Development of a Layout-Level Hardware Obfuscation Tool

Shweta Malik[1], Georg T. Becker[2], Christof Paar[1,2], Wayne P. Burleson[1]
[1] *University of Massachusetts Amherst*
[2] *Horst Görtz Institute for IT-Security, Ruhr University Bochum*

***Abstract*—While hardware obfuscation has been used in industry for many years, very few scientific papers discuss layout-level obfuscation. The main aim of this paper is to start a discussion about hardware obfuscation in the academic community and point out open research problems. In particular, we introduce a very flexible layout-level obfuscation tool that we use as a case study for hardware obfuscation. In this obfuscation tool, a small custom-made obfuscell is used in conjunction with a standard cell to build a new obfuscated standard cell library called *Obfusgates*. This standard cell library can be used to synthesize any HDL code with standard synthesis tools, e.g. Synopsis Design Compiler. However, only obfuscating the functionality of individual gates is not enough. Not only the functionality of individual gates, but also their connectivity, leaks important important information about the design. In our tool we therefore designed the obfuscation gates to include a large number of "dummy wires". Due to these dummy wires, the connectivity of the gates in addition to their logic functionality is obfuscated. We argue that this aspect of obfuscation is of great importance in practice and that there are many interesting open research questions related to this.**

***Keywords*-Layout-level hardware obfuscation, reverse-engineering**

I. Introduction

There are many reasons why hardware companies want to prevent their designs from being reverse-engineered. One of the main reasons is to prevent intellectual property (IP) theft and counterfeiting. IP theft and counterfeits products are huge problem in the IT industry. According to a SEMI report[1] approximately $4 billion is lost due to IP infringement, which includes counterfeiting through reverse engineering, theft of trade secrets and trade marks. Furthermore, for security critical devices, attackers can use hardware reverse-engineering to reveal design details that can compromise the security of the system. For example, an attacker might be able to identify the deployed security mechanisms using reverse-engineering and how to circumvent them.

One approach to combat reverse-engineering is the use of layout-level hardware obfuscation. In layout-level hardware obfuscation the design is implemented using gates that are designed to be especially hard to reverse-engineer. While layout-level hardware obfuscation has been of interest to the industry till the 90s, there has been very little work on the topic in academia. Most scientific papers dealing with layout-level obfuscation have only been published in the last two years. The goal of this paper is to close this gap and and to initiate more scientific debates on layout-level obfuscation. The tool we introduce in this paper is based on an obfuscation gate that uses dopant modifications to combat optical reverse-engineering. Using dopant modifications to prevent reverse-engineering is not new and the industry and is already been used for years in the industry [2], [3].

The idea behind dopant obfuscation is to build gates which are identical on all routing layers (metal and polysilicon) and only differ in the dopant polarity of the active area. Depending on the dopant configurations, the gates can have many different logic functions. Since it is harder to determine the dopant polarity than the metal or polysilicon layer, an attacker can not differentiate between the different gates, and hence cannot reverse-engineer the design. At least this is the goal. However, while it is more difficult to determine the dopant polarity of a gate, it is not impossible. For example, recently Sugawara *et al.* showed how such a dopant obfuscation cell can be reverse-engineered using a SEM by looking at the contact layer [4].

II. Related Work

Layout-level hardware Obfuscation has been of interest to industry since the 90s [3] and has been used in many chips in the past. One of the companies providing layout-level hardware obfuscation techniques is for example SypherMedia International (SMI). They use two fundamentally different approaches for layout-level obfuscation [2], [5]. In the first approach, they use custom made obfuscation gates that are used instead of a standard cell library to obfuscate the designs. The second approach is to modify existing cells of a standard cell library to create obfuscated cells that look identical to the standard cells. In the first approach a single obfuscation gate can have many different functionality and hence offers more obfuscation per gate. The advantage of the second approach is that the attacker does not know which parts of the design is being obfuscated or that obfuscation was used at all. Hence, if only a few obfuscation gates are used in the design, the second

approach is favorable, while the first approach should be used if an entire part of a design is being obfuscated. The individual gates are being obfuscated based on the dopant polarity, similar as it is being proposed in this paper. A technique suggested by Rajendran *et al.* uses a mixture of real and dummy contacts to camouflage the functionality of the obfuscated gates [6]. The main idea behind this technique is that extra effort is needed to reverse-engineer the contact layer. In [7] an obfuscation gate that is also based on the dopant polarity has been proposed. In this approach a single 2-input look-alike gate is used. This gate is basically a 2-input look-up-table. The value of the look up table can be configured based on the dopant polarity and hence their obfuscation gate can function as any 2-input function without changing any other layer than the dopant layer.

