speaking, circuit camouflaging is formally defined so to prevent an efficient adversary which has both partial information about the circuit, and oracle access to it, to obtain some information about the functionality of the camouflaged gates in the circuit. Although camouflaging seems superficially similar to program obfuscation [2], our camouflage security definitions are significantly different than obfuscation definitions in the literature (although we do propose similar exact functionality and bounded slowdown requirements).

Our first result is a limitation on what circuits can be camouflaged in this model. We rigorously prove that monotone formulae cannot be camouflaged in a strong sense, in that any adversary having oracle access to a monotone formula with even all gates being camouflaged, can quickly find all formula gates, with probability 1. This result can be extended to all circuits that are learnable by a learner who knows the circuit structure, has oracle access to the camouflaged circuit, and targets exactly learning its camouflaged gates. Previously, no limitations were known on camouflaging. Limitations based on learning, although of different type, were known in the program obfuscation area (see, e.g., [30]), and somewhat inspired our result.

Our second (and main) result is a positive result. Assuming physical means of camouflaging single Boolean gates (as surfaced in recent advances on hardware techniques like dummy contacting) and the existence of families of pseudo-random functions (a standard assumption in modern cryptography), we show how to provide provable camouflaging of the keys in pseudo-random functions, thus resulting in a concrete hardware-based implementation of a random oracle (or, more accurately speaking, a circuit whose input/output behavior is computationally indistinguishable from that of a random oracle). This result is the *first* result *provable in a cryptographic model* on circuit camouflaging (previously, constructions were given under no formal model and were eventually reverse-engineered, or only targeted specific attacks). We show that even an adversary with read and evaluation access to the camouflaged circuits can only efficiently guess the value of the camouflaged gates better than by random guessing by no more than negligible probability. Also, this result implies a *concrete and provable realization of random oracles*, which, even if under a hardware-based assumption, is applicable to scenarios where a server can be trusted to ship hardware circuits to clients (e.g., public-key infrastructures among companies, universities, organizations). Finding special conditions under which provable realizations of random oracles has been an open problem for many years. On one hand, random oracles have been used to enable several cryptographic constructions; on the other hand a software-only provable implementation of random oracles was proved to be (almost certainly) impossible (see [9], as well as [20] for a survey of the many results in this area). This should be contrasted with the positive results in cryptographic program obfuscation, where only for a handful of simpler specific functions, obfuscators have been designed (see, e.g., [8, 23, 30]) and demonstrated (see, e.g., [1, 13]) using software-only techniques, based on intractability assumptions that are standard in cryptography.

Related work. Rajendran *et al.* showed that when gates in a design are camouflaged randomly, an attacker can infer their correct functionality by propagating the camouflaged gates' output to the primary outputs of the circuit [25]. To thwart this attack, clique-based selection camouflages a set of gates such that the outputs of these gates cannot be sensitized to a primary output without accounting for the other gates in the set; this set of camouflaged gates is referred to as a clique [25]. The DeCamo attack demonstrated that the functionality of camouflaged gates selected using clique-based selection can be determined by generating a set of input patterns [22]. Despite these attacks, camouflaging with improved security guarantees against the DeCamo attack has been demonstrated [21, 31]. However, [21] has been broken by removal attacks [32]. Note that all these techniques try to protect general Boolean circuits, under specific attack models. In this work, we consider camouflaging of specific sets of Boolean functions, in general attack models, as in the cryptography literature. For a more detailed survey of gate camouflaging approaches, including and beyond dummy contacts, as well as alternative methods to strengthen circuits against reverse-engineering, we refer the reader to Section 7 of [31]. Recent alternative approaches also include threshold-based camouflaging (see, e.g., [12]).

Organization of the paper. In Section 2 we detail definitions and models of interest, including our adversary model, and a formal definition for both gate camouflaging and circuit camouflaging. In Section 3 we present our limitation result on camouflaging of circuits derived from monotone formulae. In Section 4 we present our positive result on how to provably camouflage pseudo-random functions starting from any gate camouflaging technique. Finally, in Sections 5 we present our conclusions.

2 MODELS AND DEFINITIONS

In this section we detail definitions and models of interest, including our adversary model, and a formal definition for gate camouflaging and circuit camouflaging.

2.1 Basic Definitions

By $|x|$ we denote the length of x if x is a string, the size of x if x is a set, the number of gates of x if x is a circuit. A function ϵ over the set of natural numbers is *negligible* if for any constant c, there exists an integer n_0 such that for all $n \geq n_0$, it holds that $\epsilon(n) \leq 1/n^c$. Given a (discrete) probability distribution D, the notation $x \leftarrow D$ is used to denote the random process of independently drawing a sample

x according to D. Similarly, the notation $y \leftarrow A(x)$, where A is an algorithm, denotes the random process of obtaining y when running algorithm A on input x, where the probability space is given by the random coins (if any) of algorithm A. By $\text{Prob}[R_1; \ldots; R_n : E]$ we denote the probability of event E, after the sequential execution of random processes $R_1, \ldots, R_n$.

Logical representations of gates and circuits. For modeling purposes, we consider function computation as implemented on hardware via generic, yet standard, notions of *physical gates* and *physical circuits*. As algorithmic computation is defined over digital strings, we now define logical abstractions of physical gates and circuits into digital strings.

We consider a *logical abstraction* as a map associating a physical gate to a binary string, called *logical gate*. A conventional logical abstraction entails mapping a physical gate to its truth table, where the truth table of a gate is defined as the list of values output by the gate for all possible input values. We consider that the *logical representation* of a physical gate is the logical gate obtained after applying a logical abstraction to the physical gate. However, we note that if a physical gate is available to an adversary, the adversary can choose to generate its own logical abstraction, which may be different than the conventional one.

To every physical circuit c, we associate a directed graph $G(c) = (V(c), E(c))$, called the *c-graph*, as follows:

1. for each input in c, an associated vertex is entered in $V_i(c)$;
2. for each gate in c, an associated vertex is entered in $V_g(c)$;
3. for each output in c, a vertex is associated and entered in $V_o(c)$;
4. $V(c) = V_i(c) \cup V_g(c) \cup V_o(c)$;
5. for each wire between an input and a gate, between two gates, or between a gate and a circuit output, an associated directed edge between the two associated vertices is entered into $E(c)$.

We can view a c-graph as a representation of circuit c that ignores the types of gates present in c. For every physical circuit c and its c-graph $G(c) = (V(c), E(c))$, we associate a *vertex labeling* $L(c)$ that maps every $v \in V(c)$ to a pair of attributes $(vtype, gtype)$, defined as follows:

1. $vtype \in \{input, gate, output\}$ denotes the vertex type,
2. $gtype \in \{and, or, \ldots, \perp\}$ denotes the gate type if the vertex type is 'gate' or an 'undefined' symbol $\perp$ otherwise.

Finally, we define the *logical representation of a physical circuit* c as the pair $(G(c), L(c))$, where $G(c)$ is the c-graph and $L(c)$ is the vertex labeling of circuit c. If a physical circuit is available to an adversary, the adversary can choose to generate its own logical abstraction, which may different than what was just defined.

