Centrality Indicators For Efficient And Scalable Logic Masking

Brice Colombier, Lilian Bossuet
Hubert Curien Laboratory, UMR CNRS 5516, University of Lyon
42000 Saint-Étienne - France
{b.colombier,lilian.bossuet}@univ-st-etienne.fr

David Hély
LCIS, Grenoble Institute of Technology
26000 Valence - France
david.hely@lcis.grenoble-inp.fr

***Abstract*—Modifying the logic at register transfer level can help to protect a circuit against counterfeiting or illegal copying. By adding extra gates, the outputs can be controllably corrupted. Then the circuit operates correctly only if the right value is applied to the extra gates. The main challenge is to select the best position for these gates, to alter the circuit's behaviour as much as possible. However, another major point is the computational efficiency of the selection process, which should be as good as possible for integration in EDA tools. State-of-the art methods, based on fault analysis, are very demanding and cannot cope with large netlists in a reasonable runtime. We propose to use centrality indicators instead. Centrality is used to identify the most significant vertices of a graph. We show that, when used to select the nodes to modify, they lead to low correlation between original and altered outputs while being computationally efficient. We give experimental results on combinational benchmarks and compare to other previously proposed heuristics. We show that this method is the only efficient selection heuristic which is able to handle large netlists and integrate smoothly into EDA tools.**

***Keywords*-centrality; logic masking;**

I. Introduction

Integrated circuits (IC) are increasingly complex, leading to outsourcing of manufacturing to overseas foundries and adoption of a design-and-reuse paradigm. Therefore, multiple actors take part in the realisation of an IC, from Intellectual Property (IP) core providers to system integrators and foundries. The designer must fully disclose the design for it to be further used, leading to the rise of counterfeiting [1].

Intellectual property protection means were proposed to thwart this threat [2] and some are based on RTL modifications [3]. We add our voice to authors of [4] to explicitly define the terms used for gate level modifications of netlists to achieve intellectual property protection. We use the terminology recently defined in [3], and focus on logic masking.

Logic masking aims at disrupting the outputs of the IC if the wrong masking key is applied. In order to disrupt the outputs as much as possible, some internal nodes of the netlist are modified to be controllably invertible. This is achieved by inserting XOR or XNOR gates on these nodes and connecting the other input of the added gates to the masking key bit inputs. Therefore, the IC operates correctly only if the correct masking key is applied. Otherwise, the added gates act as inverters and disrupt the circuit behaviour.

978-1-5090-6762-6/17/$31.00

Two research directions can essentially be observed in logic masking. The first trend aims at making the logic masking scheme resistant to various key-recovery attacks such as hill-climbing [5] or SAT [6]. It can be achieved by inserting an extra module before the key inputs of the masking scheme. Such module can either be an AES core [7] or a "hardware point function" which output is 1 only when the correct key is applied [8], [9]. The second research direction consist in finding the best location for the extra gates so that the outputs of the netlist are maximally corrupted when the wrong key is applied. Initially, their positions were randomly selected [10]. However, this led to very small drop in correlation, hence inefficient masking. More advanced heuristics were proposed later, based on fan-in/out [11], interference graphs [12], corruptibility [13] or fault-analysis [14]. Such methods improve the masking quality with an increasing computational effort. The latest heuristic to date [14] strongly disrupts the circuit outputs but becomes impractical to compute for netlist including more than a few thousand gates nodes. However, the masking scheme insertion method is meant to be integrated into the standard EDA design flow, where performance is crucial. Moreover, designs which are worth protecting are usually large. There is therefore a strong practical requirement for heuristics that offer a better trade-off between computational complexity and masking efficiency.

In this article, we propose to use centrality indicators from graph theory to select the nodes to mask. They leverage graph algorithms for efficiency and allow to reach low correlation between the normal and masked outputs, hence efficient masking. We start with a comparison of centrality indicators. We then compare with existing heuristics in two ways. First, by measuring the masking efficiency using correlation. Second, by comparing computation times required.

This article is organised as follows. Section II describes the use case, including the attacker model. Section III presents existing heuristics to select the nodes to modify for logic masking. Section IV discusses centrality indicators, and how they could be used in the considered context. Section V gives experimental results and compares with existing heuristics. Section VI discusses implementation issues. Section VII concludes the article. Our source code is fully available online[1]

[1] https://gitlab.univ-st-etienne.fr/b.colombier/centrality-based-logic-masking

for reproducibility.