However, the authors also showed at CHES 2014 that detecting such a dopant-based obfuscation gate can be reverse engineered using a technique called passive voltage contrast using a SEM [4]. In their proposed reverse-engineering method, they use a technique from failure analysis called passive voltage contrast to successfully reverse-engineer such dopant based obfuscation gates. Hence, while currently proposed circuit level obfuscation techniques increase the costs of reverse-engineering, they cannot completely prevent it. At a higher level, the paper by Rajendran *et al.* looks at how to efficiently reverse-engineer designs efficiently in which only a few gates are being obfuscated using camouflage gates [6]. But all in all, there are relatively few scientific papers about layout-level hardware obfuscation which is quite surprising considering the attention it has gained in the industry.

In the following we will first introduce our obfuscation tool and will then present some overhead results for different designs. In Section V the importance of not only obfuscating the individual gates but also the connectivity between gates is highlighted at the example of the PRESENT block cipher. The paper is concluded with a discussion of open research problems.

III. Overview of Obfusgate

The basic tool-flow of the obfuscation tool is depicted in Figure 1. A Obfuscation Library is created in which each gate only differs in the dopant layer but is identical in all other layers (i.e. polysilicon, metal, contacts, wells and active area). This library can be used like any other standard cell library to synthesis any HDL code using standard tools such as Synopsys Design Compiler. After synthesis, an additional step called the "Wiring Step" is added to the standard design flow. The Wiring Step adds randomness to the design process, so that calling the obfuscation tool twice with the same HDL code does not result in the same obfuscated design. Besides adding randomness to the design process, the main purpose of the Wiring Step is to obfuscate the connectivity of the design.

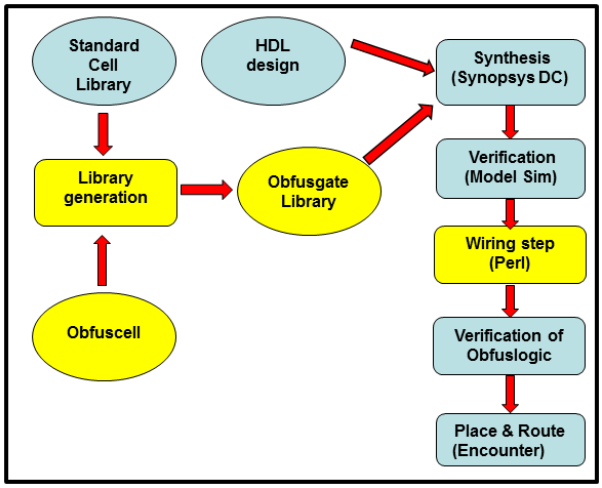


Figure 1. Overview of the obfuscation tool-flow. In a one-time process a Obfuscation Library is created based on the custom Obfuscell and the standard cell library. This is used as an input to the standard synthesize step. After synthesize, an additional step called the "wiring" step is used to introduce to make non-deterministic changes to the design and obfuscate the connectivity using dummy wires and inputs.

The obfuscation library is based on an obfuscation gate called "Obfusgate", which is depicted in Figure 2. It consists of five so-called "Obfuscells" and a 4-input NAND gate. Depending on the dopant polarity within the active area of the Obfuscell, the Obfuscell can have four different logic functions. The Obfuscell is either an inverter, a buffer, or it outputs a constant 1 or 0. The Obfuscell with different configurations is depicted in Figure 3 and 4. The design is similar to the dopant-level hardware Trojans from [8] and is based on the same idea as the obfuscation gates used in [5], [7]. Depending on how the Obfuscells are configured, the Obfusgate can have many different logic functions. In total, 162 different configurations, each with a unique logic behavior are possible with the Obfusgate as depicted in Figure 2. In particular, by setting one of the Obfuscells that are connected to an input of the Obfusgate to a constant '1', this input has effectively been turned into a "Dummy Input". That is, the value of the corresponding input does not influence the output of the Obfusgate. Therefore, any signal can be connected to such an input, creating "Dummy Wires". In practice, many Dummy Inputs are available in such a design, since 2-input gates are the most commonly used gates in a typical design. For example, in our case study of an AES SBox, more than half of the gates were two-input gates.