2.2 The circuit camouflaging problem

In general terms, the circuit camouflaging problem asks the following question. Let c be a physical circuit computing a function f; is it possible to transform c into a related circuit that computes an "equivalent" function, building on, among other techniques, "a class of dummy contacting techniques" so to limit an adversary with "some computational resources" and "some type of access to c" in his/her efforts to reverse-engineer c and obtain "some information" about c or f. There are several ways to formalize each of the expressions between quotes in the above description, resulting in multiple distinct formalizations of the circuit camouflaging problem. We specify and then study some of these formulations, by focusing the class of dummy contacting techniques, the adversary model, the camouflage circuit oracle model, and the camouflage correctness and security properties.

Dummy contacts. Contacts are conducting materials that connect adjacent metal layers in a physical gate. While a true contact has no gap and realizes an electrical connection, a dummy contact has a gap in the middle and thus fakes a connection between the layers. From the top view of the IC, which is used by an attacker in reverse engineering, both the true and dummy contacts appear identical, even under a microscope. While the layouts of two distinct physical gates look obviously different (and are hence easy to reverse engineer), the layouts of the same gates, after using dummy contacts, look identical and difficult to differentiate [4, 5, 7, 28]. We refer to papers [12, 14, 16, 24, 25, 28] for more on the class of dummy contacts and other camouflaging techniques.

Adversary model. To formalize the adversary's resources, we consider adversaries as algorithms that run in time polynomial (in a security parameter σ), where the size (i.e., number of gates) of circuit c, denoted as $|c|$, is also polynomial in σ.

In terms of resource access, we consider adversaries with both read and oracle access to resources related to circuits. For a circuit c, we consider adversaries that have *read access* to the logical representation $(G(c), L(c))$ of circuit c, as defined above; we note, however, that an adversary could also generate its own representation of c. An efficient algorithm Adv has *oracle access* to function O if it can run the following attack experiment, for some m polynomial in σ:

1. for $i = 1, \ldots, m$,
   on input $x_1, y_1, \ldots, x_{i-1}, y_{i-1}$, compute x_i;
   call oracle O on input x_i; and
   set y_i be the response obtained from O on input x_i.
2. on input $x_1, y_1, \ldots, x_m, y_m$, return: *out*

We summarize this experiment by the random process denoted as '$out \leftarrow Adv^O(1^\sigma)$'. We say that Adv is an *oracle adversary* if it is an efficient algorithm that is given oracle access to function O.

In our model, the adversary will have access to the logical representation of a camouflaged circuit as well as oracle access to the camouflaged circuit.

Camouflaged circuit oracle model. Using the described dummy contacting techniques, for modeling purposes, we define a *gate camouflaging* transformation $gCam$ that maps a physical gate g into a physical gate g', and assumed to satisfy certain security properties (which is defined in Section II). Analogously, we define a *circuit camouflaging* transformation $cCam$ that maps a physical circuit c to another physical circuit $c' = cCam(c)$, where c' differs from c in that some or all gates in c have been transformed through gate camouflaging, and is assumed to satisfy certain security properties (which we define later).

For every camouflaged physical circuit $c' = cCam(c)$ and its c'-graph $G(c') = (V(c'), E(c'))$, we associate a *vertex labeling* $L(c')$, defined similarly as before, with the extension that we add *camouflaged* as a new gate type; specifically, $L(c')$ maps every $v \in V(c')$ to a pair of attributes $(vtype, gtype)$, defined as follows:

1. $vtype \in \{input, gate, output\}$ denotes the vertex type,
2. $gtype \in \{and, or, \ldots, camouflaged, \perp\}$ denotes the gate type if the vertex type is 'gate' or 'undefined' symbol $\perp$ otherwise.

For any physical gate g, we define the *logical representation of a camouflaged physical gate* $g' = gCam(g)$ as a fixed binary string, with the only property of being different from any logical representation of any (not camouflaged) physical gate. Note that, as defined, this representation does not allow an adversary to compute g'; however, as formalized for circuits, g' can be considered as an oracle that might be evaluated by an adversary. Even after camouflaging, g' can always be evaluated, unless both its inputs are set to a constant (a case that will appear in our construction).

We define the logical representation of a camouflaged physical circuit as an extension of the logical representation of a (not camouflaged) physical circuit, with special attention to gates being camouflaged or not, and treating the camouflaged circuit as an oracle that can be evaluated. Formally, for any physical circuit c, we define the *logical representation of a camouflaged physical circuit* $c' = cCam(c)$ as the triple (c'-graph, $L(c')$, $eval(c')$), where by $eval(c')$ we denote the oracle that evaluates c'.

We express the information available to the adversary in different scenarios of interest, depending on whether all, some, or none of the gates in the given circuit have been camouflaged First of all, in all the three cases, the adversary has access to the c-graph. When $L(c)$ maps some or all vertices of the c-graph to a pair such that $vtype = gate$ and $gtype = camouflaged$, then those vertices correspond to physical gates in c that have been camouflaged, and access to $eval(c')$ guarantees the adversary the capability to evaluate c'. Finally, when no such pairs exist in the vertex labeling $L(c)$, the c-graph and the labeling $L(c)$ contain a full description of circuit c, on which no gate has been camouflaged.

Gate Camouflaging. By applying dummy contacting techniques, every 2-input, 1-output Boolean gate can be camouflaged within all sets of 2-input, 1-output gates. Informally, our formal definition for gate camouflaging consists of two requirements and one (arguably reasonable) assumption. First, we require that a physical gate is camouflaged by an implementation that preserves computation of the same function. Second, we require that camouflaging does not significantly slow down the gate's running time. Finally, we would like to formulate a security requirement for gate camouflaging.

A first intuition is that gate camouflaging does not leak any information on what physical gate was camouflaged. Because, in any reasonable application gates will not exist in isolation but will be part of Boolean circuits, this preliminary intuition of gate camouflaging needs to take into account that an adversary will most likely evaluate the Boolean circuit that contains the camouflaged gate. Evaluation of the circuit might reveal information on what physical gate was camouflaged. Accordingly, we modify the original intuition to say that to any efficient adversary, oracle access to a circuit containing a set of camouflaged gates 'does not help' more than oracle access to the logical representation of the circuit (which is independent on the gate types of the camouflaged gates, and thus provides ideally minimal security leakage on them). Formally, the adversary's output with one type of oracle is computationally indistinguishable from the adversary's output with the other type of oracle, using the computational indistinguishability notion [19] that is pervasive in formalizing security requirements of cryptographic primitives in the literature.

By gS_d we denote a set of 2-input, d-output Boolean gates, for some known $d \geq 1$, and we denote gS_1 as gS.