II. Preliminaries

A. Use-case

Counterfeiting can be fought by protection means based on logic modification and meant to be integrated in EDA tools. Their intended users are fabless designers who wish to protect the intellectual property of their designs by modifying them prior to sending them to the foundry for manufacturing. Therefore, the proposed modification methods should have the following properties:

- **Efficiency:** the modifications should alter the outputs as much as possible when activated, leading to the lowest possible correlation between normal and modified outputs.
- **Complexity:** the modification process should be as computationally efficient as possible, in order to integrate smoothly in the design flow of EDA tools and be capable of handling large netlists which are worth being protected.

B. Attacker model

Since the aim of these protection method is to prevent counterfeiting, the attacker model we use is the following. An attacker owns two copies of the same circuit. One is fully functional, and seen as a black box. We then use the black box model for the circuit: the attacker can choose the inputs and observe the outputs. On the other hand, the attacker also owns a locked circuit, and wants to obtain the correct key for it. This occurs typically when a customer purchases circuits from regular and black market, and hopes to activate the ones obtained on the black market with the help of the legitimate circuits.

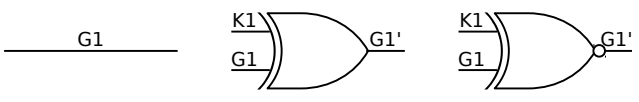
We assume the attacker cannot micro-probe the functional circuit to get the key. This requires a broader model, and is also much more costly from an attacker point of view.

III. State-of-the-art

A. Logic masking

As stated in [3], "*Logic masking consists in inserting XOR or XNOR gates in the data path of the logic circuit of a Boolean function in order to change the logic behaviour of the circuit if the wrong masking key is applied*".

First, the designer choses how many gates are to be modified in the netlist. To this end, an n-bit *masking word* is randomly generated. Next, a masking gate is inserted according to every bit of the masking word. If the bit is 0, then an XOR gate is inserted. If the bit is 1, then an XNOR gate is inserted. This is shown in Fig. 1.



(a) Original node (b) Modified node if $K_i = 0$ (c) Modified node if $K_i = 1$

Fig. 1. Modification of a node by logic masking.

If the correct masking word is applied the extra gates behave as buffers and the design operates normally. However, if a wrong masking word is applied, some of the extra gates will behave as inverters, effectively disrupting the circuit behaviour and corrupting the outputs.

B. Nodes selection heuristics

The following nodes selection heuristics select n nodes in the netlist, on which additional masking gates are to be inserted.

1) Random: The most basic way to select the nodes is random selection. This was the first proposed method, in EPIC [10]. This is fast, since no computation is required.

2) Fan-in/Fan-out cones: In 2009, authors of [11] proposed the first heuristic which improves the selection. It is based on the number of netlist nodes that are in the fan-in and fan-out cones of every other node in the netlist. The exact metric is given in Equation (1): FI and FO are the number of nodes in the fan-in and fan-out cones for every node. FI_{max} and FO_{max} are the maximum values of FI and FO observed in the netlist. w_1 and w_2 are normalisation weights which are set to 0.5. The nodes that maximise this metric are modified.

$$M_{node} = \left(\frac{w_1.FO}{FO_{max}} + \frac{w_2.FI}{FI_{max}} \right) \times \frac{FO.FI}{FI_{max}.FO_{max}} \quad (1)$$

According to this metric, the nodes with the greatest number of nodes in their fan-in/out cones are the most significant.

3) Interference graph: In [12], the random method has been improved. Initially, 10% of the masking gates are inserted randomly, to initiate the procedure. An interference graph is then built from the relative positions of the gates. The interference graph represents how the inserted gates interact with one another. For example, two masking gates placed in a row or two gates that converge to the same node are represented differently in the interference graph. Then, for every node of the netlist a metric is computed with respect to the existing masking gates, from the interference graph. The node that maximises this metric is selected, added to the interference graph and a masking gate is inserted on it. The process is then repeated again until all the masking gates are inserted.

4) Corruptibility: The authors of [12] improved their interference graph-based heuristic in [13] by adding a so-called *corruptibility* metric. This ensures that non-resolvable gates which are selected after analysing the interference graph corrupt the outputs as much as possible. Corruptibility is computed as the ratio of output patterns that differ between the normal and masked behaviour of the circuit. A node then has a high corruptibility if modifying it for logic masking changes the outputs most of the time. Computing the corruptibility requires to simulate the netlist using a dedicated tool. In [13], the authors used a fault-simulation tool, and computed corruptibility by observing one thousand input/output patterns. Such tools are usually computationally heavy.