Our obfuscation tool uses these dummy inputs and

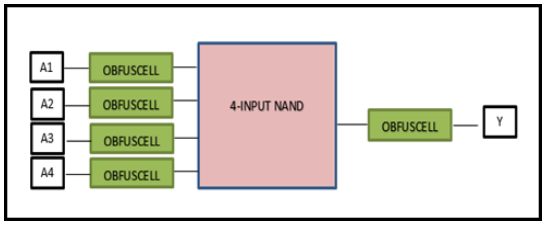
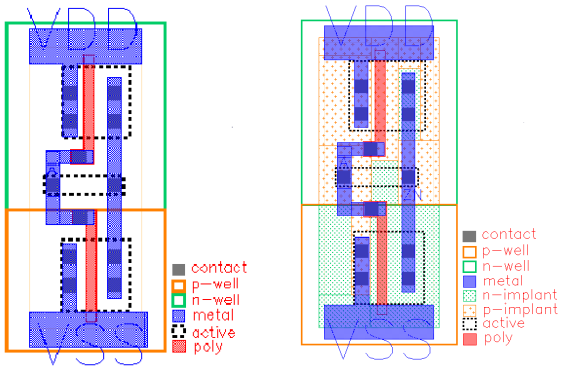


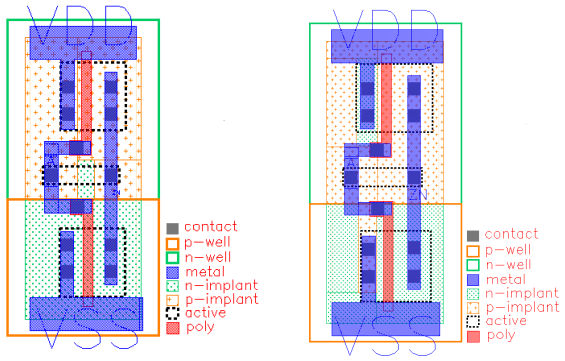
Figure 2. Schematic of a single Obfusgate that consists of 5 Obfuscells together with a 4-input NAND gate. Depending on the configuration of the Obfuscells the Obfusgate can have 162 different logic functions.



(a) (b)

Figure 3. Layout view of the obfuscell. On the left one can see the basic structure of the obfuscell. The obfuscell has three active regions who's dopant polarity defines the logic function of the gate. On the right the dopant configuration is shown that will result in an "Always 1 gate".

dummy wires to connect otherwise unconnected gates and blocks. Due to the large amount of dummy wires, an attacker will not be able to determine the connectivity of the gates. This makes it much harder for an attacker to determine individual blocks in the design and therefore to identify the used architecture or structure. This aspect of the obfuscation tool will be discussed in more detail in Section V. One advantage of our approach is that it is very flexible and different gates can be used in the center of the Obfusgate. For example, using a 3-input NAND gate instead of the 4-input NAND gate can decrease the introduced area overhead of the Obfusgates at the cost of less obfuscation. The main advantage is that to do such trade-offs, no additional custom cell design is needed. Once the Obfuscell has been designed, it can be combined easily and automatically with any logic gate of a standard cell library. We used only one Obfusgate design based on a 4-input NAND gate in our implementation. However, if the designer accepts more than one look-alike gates (i.e. different obfusgate designs), additional obfuscation gates can be integrated into the library easily. For example, the designer could add an Obfusgate based on a 6-input And-Or-Inverter



(a) (b)

Figure 4. Layout view of the obfuscell when configure (a) as an inverter or (b) as a buffer in which the output is the same as the input.

gate (AOI222). In this case, the number of gates after the synthesis step can be reduced since additional gates are available to the synthesize tool. But this comes with the penalty that some information about the used gates is revealed. The fact that only a single custom made obfuscell is needed also makes porting the obfuscation tool to different technologies very simple and cheap. Furthermore, we have automated the generation of the library so that when a new library is created, only the middle gate needs to be defined. Everything else can be done by the tool.