Definition 2.1. Let $gCam$ be a transformation mapping a physical gate $g \in gS$ to a physical gate g'. We say that $gCam$ is a *gate camouflage transformation* for gate set gS if the logical representation $LogRep(g') = (g'\text{-graph}, L(g'), eval(g'))$ of g' satisfies the following properties:

(1) (Exact functionality): For any gate $g \in gS$, and all $b_0, b_1 \in \{0, 1\}$, it holds that $eval(g')(b_0, b_1) = g(b_0, b_1)$.
(2) (Bounded slowdown): The running time of $eval(g')$ is at most ρ times the running time to evaluate physical gate g, for some small ρ.
(3) (Camouflaging security): For any circuit c, any circuit c' obtained by applying algorithm $gCam$ to (some or all) gates in circuit c, for any distribution D over the set gS of gates, any logical abstraction $LogAbs$, and any efficient algorithm Adv it holds that $|p_0 - p_1| \leq \epsilon$, where
   - $p_0 = \text{Prob}\left[c'' \leftarrow Adv^{eval(c')}(LogAbs(c')) : c'' = c \right]$;
   - $p_1 = \text{Prob}\left[c'' \leftarrow Adv^{eval(c')}(LogRep(c')) : c'' = c \right]$;
   - ϵ is negligible as a function of σ.

For any given transformation $gCam$ between physical gates, the exact functionality and bounded slowdown properties can be verified, while the security property of the camouflaging gates can at best be conjectured to hold (which has been done for camouflaging based on dummy contacts).

Remark on physical aspects. To give more details on known justifications of this assumption, we recall that a reverse engineer may face the following difficulties while identifying the functionality of camouflaged cells through physical means [25, 28]:

(1) Delayering the lower metal layers (M1 and M2) is more difficult than delayering a higher metal layer (M3 and above). Consequently, the dummy contacts placed at those layers cannot be reverse engineered.
(2) A reverse engineer can try to differentiate between a true and a dummy contact by slicing the die and imaging the side-view. However, this is not possible as there are millions of contacts in an IC. Classifying them is a cumbersome task and will not be feasible.
(3) A reverse engineer can use anisotropic techniques like reactive-ion etching to partially etch the layers. However, this causes the the dummy contacts to be eroded because of the chemicals used in the upper layers. One may not know whether a an eroded/dummy contact is because of chemical erosion or camouflaging [3, 6].

Remark on distribution aspects. For simplicity we have used in the definition a single set of gates gS and a single distribution D over gS. We note that it is immediate (although notationally cumbersome) to generalize the definition to multiple gate sets gS_i (up to one for each gate in the circuit), as well as to multiple distributions D_i over gate set gS_i (again, up to one for each gate in the circuit). All our results continue to hold when considering this generalized version of the definition of gate camouflaging.

Remark on security limitations. Let us consider some extreme conditions. First, consider a 1-gate circuit c, and assume this gate is camouflaged, using *gCam*, into a circuit c'; note that oracle access to $eval(c')$ can easily be used to derive the truth table of c. Thus, camouflage security is easily seen to not hold for 1-gate circuits, regardless of how powerful algorithm *gCam* is. Furthermore, limitations on the power of camouflaging similar to this do not only depend on the fact that c has a single gate, as detailed in Section 3. Moreover, under assumptions that are standard in cryptography, for some circuits, oracle access does not help and camouflaging does keep gates secret, as detailed in Section 4.

Remark on adversary goals. Definition 2.1 considers an adversary that, given its resources, attempts to exactly compute the entire circuit with camouflaged gates. It is not hard to obtain similar definitions with alternative and interesting goals for the adversary. For instance, an adversary could be interested in obtain a related circuit which computed the same function only with some probability, or which reasonably well approximates the computation by the original circuit. We defer the investigation of these alternative definitions to the full version of the paper.

Circuit camouflaging. By applying dummy contacting techniques, some or all of the physical gates in a circuit can be camouflaged, by independently camouflaging each gate within all sets of gates with the same number of inputs and outputs. This leads to some form of circuit camouflaging, which we formalize here, by extending the definition for gate camouflaging. Informally, our formal definition for circuit camouflaging consists of two requirements and one assumption.

First, we require that a physical circuit is camouflaged by an implementation that preserves computation of the same function.

Second, we require that camouflaging does not significantly slow down the circuit's running time.

Finally, we would like to formulate a security requirement on circuit camouflaging. A first intuition could be that circuit camouflaging, being defined as a natural application of gate camouflaging, does not leak any information on what physical gates were camouflaged (similar to the requirement for gate camouflaging). As for gate camouflaging, we take into account that an adversary will most likely evaluate the camouflaged circuit. However, differently than gate camouflaging, in circuit camouflaging the goal is to protect the privacy of the circuit, even as the adversary is allowed oracle access to the camouflaged circuit. Accordingly, we modify the original intuition to say that even oracle access to the camouflaged circuit does not allow the adversary to guess the circuit any better than by randomly guessing the gates that were camouflaged, plus at most a negligible probability amount.

By cS we denote a set of n-input, n-output Boolean circuits, for some integer $n \geq 1$.

Definition 2.2. Let *gCam* be a transformation mapping a physical gate $g \in gS$ to a physical gate g', and let *cCam* be a physical transformation mapping a physical circuit $c \in cS$ to a physical circuit c', by applying *gCam* to some or all of the physical gates in c. We say that *cCam* is a *circuit camouflage transformation* for circuit set cS if the logical representation $LogRep(c') = (c'\text{-graph}, L(c'), eval(c'))$ of circuit c' satisfies the following properties:

(1) (Exact functionality): For any n-input circuit $c \in cS$, and all $b_1, \ldots, b_n \in \{0,1\}$, it holds that $eval(c')(b_1, \ldots, b_n) = c(b_1, \ldots, b_n)$.
(2) (Bounded slowdown): The running time of $eval(c')$ is at most ρ times the running time to evaluate physical circuit c, for some small ρ.
(3) (Camouflaging security): For any circuit c, any circuit c' obtained by applying algorithm *cCam* to circuit c, for any distribution D over the set gS of gates, any logical abstraction *LogAbs*, and any efficient algorithm Adv, it holds that $p \leq |gS|^{-t} + \epsilon$, where
   - $p = \text{Prob}\left[c'' \leftarrow Adv^{eval(c')}(LogAbs(c')) : c'' = c \right]$;
   - t is the number of camouflaged gates in c';
   - ϵ is negligible as a function of σ.

Similarly as for gate camouflaging, we note that for any given transformation *cCam* between physical circuits, the exact functionality and bounded slowdown properties can be proved unconditionally, while the camouflaging security property can at best be proved to hold under some assumption (say, on gate camouflaging, which, in turn can be assumed to hold for gate camouflaging transformations based on dummy contacting techniques).

Remark on physical aspects. We notice that even if the camouflaging security property of the transformation *gCam* (applied to a single gate) holds, it needs to be justified why one expects the camouflaging security property the *cCam* transformation (applied to multiple gates) to holds. This can be justified using the fact that the structure of a camouflaged seems to look identical to an attacker, independent of the function implemented by it. Thus, the information on the functionality of one camouflaged gate can neither reveal the functionality of another camouflaged gate through visual inspection, nor help him performing previously mentioned reverse engineering attacks on another camouflaged gate (i.e., delayering, differentiating between true and dummy contact, and using anisotropic techniques).