5) Fault analysis: This is the latest proposed heuristic to date [14]. Based on fault simulation, it acts by computing the *Fault Impact* for every node of the netlist, given in Equation (2). NoP_0 is the number of patterns that can detect that the node is

stuck-at-0. NoO_0 is the number of output bits affected by this stuck-at-0 fault. NoP_1 and NoO_1 are similar but for stuck-at-1 faults.

$$\text{Fault Impact} = NoP_0.NoO_0 + NoP_1.NoO_1 \tag{2}$$

By considering both stuck-at-0 and stuck-at-1 faults, authors select the node with the greatest impact on the outputs. However, this selection heuristic is based on fault simulation, hence it remains computationally heavy. Moreover, it is recomputed every time a gate is added.

C. Netlist to graph conversion

We chose to convert the netlists to directed acyclic graphs using the same method as in [3]. Vertices are the netlist nodes and edges are logic functions connecting the nodes. A toy example is shown in Fig. 2.

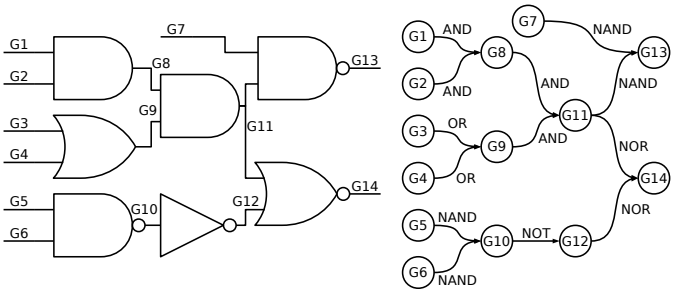


Fig. 2. A netlist and the equivalent directed acyclic graph. Netlist nodes are converted to vertices and logic functions to edges.

IV. Centrality indicators

Centrality indicators determine which vertices are the most significant in a graph. This "significant" term is very broad, and different centrality indicators perform better at identifying the "significant" vertices in different contexts. From a logic masking point of view, significant nodes are the ones for which a modification alters the most the circuit operation. Intuitively, those nodes are the ones through which a lot of information transits, from the inputs to the outputs of the circuit. Among centrality indicators, there are *local* and *global* ones. Local centrality indicators are computed according to the vertices found in the direct neighbourhood of the considered vertex. On the other hand, global indicators take the whole graph into consideration and thus they are more suited in our use case. We start by examining common global centrality indicators, before considering more sophisticated ones that are well suited to identify significant nodes in terms of "information transit".

Normalised centrality indicators: In the literature, some indicators are normalised according to the number of vertices in the network. Depending on how vertices are considered, the final indicator value can be divided by the total number of vertices. Since we are interested in the relative value of centrality for the vertices, this normalisation is not necessary. We then use only the non-normalised versions of the following indicators.

A. Closeness centrality

Closeness centrality is defined as the inverse of farness [15]. For a given vertex v, the farness is the sum of the distances from v to the other graph vertices (see Equation (3), where $d(v, y)$ stands for the distance between vertices v and y).

$$C_C(v) = \frac{1}{\sum\limits_{y:y\in V} d(v, y)} \tag{3}$$

Therefore, a vertex is significant in the sense of closeness centrality if it is the closest to all the other vertices in the graph. Practically, the vertices that have the highest closeness centrality are "in the middle" of the netlist.

It is more interesting in the case of logic masking than degree centrality because it is a global centrality indicator. Therefore, it is influenced by the graph structure.

B. Betweenness centrality

Betweenness centrality [16] of node v is given by the ratio of shortest paths between all other vertices in the graph that traverse v. This is given in Equation (4), in which σ_{st} stands for the number of shortest paths that go from s to t, and σ_{svt} stands for the total number of shortest paths that go from s to t through v.

$$C_B(v) = \sum_{s\neq t\neq v:\{s,t,v\}\in V} \frac{\sigma_{svt}}{\sigma_{st}} \tag{4}$$

Intuitively, for a netlist, betweenness centrality will be the highest for nodes that are on the shortest paths from inputs to outputs. This is interesting for logic masking, since those nodes are typically the ones for which masking will have the greatest impact on information transiting from inputs to outputs. The main drawback of this indicator, however, is that it only accounts for shortest paths. As pointed out in [17], this is quite a restrictive constraint. Indeed, it assumes that information only flows along shortest paths, which is certainly not always true.