IV. Implementation results

We implemented our design in 45nm technology using the Nangate Open Cell Library [9]. Synthesis was done using Synopsys Design Compiler and place and route was performed using Cadence Encounter and the custom cell design using Cadence Virtuoso. The wiring step and the library generation was performed using a Perl script and the design has been verified using Modelsim. In the first experiment we obfuscated an SBox of the Advanced Encryption Standard (AES) using our tool. The AES SBoxes is a function with 8-input bits that produces 8 output bits. We used a look-up table implementation for the AES SBox in HDL and used a design synthesized using the entire Nangate Open Cell Library as a reference implementation. Two different Obfusgates were tested, one based on a 4-input NAND and one based on a 2-input AND gate. For an Obfusgate based on a 4-input NAND gate, in total $(3)^4 \cdot (2) = 162$ different configurations for one gate are possible. For the 2-input AND gate there are only $3^2 * 2 = 18$ possible combinations. In both case this also includes the gates that output a constant '1' and constant '0' which we actually did not need in our design. We used Cadence Encounter for the place and route step and also used

Encounter for an estimation of the power and delay overhead introduced by the obfuscation tool. Table II summarizes the overhead for the AES SBox. We also tested the tool for the round function of the lightweight block cipher PRESENT[10][1]. The results of this analysis is are also included in Table II. The number of combinations is simply the number of combinations per gate to the power of the used number of gates in the design. The area overhead for the 4-input NAND gate based design is an increase by a factor of 4.25 compared to the unobfuscated design for the PRESENT implementation while it is 7.09 for the AES S-Box design. For both the designs the overhead is smaller if Obfusgate based on a 2-input AND gate is used. However, please note that that in the 2-input AND gate design no dummy wires are used and hence the obfuscation is considerably weaker. It is also interesting to note that due to the large number of dummy wires, the area bottleneck for the AES SBox was actually not the gate area but the the routing overhead. That is, to be able to route the design, the gate density had to be reduced during the place and routing step from 0.7 to 0.5. Please note that in our current design any net can be chosen as a dummy input. While this is good for obfuscation reasons, this obviously adds considerable overhead when designs become larger. Hence, for larger designs, the selection of dummy inputs need to be adjusted to reduce the routing overhead.

Table I
Overhead and performance for different Obfuscation Libraries for AES 8-bit SBox (421 gates) and the round function of PRESENT (383 gates)

Design	AES SBox		Present Round	
Library	NAND4	AND2	NAND4	AND2
No. of gates	865	650	626	511
Number of combinations	2^{3096}	2^{4758}	2^{2241}	2^{3741}
Area increase	7.09x	6.03x	4.25x	3.2x
Power increase	6.45x	4.59x	5.65x	2.59x
Delay increase	3.12x	2.5x	2.51x	2.12x

Besides these two case studies based on cryptographic building blocks we also tested our tool using the ITC'99 benchmark circuits. The results of this analysis are presented in Table II. For the used benchmark circuits, the area increase was between a factor of 3.2x to 7.5x with an average of 5.87x. The results from the benchmark circuits suggest that the overhead increases for larger designs. In general, the introduced overhead is quite large. While the overhead can be reduced e.g. by using Obfusgates with less inputs or not obfuscating every gate in the design, this also comes with a decreased level of obfuscation. Unfortunately, obfuscation is very hard to express as a metric as the next example will show.

[1]To be more precise, we only implemented the Substitution and Permutation Layer without the key addition.

Table II
Overhead for different ITC'99 benchmark circuits when using OBNAND4 obfuscation gates in comparison to using an unobfuscated standard cell library (Nangate Opencell Library)/

	Area in um^2 (OBNAND4)	Area in um^2 (standard cells)	Area increase
b01	324.74	77.532	4.18x
b02	177.66	48.272	3.6x
b03	1103.81	324.912	3.39x
b04	5639.94	1035.496	5.44x
b05	5319.13	895.67	5.93x
b06	316.42	101.47	3.11x
b07	3961.80	653.68	6.06x
b08	1035.34	273.27	3.7x
b09	1261.05	318.44	3.96x
b11	7056.84	890.76	7.7x