Remark on distribution aspects. As for gate camouflaging, we note that it is immediate (although notationally cumbersome) to generalize the definition to multiple gate sets gS_i (up to one for each gate in the circuit), as well as to multiple distributions D_i over gate set gS_i (again, up to one for each gate in the circuit). In this case, even the expression of the probability of a random guess of the circuit in the upper bound for p becomes somewhat more complex. All our results continue to hold when considering this generalized version of the definition of circuit camouflaging.

Remark on adversary goals. As for gate camouflaging, it is not hard to obtain similar definitions with alternative and interesting goals for the adversary, such as obtaining a related circuit which computed the same function only with some probability, or which reasonably well approximates the computation by the original circuit, or which tries to distinguish the original circuit from a black-box computing the same function (this latter definition being in the spirit of cryptographic program obfuscation, as defined in [2]). We defer the investigation of these alternative definitions and comparisons with Definition 2.2 to the full version of the paper.

3 LIMITATIONS ON CAMOUFLAGING

A first, natural, approach to camouflage Boolean circuits starting from gate camouflage techniques in Definition 1 is using these techniques to camouflage every gate in the given circuit. It turns out that there are classes of circuits for which this approach completely fails. In this section we prove that this is the case for a family of monotone circuits (i.e., circuits only containing AND and OR gates with fan-out 1), where even camouflaging all gates would not suffice to hide them against an efficient adversary that can evaluate the camouflaged circuit. We start by formally stating our result.

Theorem 3.1. Let mtC be the class of monotone circuits with fan-out-1 gates. Also, let $gCam$ be a gate camouflaging transformation for 2-input, 1-output, physical gates, and let $cCam$ be the circuit camouflaging transformation defined by applying $gCam$ to each gate in the input circuit. There exists an efficient adversary Adv such that for any circuit $c \in mtC$,

$$\text{Prob}\left[c' \leftarrow cCam(c); c'' \leftarrow Adv^{eval(c')}(LogRep(c')) : c'' = c \right] = 1,$$

where $LogRep(c') = (c'\text{-graph}, L(c'), eval(c'))$ is the logical representation of circuit c'.

We note that the adversary claimed in Theorem 3.1 negates Definition 2.2 in a very strong sense, as in particular it computes the exact circuit c with probability 1 (as opposed to, say, only learning partial information about the camouflaged gates better than by randomly guessing). To describe the proof of Theorem 3.1, we first state known and new definitions related to monotone formulae and circuits, and then we prove the theorem by induction over the monotone formula. Finally, we conclude the section with a discussion of the theorem and its possible extensions.

Definitions on monotone formulae. Let $\Phi = \{\phi_n \mid n \in \mathcal{N}\}$ denote a *family of (Boolean) monotone formulae*, where each ϕ_n is a (Boolean) monotone formula over n input bits $x_1, \ldots, x_n$. Recall that a monotone formula can be expressed using only AND and OR Boolean operators. It is well known that any monotone formula can be computed by a monotone circuit with fan-out-1 gates. By $mtC = \{mtC_n \mid n \in \mathcal{N}\}$ we denote the family of circuits computing monotone formulae. Accordingly, monotone circuits in mtC will only have AND and OR Boolean gates, with fan-out 1.

Base case: single-gate monotone formulae. Let mtC_2 be a monotone circuit with a single 2-input, 1-output gate $g \in \{\text{AND, OR}\}$. Also, let $g' = gCam(g)$ and let $LogRep(g') = (g'\text{-graph}, L(g'), eval(g'))$ be the logical representation of g'. We define the following oracle adversary Adv.

Input to Adv: access to a 2-input, 1-output oracle $eval(g')$

Instructions for Adv:

(1) set $b_1 = 0$ and $b_2 = 1$;
(2) if $eval(g')(b_1, b_2) = 0$ then return: AND
    else return: OR.

By the definition of Adv and direct inspection of a single gate's possible truth tables, one sees that Adv's output is a correct guess for gate g with probability 1.

Induction case. Let mtC_n be a monotone circuit over n inputs. We can write, without loss of generality, mtC_n as the circuit computing monotone formula $\phi'_n = \alpha(\beta'_1, \ldots, \beta'_k)$, for some monotone formula α, some camouflaged monotone formulae $\beta'_1, \ldots, \beta'_k$, and some integer $k \geq 1$. In other words, the outputs of camouflaged monotone formulae $\beta'_1, \ldots, \beta'_k$ are inputs to (not camouflaged) monotone formula α, where the latter is assumed, by induction hypothesis, to have been correctly guessed by Adv.

To show the inductive step, we need to show that Adv can use oracle access to $eval(\phi'_n)$ to correctly guess the root camouflaged gate in any one among camouflaged monotone formulae $\beta'_1, \ldots, \beta'_k$.

Without loss of generality, let us pick β'_1, and assume it can be written as $g'(\beta'_{1,L}, \beta'_{1,R})$, for some camouflaged gate g', and camouflaged subformulae $\beta'_{1,L}, \beta'_{1,R}$.

Generalizing the base case, Adv would like to find a setting for input $b_1, \ldots, b_n$ such that $eval(\phi'_n)(b_1, \ldots, b_n) = 0$ (resp., 1) implies that g' is a camouflage of an AND (resp., OR) gate. A direct generalization of the base case approach would consist of setting all input bits to $\beta'_{1,L}$ to 0 and all input bits to $\beta'_{1,R}$ to 1. This setting has the property that $eval(\beta'_1)$ evaluating to 0 (resp., 1) implies that g' is a camouflage of an AND (resp., OR) gate. However, Adv cannot directly observe the $eval(\beta'_1)$, since it has oracle access to $eval(\phi'_n)$. To bypass this problem, in our construction, Adv determines a setting for input $b_1, \ldots, b_n$ in more steps, depending on the gates already guessed in (not camouflaged) monotone subformula α. Specifically, the input bits $b_1, \ldots, b_n$ are set in a way so that $eval(\phi'_n) = eval(\beta'_1)$, which allows Adv to use the mentioned direct generalization of the base case approach.

We now formally define the oracle adversary Adv.

Input to Adv: Access to an n-input, 1-output oracle $eval(\phi'_n)$, which can be written as $\phi'_n = \alpha(\beta'_1, \ldots, \beta'_k)$, for some monotone formula α, some camouflaged monotone formulae $\beta'_1, \ldots, \beta'_k$, and some integer $k \geq 1$.

Instructions for Adv:

(1) Write $\phi'_n = \alpha(\beta'_1, \ldots, \beta'_k)$
(2) write $\beta'_1 = g'(\beta'_{1,L}, \beta'_{1,R})$, for some camouflaged monotone gate g' and subformulae $\beta'_{1,L}, \beta'_{1,R}$
(3) set all b_i input to $\beta'_{1,L}$ as = 0 and all b_i input to $\beta'_{1,R}$ as = 1;
(4) set $h_0 = g'$ and h_1 as the gate parent of g'
(5) repeat
    consider the subformula η of h_1 not including h_0
    if h_1 is an AND gate
      set as 1 all b_i inputs to subformula η
    if h_1 is an OR gate
      set as 0 all b_i inputs to subformula η
    set $h_0 = h_1$ and h_1 as the gate parent (if any) of h_0
(6) until h_0 is the root gate of ϕ'_n;
(7) if $eval(\phi'_n)(b_1, \ldots, b_n) = 0$ then return: $g' =$ AND
    else return: $g' =$ OR.