Alternative centrality indicators based on current flow have been proposed. By assuming that information behaves in the same way as electrical current, the authors of [17], [18] account for the fact that it can split and spread in the network. This is discussed in the next subsections.

C. Current-flow betweenness centrality

Current-flow betweenness centrality [18] considers the graph as an electrical network. Vertices are converted to nodes, and the edges connecting them are replaced by unit resistors. Pairs of vertices are successively picked as current input and output. The sum of current that flows through node v for all the pairs of vertices picked gives the current-flow betweenness centrality for this node. This is shown in Equation (5), where $I_v^{(st)}$ is the current flowing through node v when s is the input and t is the output.

$$C_{CFB}(v) = \sum_{s\neq t:\{s,t\}\in V} I_v^{(st)} \tag{5}$$

This measure of centrality is more subtle than betweenness centrality. Indeed, instead of considering only shortest paths

between vertices, the current is inversely proportional to the path length. This is a more precise assumption about the way information spreads in a network.

1) Approximate current-flow betweenness centrality: For current-flow betweenness centrality, running time and space requirements rapidly become prohibitive for large graphs. An approximate version has been proposed in [17]. Instead of using all the s and t pairs in the graph as current inputs and outputs, they show that using a smaller number of randomly selected pairs leads to a good approximation. This is an interesting point in the considered use-case. Large netlists can then be analysed, by relaxing precision.

D. Current-flow closeness centrality/Information centrality

Current-flow closeness centrality was proposed in [17], and is equivalent to information centrality [19]. Instead of using distance between nodes as a measure of closeness, it proceeds similarly to current-flow betweenness centrality. First, the graph is converted to an equivalent electrical network with edges replaced by unit resistors. Afterwards, farness is the difference of potential (voltage) between the two nodes. It is the equivalent resistance between the two nodes (see Equation (6)). Thus it also accounts for paths which are not the shortest ones. All the paths between two nodes are considered, contributing to the overall equivalent resistance depending on their length.

$$C_{CFC}(v) = \frac{1}{\sum\limits_{y:y\in V} p(v) - p(y)} = \frac{1}{\sum\limits_{y:y\in V} R_{eq}(v,y)} \tag{6}$$

V. Experimental results

The centrality indicators were computed using Python *igraph* [20] and *NetworkX* [21] libraries. Our workstation embeds an Intel Core i5-4570 operating at 3.20GHz and 16GB of RAM. Experimental results were obtained with ITC'99 [22] and EPFL [23] combinational benchmarks. We restrict the size of the benchmarks from 1k to 100k gates. Note that this is the only method demonstrated on large benchmarks, up to 100k gates, when [10], [11], [14] do not exceed a few thousand gates. For large benchmarks, come centrality indicator computations ran out of memory and are not presented. Moreover, we fix a timeout limit for computation of 1h.

A. Masking efficiency evaluation

The Hamming distance criterion used in previous articles to evaluate the masking efficiency is not suited as detailed in [3]. As stated in [13], "*We need maximum corruption, and thus minimum correlation at the outputs*". Therefore, the masking efficiency E_m is evaluated by computing the quadratic mean of the Pearson's correlation coefficient (see Eq. (7)) obtained between the normal and masked mode for every output bit (see Eq. (8)).

$$r(x,y) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{7}$$

$$E_m = \sqrt{\frac{1}{n}\sum_{o\in outputs} r^2(o_{normal}, o_{masked})} \tag{8}$$

The output values are obtained by applying 10k random vectors at the key and primary inputs of the netlist. For each benchmark and area overhead, the most efficient centrality indicator is in bold face.