V. Obfuscating the Connectivity

Compared to other approaches, e.g. in [7], [5] the NAND4 based Obfusgate will likely result in a larger area overhead. However, compared to these approaches, our approach adds considerable amount of obfuscation in terms of Dummy Wires. More than half of the obfuscation gates in the AES SBox and PRESENT round functions were 2-input gates. Each Obfusgate that is configured as a 2-input gate has two Dummy Inputs and hence also two Dummy Wires. In the case of our implementation of the Substitution and Permutation Layer of PRESENT, 941 dummy wires and 1103 normal wires are used. This very large amount of dummy wires effectively hide the connectivity and hence the structure of the design. To illustrate how important dummy wires are, we would like to use the round function of the block cipher PRESENT as an example [10]. The present round function is depicted in Figure 5. It consists of a key addition, a Substitution Layer and a Permutation Layer. In the key addition the 64 bit state is XORed with the 64 bit round key. The Substitution Layer consists of 16 identical 4-bit SBoxes. The Permutation Layer is a permutation that can be realized simply by wiring. If an obfuscation approach is used in which each gate is replaced by an obfuscated gate without dummy wires, the logic functionality of the design would be obfuscated. That is, an attacker would not know the individual gate functions. However, the connectivity of the gates are not hidden and hence the attacker can still see the inputs and outputs of each gate. He can use this information to build a graph, in which each node is a obfuscated gate with the input and outputs being the edges of the graph. From this graph, the attacker can derive the basic structure of the design. In particular, an attacker can derive the structure of the round function as depicted in Figure 6 by looking at this connectivity graph. The attacker could not derive the functionality of one of the SBoxes. How-

ever, from the connectivity the attacker would see each SBox as a set of gates that have 4 inputs and produce 4 outputs. Since a 4-bit SBox is a 4-input function with 4 outputs the attacker would know that the design uses 16 4-bit SBoxes. Similarly, the attacker would not know that the key addition is an XOR. But he could see that each bit of the state registers goes into one or more gates that only get a single input from the round key as an additional input. Hence, the attacker would know that the key addition is a 2-input function with one input being the state register and the other the round key. The Permutation Layer is not obfuscated at all, since it does not consist of combination logic but only wiring. Hence, it would be rather simple for an attacker to derive the basic structure of the used encryption function. From this information the attacker can make educated guesses to recover more functionality and derive the used algorithm. Furthermore, the basic structure is very helpful as a starting point for other reverse-engineering techniques such as side-channel based reverse engineering or reverse-engineering based on scan chain outputs.

The heavy used of Dummy Wires in our Obfusgate prevents the attacker from deriving such a structure. In the resulting obfuscated design, each output bit of each Sbox will depend on nearly all of the 64 state register. Therefore the attacker would not know that the design is based on 4-bit SBoxes, as he would just see one large graph with lots of edges.

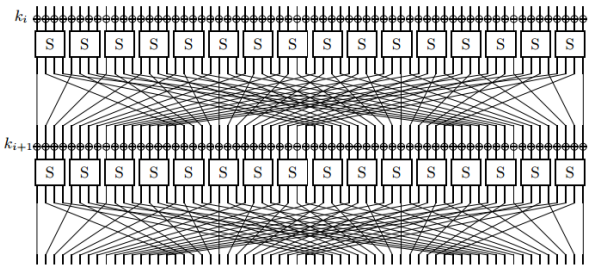


Figure 5. Figure of the PRESENT round function taken from [10]. It consists of a key addition layer, a Substitution Layer consisting of 16 identical 4-bit SBoxes and a Permutation Layer consisting of simple wiring.

VI. Discussions and Open Research Problems

In this paper we have introduced a new obfuscation tool flow that not only tries to hide the functionality of individual gates, but also their connectivity. One aim of this paper is to give a rough estimation of the area penalty of using obfuscation as a baseline for future reference. Unfortunately it is very hard to compare different designs in terms of overhead and obfuscation level. For example, one could reduce the overhead by using a obfuscation gate with less inputs and hence less

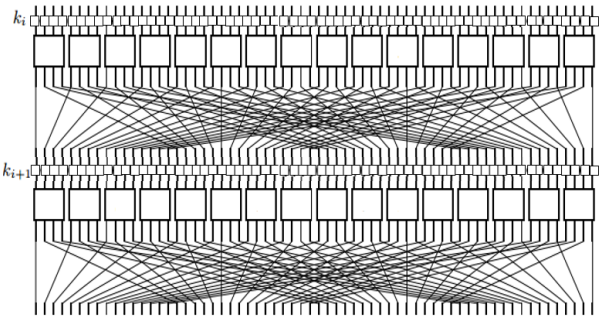


Figure 6. Illustration what an attacker could deduct from an obfuscated design that only obfuscate individual gates without introducing dummy wires. The main structure can still be derived from the connectivity graph, while only the individual functions are obfuscated.