We now show that Adv's output is a correct guess for gate g such that $g' = gCam(g)$ with probability 1. We start by proving a few properties of Adv and then combine the properties to obtain our main claim.

Lemma 3.2. *Algorithm Adv assigns all input bits $b_1, \ldots, b_n$ exactly once during its execution.*

Proof. Input bits $b_1, \ldots, b_n$ may either belong to the subformula having g' as a root gate or not. All those who do belong are

assigned by Adv in step 3. Those who don't are set in step 5, as the repeat loop steps over all gates h_0 between g' and the root gate of ϕ'_n, where, at each step, it sets all b_i bits in the subformula eta of h_0's parent which does not include h_0. Then the lemma follows by observing that the set of the n inputs to ϕ'_n is the disjoint union of the input bits to the subformula having g' as a root gate and to the subformulae η defined in step 5. □

Lemma 3.3. *As set by algorithm Adv, input bits $b_1, \ldots, b_n$ satisfy $eval(\phi'_n)(b_1, \ldots, b_n) = eval(\beta'_1)(b_1, \ldots, b_n)$.*

Proof. The lemma follows by observing that at each iteration of the loop in step 5 of algorithm Adv the bits b_i input to subformula η are set so to preserve the invariant $eval(\psi)(b_1, \ldots, b_n) = eval(\beta'_1)(b_1, \ldots, b_n)$, where ψ is the subformula with h_1 as root gate. Since h_1 varies from the root gate of β'_1 to the root gate of ϕ'_n, the equality in the lemma statement follows from the mentioned invariant. □

By combining Lemma 3.2 and Lemma 3.3, and by applying inductively over the path between g' and the root gate of ϕ'_n the same reasoning used in the base case of a single-gate monotone formula, we obtain that Adv's output is a correct guess for gate g such that $g' = gCam(g)$ with probability 1.

This concludes the induction case, from which Theorem 3.1 follows.

Theorem extension and discussion. The result in Theorem 3.1 can be extended to all circuits that are learnable, under a suitable definition of computational learning. (In fact, the result could also be stated saying that monotone formulae are learnable in this model.) This model and associated definition of learning seem somewhat different than those more frequently used in computational learning theory (most importantly, PAC-learning), in at least the following 3 important aspects. First, as for the learner's knowledge of the circuit, our result allows the learner knowledge of the structure of the circuit, while in PAC learning this is generally not allowed. Then, as for the learner's access to the circuit, our result allows the learner access to the camouflaged circuit, while in PAC learning the learner views the circuit as a black box. Finally, as for the learner's objective, in our result the learner tries to obtain the camouflaged gates, while in PAC learning the learner tries to obtain a potentially different circuit which well approximates the unknown circuit with high probability. Because of these differences, we believe positive or limitation results in the area of PAC learning do not directly carry to our model.

In light of Theorem 3.1, in the rest of the paper, we focus on very specific families of circuits, with pseudo-randomness (and thus, unlearnability) properties for which we will be able to obtain positive camouflaging results.

4 PROVABLE CAMOUFLAGE OF PSEUDO-RANDOM FUNCTIONS

In this section, we present our main result: a provable circuit camouflage transformation for any class of circuits associated with an arbitrary family of pseudo-random functions. Formally, we obtain the following:

Theorem 4.1. Let prF be a family of pseudo-random functions with n-bit keys, inputs and outputs, and let prC be the class of circuits computing pseudo-random functions in prF. If there exists a gate camouflage transformation $gCam$ satisfying Definition 2.1, there exists (constructively) a circuit camouflage transformation $cCam$ for prC that satisfies Definition 2.2 and returns a circuit with a set of κ camouflaged gates. Moreover, $cCam$ returns a family of camouflaged circuits roC' such that $roC'(x) = prF(k, x)$, for some k randomly distributed in $\{0,1\}^\kappa$.

There are two statements of interest in Theorem 4.1: the first statement says that there exists a circuit for which gate camouflaging (in the sense of Definition 2.1) implies circuit camouflaging (in the sense of Definition 2.2); the second statement says that, after camouflaging, this circuit behaves like a pseudo-random function with a random key, which is computationally indistinguishable from a random oracle, by definition of pseudo-random functions [18].

We divide the proof of (the first part of) Theorem 4.1 in 4 steps, with relative subsections. First, in Section 4.1 we recall basic notions and introduce new notions about pseudo-random functions. Then, in Section 4.2 we describe a first physical circuit transformation that transforms circuits computing (keyed) pseudo-random functions to circuits computing pseudo-random functions where the key is instantiated. In Section 4.3 we describe a second physical circuit transformation that uses gate camouflaging to transform circuits computing pseudo-random functions where the key is instantiated into analogue circuits where gates associated with key bits are camouflaged. Finally, in Section 4.4 we show that the resulting camouflaged circuit family satisfies Definition 2.2, and therefore Theorem 4.1. The second part of Theorem 4.1, namely, that $cCam$ returns a family of camouflaged circuits roC' such that $roC'(x) = prF(k, x)$, for some k randomly distributed in $\{0,1\}^\kappa$, directly follows from the proof for the first part.

4.1 Random and Pseudo-random Functions

We recall formal definitions of random and pseudo-random functions, and define a notion of key-set pseudo-random functions.

Random functions. A function $R_n : \{0,1\}^n \to \{0,1\}^n$ is a *random function over* $\{0,1\}^n$ if it is randomly chosen among all functions with n-bit inputs and n-bit outputs. In a random function R_n, for any input string $x \in \{0,1\}^n$, the output string $R_n(x) \in \{0,1\}^n$ is uniformly and independently distributed. We say that a family of functions $\{R_n : n \in \mathcal{N}\}$ is a *family of random functions* if each function R_n is a random function over $\{0,1\}^n$. As a consequence, an adversary querying R_n on several input strings and obtaining the corresponding output strings still cannot predict the output string $R_n(x)$ corresponding to a new input string x, better than by randomly choosing a string of the same length.

As any logical description of a random function over $\{0,1\}^n$ requires $\Omega(2^n)$ space, families of random functions cannot be efficiently described. Pseudo-random functions are widely used to approximate the properties of random functions in in both theoretical research and (a large number of) practical applications [18]. Their evaluation only requires a short random key, and their pseudo-randomness property holds as long as the key used to evaluate them is kept secret.

Pseudo-random functions. A function $prF_n : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a *keyed function over* $\{0,1\}^n$ if for each $k \in \{0,1\}^\kappa$, the resulting function $prF_n(k,\cdot)$, also denoted $prF_k(\cdot)$, is a function with n-bit inputs and n-bit outputs.