dummy wires. However, this would reduce the amount of obfuscation of the connectivity and hence it would be easier to derive the structure of the obfuscated design. However, currently no good metric for this obfuscation is available. Simply counting the number of possible gate combinations to determine the brute-force complexity is not a suitable metric. For example, Table I we counted for each obfuscated design the possible configurations. If one wants to try to find the correct configuration using brute-force, even for the 2-input Obfusgate design this results in a ridiculous high number of 2^{3741} for the PRESENT round function. But as pointed out in the previous section, it would be fairly easy to derive the internal structure of this design if the 2-input obfusgate would be used, due to the missing dummy wires. Hence, such numbers are only misleading and do not tell much about the actual level of obfuscation. A meaningful metric of obfuscation is therefore a very difficult but also important open research problem.

Furthermore, in the current design the Dummy Wires are simply connected to any of the available nets. It seems that especially for larger designs different strategies for the connection of dummy wires need to be examined. Otherwise the area overhead would increase drastically. The area penalty of the dummy wires can be reduced greatly by taking the placing of the gates into account when choosing Dummy Inputs. But only taking Dummy Inputs that are close by would not efficiently hide the structure of the design since gates belonging to one part of the design are usually placed close to each other. Therefore one should also try to optimize the level of obfuscation by adopting a wiring mechanism that tries to optimize the connectivity graph. However, the first open research question is already how such an optimized connectivity graph should looks like. This is also of great importance when methods based on small 2-input obfuscation gates are used. In these 2-

input obfuscation libraries dummy wires can be realized by introducing dummy gates into the design. The big question is then how many dummy gates are needed and how these dummy gates should be connected to the design to optimize the obfuscation of the connectivity. It seems that this is an entire research area on its own that heavily relies on graph theory.

How to construct obfuscation gates at the circuit level that are impossible or at least extremely hard to reverse-engineer using optical methods is another interesting line of research. The current state of the art in the academic literature, dopant polarity changes and dummy contacts, add to the reverse-engineering costs but are far from impossible for a well equipped attacker. Hence, new and alternative approaches are needed. Furthermore, relatively little is known of the costs of reverse-engineering such obfuscated designs. One reason for this is that very few public information is available of what hardware reverse-engineering companies are currently capable of. In that regard, the fault-testing community might be able to provide a lot of insight, since many techniques to optically detect faults can be used to reverse-engineer such obfuscated gates. This has also been pointed out by Sugawara *et al.* in the only academic paper discussing such a reverse-engineering attack we are aware of [4]. We would like to note that these different research directions can be explored independently from each other.

To summarize, while layout-level hardware obfuscation has been studied by the industry for 20 years, the academic community has only recently started to look into this research area. Therefore, there are still many open research problems, in both circuit level obfuscation as well as how to efficiently use these gates to maximize the level of obfuscation. Ideally, these open research questions do not only attract the attention of circuit designers but also of computer scientist and reverse-engineers.

References

[1] "Innovation at risk: Intellectual property challenges and opportunities," white paper, Semiconductor Equipment and Materials International, June 2008.

[2] S. International, "Circuit camouflage technology - smi ip protection and anti-tamper technologies," White Paper Version 1.9.8j, March 2012.

[3] J. P. Baukus, L. W. Chow, and W. M. Clark, "Digital circuit with transistor geometry and channel stops providing camouflage against reverse engineering," Patent US 5 783 846, 06 21, 1998.

[4] T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, and T. Fujino, "Reversing stealthy dopant-level circuits," in *Cryptographic Hardware and Embedded Systems (CHES 2014)*, ser. LNCS. Springer, 2014, vol. 8731, pp. 112–126.

[5] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware ip protection," in *Proceedings of the 51st Annual Design Automation Conference (DAC 14)*. New York, NY, USA: ACM, 2014, pp. 153:1–153:5.

[6] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 709–720.

[7] M. Shiozaki, R. Hori, and T. Fujino, "Diffusion programmable device : The device to prevent reverse engineering," *IACR Cryptology ePrint Archive*, vol. 2014, p. 109, 2014.

[8] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Cryptographic Hardware and Embedded Systems (CHES 2013)*, ser. LNCS. Springer, 2013.

[9] N. Inc., "Nangate open cell library, version pdkv1_3_v2010_12," http://www.si2.org/openeda.si2.org/projects/nangatelib, August 2011.

[10] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 450–466.