For any $n \in \mathcal{N}$, let $Rand_n$ be the set of all functions $R_n : \{0,1\}^n \rightarrow \{0,1\}^n$ and let $prF_k : \{0,1\}^n \rightarrow \{0,1\}^n$ be a keyed function. Consider the following probabilistic experiment $Init$:

1. Uniformly choose R_n from $Rand_n$; and
2. uniformly choose k from $\{0,1\}^\kappa$.

We say that a family of functions $prF = \{prF_n : n \in \mathcal{N}\}$ is a *family of pseudo-random functions* if for any efficient oracle adversary Adv, the difference $|p_R - p_{prF}|$ is negligible in n, where

1. $p_R = \{Init; O(\cdot) \leftarrow R_n(\cdot); A^O(1^n) = 1\}$; and
2. $p_{prF} = \{Init; O(\cdot) \leftarrow prF_n(k,\cdot); A^O(1^n) = 1\}$;.

To the family of functions prF, we can associate a family of circuits prC that computes the same family of functions.

Key-set pseudo-random functions. Let prF be a family of pseudo-random functions, and let $K = \{k_n : n \in \mathcal{N}\}$ be a sequence of keys such that k_n is uniformly distributed over $\{0,1\}^n$ for all $n \in \mathcal{N}$. We define the family of *K-keyed pseudo-random functions* prF_K, as the family of functions such that $prF(k_n, x) = prF_K(x)$ for all $x \in \{0,1\}^n$ and all $n \in \mathcal{N}$. To the family of K-keyed pseudo-random functions prF_K, we can associate a family of circuits prC_K that computes the same family of functions.

Pseudo-random functions well approximate the properties of random functions in all applications where the key used to evaluate them is kept secret. However, in some applications, we cannot afford to keep the key secret from the adversary, and the stronger properties of random functions happen to be extremely useful to obtain plausibility and/or efficiency results. Because of their wide usefulness and applicability, practitioners often approximate random functions with efficient cryptographic functions like block ciphers (e.g., AES) or hash functions (e.g., SHA3), which, while known to be not random, are conjectured to have somewhat close randomness properties.

4.2 A Procedure to Hardware Keys

Our circuit transformation uses a procedure that involves no camouflaging and takes as input a key k and the family prC of circuits associated with the pseudo-random functions prF. This procedure's goal is to pre-process the circuit, based on key k, before applying gate camouflaging. The pre-processing consists of hardwiring the input key k into the circuit and then suitably encoding each key bit into the circuit as a gate, where the encoding satisfies the following two properties:

(1) the gate differs depending on the key bit value; and
(2) the gate's output is equal to the key bit value.

Thus, this procedure turns a circuit with a key and a second string as inputs into a circuit with a single string as input, getting us one step closer to realizing a random oracle. Now, we proceed more formally.

Formal description: Let prF denote a family of pseudo-random functions, and let prC denote the associated family of circuits computing the same functions. Then, we consider the associated family $prF(k,\cdot)$ of k-keyed pseudo-random functions, and the associated family $prC_k(\cdot)$ of circuits computing the same functions.

Input to KH**:** circuit $prC_n \in prC$ associated with function $prF_n(\cdot,\cdot) \in prF$, key $k \in \{0,1\}^\kappa$

Instructions for KH**:**

(1) let $k = k_1|\cdots|k_\kappa$, with $k_i \in \{0,1\}$, for $i = 1,\ldots,\kappa$;
(2) for each $i = 1,\ldots,\kappa$,
    let h_i denote a gate with 2 input bits set = 1;
    if $k_i = 0$ then let h_i be a NAND gate;
    if $k_i = 1$ then let h_i be an AND gate;
(3) consider k-keyed pseudo-random function $prF(k,\cdot)$
(4) consider the associated circuit $prC_k(\cdot)$ computing the same function
(5) set $roC_k(\cdot)$ equal to $prC_k(\cdot)$
(6) for each input wire in circuit $roC_k(\cdot)$ using bit k_i,
    add one output wire for gate h_i;
(7) in circuit $roC_k(\cdot)$, do the following:
    replace input k_i with gate h_i;
(8) return: roC_k.

It is not hard to see that other settings for h_i would have also worked in guaranteeing h_i to be set as a different gate depending on the key bit value, without modifying the computation of the overall circuit.

4.3 The Circuit Transformation

On input a circuit associated with a pseudo-random function, our circuit transformation randomly chooses a key k and hardwires it into this circuit using procedure KH. Let roC_k be the family of circuits returned by this procedure execution. Then the circuit transformation continues by applying gate camouflaging to this circuit. It turns out that for our goal it only suffices to camouflage the newly introduced gates $h_1,\ldots,h_\kappa$ associated with the key bits $k_1,\ldots,k_\kappa$. Here, the intuition is that camouflaging of these gates is used to hide these key bits from the adversary. Then, once the key bits are hidden, the properties of pseudo-random functions can be used to show that the adversary's evaluation of the camouflaged circuit does not help in learning the camouflaged gates and therefore the key bits of the original pseudo-random function. Now, we proceed more formally.

Formal description: Let $prF(k,\cdot)$ denote a family of k-keyed pseudo-random functions, and let $prC_k(\cdot)$ denote the associated family of circuits computing the same function, and let KH be the procedure described in Section 4.2.

Input to $cCam$**:** circuit roC_k

Instructions for $cCam$**:**

(1) randomly choose $k \in \{0,1\}^\kappa$
(2) let $roC_k = KH(k, prC_k)$
(3) let $h_1,\ldots,h_\kappa$ denote the gates added to roC_K during procedure KH;
(4) for each $i = 1,\ldots,\kappa$,
    let $h'_i = gCam(h_i)$
    replace gate h_i with camouflaged gate h'_i;
(5) let roC'_k denote the resulting camouflaged circuit;
(6) return: roC'_k.

4.4 Properties of the Camouflaged Circuit

In what follows, we prove that our camouflage transformation *cCam* satisfies Theorem 4.1. To this, we need to prove that it satisfies the three properties of Definition 2.2: exact functionality, bounded slowdown, and security properties of camouflaging; as well as the two remaining items in the theorem statement. We start from the latter: we observe that

Exact functionality. The family of circuits prC and its camouflaged version roC' only differ in some of the inputs, where prC uses key bits $k_1, \ldots, k_\kappa$, and roC' uses some camouflaged gates $h'_1, \ldots, h'_\kappa$. Specifically, in the first step of circuit camouflage transformation *cCam* on input circuit prC, each input key bit k_i is replaced by gate h_i. In the second step, each gate h_i is replaced by its camouflaged implementation h'_i returned by $gCam(h_i)$. By the exact functionality property of gate camouflaging algorithm *gCam*, $eval(h'_i)$ computes the same function as h_i. By construction of gate h_i, in step 2 of procedure *KH*, its output is equal to bit k_i, for all $i = 1, \ldots, \kappa$ and any $k_i \in \{0,1\}$. Therefore, roC'_k computes the exact same function $prC(k, \cdot)$, for any $k \in \{0,1\}^n$.

Bounded slowdown. An execution of camouflaged algorithm *cCam* on input circuit prC only results in an increase of κ gates, as each input key bit k_i is replaced by a gate h_i and then a camouflaged version of that gate, for $i = 1, \ldots, \kappa$. Thus, we have $|roC'| \leq |prC| + \kappa$, from which the desired property follows.

Camouflage security. Let c be a Boolean circuit. If rP is a random process, we let $Pr_{cCam}[rP]$ denote the probability

$$\text{Prob}\left[\, k \leftarrow \{0,1\}^\kappa; c' = cCam(c_k); rP : c'' = c \,\right].$$

Our goal is to prove that for any distribution D over cS, any logical abstraction *LogAbs* of physical circuit $c' = cCam(c)$, and any efficient algorithm Adv, it holds that $p \leq |gS|^{-t} + \epsilon$, for some function ϵ negligible in the security parameter σ, where:

(1) $p = Pr_{cCam}[c'' \leftarrow Adv^{eval(c')}(LogAbs(c'))]$; and
(2) t is the number of gates camouflaged in c'.

We prove this property by a modified version of a hybrid argument [19], where, informally speaking, we evaluate and compare the success of adversary Adv in guessing circuit c'', and thus the κ camouflaged gates according to *cCam*, in the following 4 "worlds":

1. Adv has read access to $LogAbs(c')$ and oracle access to $eval(c')$;
2. Adv has read access to $LogRep(c')$ and oracle access to $eval(c')$;
3. Adv has read access to $LogRep(c')$ and oracle access to a random oracle R_n; and
4. Adv', depending on Adv, has read access to $LogRep(c')$.

More formally, we first define the following random processes:

$rP_1 =$ "$c'' \leftarrow Adv^{eval(c')}(LogAbs(c'))$",
$rP_2 =$ "$c'' \leftarrow Adv^{eval(c')}(LogRep(c'))$",
$rP_3 =$ "$c'' \leftarrow Adv^{R_n}(LogRep(c'))$",
$rP_4 =$ "$c'' \leftarrow Adv'(LogRep(c'))$".

Then we show the following lemmas.

Lemma 4.2. $p = Pr_{cCam}[rP_1]$

Proof. This directly follows from definitions of p, rP_1. □

Lemma 4.3. $|Pr_{cCam}[rP_1] - Pr_{cCam}[rP_2]| \leq \epsilon_{gcam}$, *for some function* ϵ_{gcam} *negligible in* σ.

Proof. Let p_0, p_1 be the probability quantities defined in Definition 2.1. By the definitions of rP_1, rP_2, we observe that $Pr_{cCam}[rP_1] = p_0$ and $Pr_{cCam}[rP_1] = p_0$. Then the lemma follows as a direct application of Definition 2.1. □

Lemma 4.4. $|Pr_{cCam}[rP_2] - Pr_{cCam}[rP_3]| \leq \epsilon_{prf}$, *for some function* ϵ_{prf} *negligible in* σ.

Proof. We first observe that the difference between rP_2 and rP_3 only consists of the oracle to which Adv has oracle access. Specifically, in rP_2, Adv has access to $eval(c')$, an oracle that evaluates the circuit c' for pseudo-random function prF, while in rP_3, Adv has access to a random oracle R_n. Moreover, the input $LogRep(c')$ to Adv in both rP_2 and rP_3 does not depend on the camouflaged gates in c' (which, in turn, encode the bits of key k), by definition of the *LogRep* function. This latter fact is critical to apply the pseudo-randomness property of prF, and obtain that Adv can only distinguish the two worlds with at most negligible probability ϵ_{prf}. (In particular, note that the pseudo-randomness property of prF might not suffice to prove that $|Pr_{cCam}[rP_1] - Pr_{cCam}[rP_3]|$ is negligible, since $LogAbs(c')$ is a function of the camouflaged gates in c' in $rP1$). □

Lemma 4.5. *For any efficient algorithm* Adv, *there exists an efficient algorithm* Adv' *such that* $Pr_{cCam}[rP_4] = Pr_{cCam}[rP_3]$.

Proof. Consider algorithm Adv in random process rP_3. Then, define algorithm Adv' as the algorithm that runs Adv and, in this execution, simulates the answers to Adv's queries to the random oracle as random strings (for new queries) or previously generated random strings (for repeated queries); finally, Adv' returns the same output as Adv. Since the simulation performed by Adv' of the random oracle answers is perfect, it holds that $Pr_{cCam}[rP_4] = Pr_{cCam}[rP_3]$, from which the lemma follows. □

Lemma 4.6. *For any efficient algorithm* Adv', *we have that*

$$Pr_{cCam}[rP_4] \leq |gS|^{-t}.$$

Proof. Note that in random process rP_4, algorithm Adv' has no access to camouflaged gates or to an oracle for c'. Accordingly, at best, it can randomly guess the value of the gates input to *gCam*. The lemma follows by observing that there are t such gates, and they are uniformly and independently drawn from set gS. □

Finally, we use Lemma 4.2, 4.3, 4.4, 4.5, and 4.6 to conclude the proof that *cCam* satisfies Definition 2.2. Specifically, we have that

$$\begin{aligned}
p &= Pr_{cCam}[rP_1] \\
&\leq |Pr_{cCam}[rP_1] - Pr_{cCam}[rP_4]| + Pr_{cCam}[rP_4] \\
&\leq |Pr_{cCam}[rP_1] - Pr_{cCam}[rP_4]| + |gS|^{-t} \\
&\leq |Pr_{cCam}[rP_1] - Pr_{cCam}[rP_2]| \\
&\quad + |Pr_{cCam}[rP_2] - Pr_{cCam}[rP_3]| \\
&\quad + |Pr_{cCam}[rP_3] - Pr_{cCam}[rP_4]| + |gS|^{-t} \\
&\leq \epsilon_{gcam} + |Pr_{cCam}[rP_2] - Pr_{cCam}[rP_3]| \\
&\quad + |Pr_{cCam}[rP_3] - Pr_{cCam}[rP_4]| + |gS|^{-t} \\
&\leq \epsilon_{gcam} + \epsilon_{prf} + |Pr_{cCam}[rP_3] - Pr_{cCam}[rP_4]| + |gS|^{-t} \\
&\leq \epsilon_{gcam} + \epsilon_{prf} + |gS|^{-t},
\end{aligned}$$

where the first equality follows from Lemma 4.2, the first and third inequalities follow by applying the triangle inequality, the second inequality follows from Lemma 4.6, the fourth inequality follows from Lemma 4.3, the fifth inequality follows from Lemma 4.4, and the last inequality follows from Lemma 4.5.

5 CONCLUSIONS

Recent hardware advances have opened the possibility for protection of integrated circuits against reverse-engineering attacks via gate camouflaging [12, 14, 21, 24, 25, 28]. While many of these works consider general Boolean functions in specific attack model, here we consider specific set of Boolean functions in general attack models.

We propose a formal model for the analysis and proof of functionality, efficiency and security properties for circuit camouflaging, inspired by the formal models in [2] for cryptographic (software) program obfuscation. We propose formal definitions for security of gate camouflaging and circuit camouflaging. We use these definitions to show the first provable limitations and constructions results for circuit camouflaging, assuming hardware constructions guaranteeing gate camouflaging. Our construction also shows that circuit camouflaging can be used to obtain one practical realization of a random oracle, a long-sought object from the cryptographic community (which was proved to be very unlikely to exist without hardware [9]).

A large number of open problems is uncovered by these results, including finding more limitations (following our result in Theorem 3.1), more constructions (following our result in Theorem 4.1), and applications of our hardware-based realization of a random oracle, as implied by our construction underlying Theorem 4.1. Our future work includes realizing hardware implementations of these results to obtain the power, area, and delay overheads.

6 ACKNOWLEDGMENTS

Many thanks to Yevgeniy Dodis for interesting discussions. Part of the first author's work was supported by the Defense Advanced Research Projects Agency (DARPA) via U.S. Army Research Office (ARO), contract number W911NF-15-C-0233. Part of the second author's work was supported by the NSF Award # CNS-1652842. The third and fourth authors are part of Center for Cybersecurity at NYU and NYU-AD. The third author is funded by ARO Award # W911NF-15-1-0278. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, ARO, NSF, or the U.S. Government.

REFERENCES

[1] Lisa Bahler, Giovanni Di Crescenzo, Yuriy Polyakov, Kurt Rohloff, and David B. Cousins. 2017. Practical implementations of lattice-based program obfuscators for point functions. In *International Conference on High Performance Computing & Simulation, HPCS 2017, Genoa, Italy, July 2017.*

[2] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. 2012. On the (im)possibility of obfuscating programs. *J. ACM* 59, 2 (2012), 6.

[3] James P. Baukus, Lap-Wai Chow, Jr. William M. Clark, and Gavin J. Harbison. 2012. Conductive channel pseudo block process and circuit to inhibit reverse engineering. *US Patent no. 8258583* (2012).

[4] James P. Baukus, Lap Wai Chow, and William Clark. 2002. Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide. *US Patent no. 20020096776* (2002).

[5] James P. Baukus, Lap Wai Chow, Ronald P. Cocchi, Paul Ouyang, and Bryan J. Wang. 2012. Building block for a secure CMOS logic cell library. *US Patent no. 8111089* (2012).

[6] James P. Baukus, Lap Wai Chow, Ronald P. Cocchi, Paul Ouyang, and Bryan J. Wang. 2012. Camouflaging a standard cell based integrated circuit. *US Patent no. 8151235* (2012).

[7] James P. Baukus, Lap Wai Chow, Ronald P. Cocchi, and Bryan J. Wang. 2012. Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing. *US Patent no. 20120139582* (2012).

[8] Mihir Bellare and Igors Stepanovs. 2016. Point-Function Obfuscation: A Framework and Generic Constructions. In *Proc. of TCC 2016-A2.* 565–594.

[9] Ran Canetti, Oded Goldreich, and Shai Halevi. 2004. The random oracle methodology, revisited. *J. ACM* 51, 4 (2004), 557–594.

[10] Chipworks. 2010. Reverse Engineering Software. https://tinyurl.com/yb9zkkpo. (2010).

[11] Chipworks. 2012. Intel's 22-nm Tri-gate Transistors Exposed. https://tinyurl.com/y7ukldbp. (2012).

[12] M. I. M. Collantes, M. El Massad, and S. Garg. 2016. Threshold-Dependent Camouflaged Cells to Secure Circuits Against Reverse Engineering Attacks. *IEEE Computer Society Annual Symposium on VLSI* (2016), 443–448.

[13] Giovanni Di Crescenzo, Lisa Bahler, Brian A. Coan, Yuriy Polyakov, Kurt Rohloff, and David B. Cousins. 2016. Practical implementations of program obfuscators for point functions. In *Proc. of HPCS 2016.* 460–467.

[14] Joseph Davis, Niranjan Kulkarni, Jinghua Yang, Aykut Dengi, and Sarma Vrudhula. 2016. Digital IP Protection Using Threshold Voltage Control. *IEEE Int. Symposium on Quality Electronic Design* (2016), 344–349.

[15] Degate. [n. d.]. http://www.degate.org/documentation/. ([n. d.]).

[16] B. Erbagci, C. Erbagci, N. E. C. Akkaya, and K. Mai. 2016. A secure camouflaged threshold voltage defined logic family. *IEEE Int. Symp. on Hardware-Oriented Security and Trust* (2016), 229–235.

[17] ExtremeTech. 2012. iPhone 5 A6 SoC reverse engineered, reveals rare hand-made custom CPU, and tri-core GPU. https://tinyurl.com/9yn23he. (2012).

[18] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to construct random functions. *J. ACM* 33, 4 (1986), 792–807.

[19] Shafi Goldwasser and Silvio Micali. 1984. Probabilistic Encryption. *J. Comput. Syst. Sci.* 28, 2 (1984), 270–299.

[20] Neal Koblitz and Alfred J. Menezes. 2015. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptography* 77, 2-3 (2015), 587–610.

[21] Meng Li, Kaveh Shamsi, and et al. Meade. 2016. Provably Secure Camouflaging Strategy for IC Protection. *IEEE/ACM Int. Conference on Computer-Aided Design* (2016), 28:1–28:8.

[22] D. Liu, C. Yu, X. Zhang, and D. Holcomb. 2016. Oracle-guided Incremental SAT Solving to Reverse Engineer Camouflaged Logic Circuits. *IEEE/ACM Design, Automation and Test in Europe* (2016), 433–438.

[23] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. 2004. Positive Results and Techniques for Obfuscation. In *Proc. of EUROCRYPT 2004.* 20–39.

[24] Ithihasa Reddy Nirmala, Deepak Reddy Vontela, Swaroop Ghosh, and Anirudh Iyengar. 2016. A Novel Threshold Voltage Defined Switch for Circuit Camouflaging. *IEEE European Test Symposium* (2016).

[25] J. Rajendran, M. Sam, O. Sinanoglu, and R Karri. 2013. Security Analysis of Integrated Circuit Camouflaging. *ACM Conference on Computer Communications and Security* (2013).

[26] Semi. 2008. Innovation Is At Risk As Semiconductor Equipment And Materials Industry Loses Up To $4 Billions Annually Due To IP Infringement. https://tinyurl.com/y8rf9v8t. (2008).

[27] Silicon Zoo. [n. d.]. The Layman's Guide to IC Reverse Engineering. http://siliconzoo.org/tutorial.html. ([n. d.]).

[28] SypherMedia. [n. d.]. SypherMedia Library Circuit Camouflage Technology. http://bit.ly/2xvXrOE. ([n. d.]).

[29] R. Torrance and D. James. 2011. The state-of-the-art in semiconductor reverse engineering. *in the Proc. of IEEE/ACM Design Automation Conference* (2011), 333–338.

[30] Hoeteck Wee. 2005. On obfuscating point functions. In *Proc. of 37th ACM STOC, 2005.* 523–532.

[31] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. 2016. CamoPerturb: secure IC camouflaging for minterm protection. In *Proc. of 35th ICCAD, 2016.* 29.

[32] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. 2017. Removal Attacks on Logic Locking and Camouflaging Techniques. *IEEE Trans. on Emerging Topics in Computing* 99, 0 (2017).