TABLE I
E_m VALUES FOR SELECTION HEURISTICS BASED ON CENTRALITY INDICATORS AND DIFFERENT LOGIC RESOURCES OVERHEADS

Benchmark	#gates	Centrality indicator	Logic resources overhead 1%	5%	10%
adder	∼1k	B	0.97	0.86	0.70
		C	0.98	0.94	0.91
		C-FB	0.94	**0.73**	**0.58**
		C-FC	0.96	0.95	0.93
		AC-FB	**0.91**	0.75	0.68
i2c controller	∼1k	B	0.97	0.91	0.85
		C	0.98	0.93	0.90
		C-FB	0.28	**0.19**	**0.17**
		C-FC	**0.27**	0.20	0.37
		AC-FB	0.29	0.28	0.23
sine	∼5k	B	0.23	0.26	0.22
		C	0.26	0.20	0.23
		C-FC	0.21	**0.01**	**0.01**
		AC-FB	**0.10**	0.11	**0.01**
b14_1_C	∼5k	B	0.92	0.61	0.50
		C	0.74	0.65	0.48
		AC-FB	**0.73**	**0.36**	**0.34**
b15_1_C	∼10k	B	0.82	0.57	**0.45**
		C	0.81	**0.61**	0.48
		AC-FB	**0.77**	0.64	0.85
round-robin arbiter	∼10k	B	0.96	**0.84**	**0.69**
		C	0.94	0.86	0.83
		AC-FB	**0.94**	0.88	0.83
memory controller	∼50k	B	0.98	0.94	0.88
		C	0.98	**0.91**	**0.83**
divisor	∼50k	B	0.65	0.64	0.64
		C	0.65	0.64	0.64
b18_1_C	∼100k	B	0.95	0.80	**0.63**
		C	0.95	0.80	0.65

B: betweenness
C: closeness
C-FB: current-flow betweenness
C-FC: current-flow closeness
AC-FB: approximated current-flow betweenness

The masking efficiency values E_m obtained are shown in Table I, in which three logic resources overheads are considered, 1%, 5% and 10%. The overhead is computed as the percentage of extra gates added to the design. E_m values differ greatly depending on the benchmark. For some of them, the correlation drops very fast, even at low overhead. This occurs for benchmarks in which outputs are strongly correlated, such as *sine*. On the other hand, some benchmarks make it very hard to reduce the correlation coefficient, even with a 10% overhead.

The centrality indicators differ in effectiveness. However, the ones based on current-flow are the most efficient is the majority of cases. For the largest benchmark, b18_1_C, which comprises 100k gates, the correlation drops to 0.63 for 10% area overhead. This shows that the masking is efficient, even

on very large netlists.

Increasing the overhead obviously reduces correlation since inserting more masking gates increases the masking efficiency.

Additionally, we estimated by simulation the corruptibility of the outputs when centrality indicators are used to select the nodes to modify. For all the circuits and all centrality indicators, when an incorrect key is applied, the normal and masked outputs were systematically different.

B. Computation time

Fig. 3 shows on a log-log scale how computation time varies with respect to the number of nodes in the netlist. The dark grey line is the baseline for computation time. It is the time required to only build the graph as described in Subsection III-C.

Centrality indicators are efficient to compute in general, although the centrality indicators based on current-flow require more time. However, even for a very large benchmark of 100k gates, computing betweenness and closeness centrality is possible in less than an hour on our desktop workstation. Surely this could be improved with a dedicated server. Moreover, recent research [24] shows that centrality indicators can be computed faster in a distributed manner. Computation time does not depend on the chosen overhead, since the chosen centrality indicator must be computed for all the nodes of the graph in all the cases.

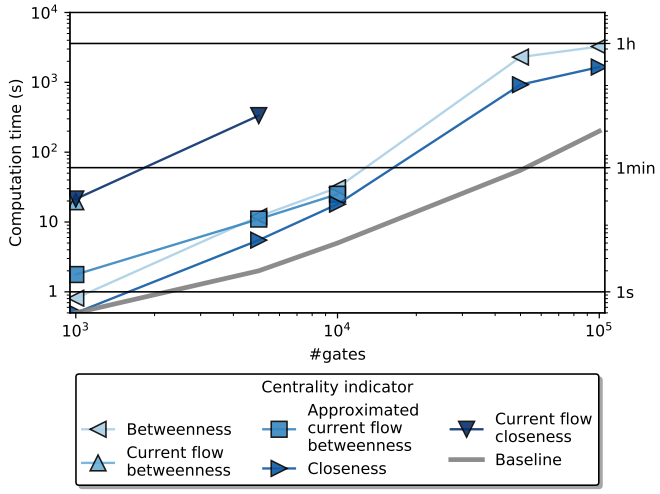


Fig. 3. Computation time required using different heuristics for selection with a 5% logic resources overhead. The baseline is random selection.

C. Comparison with existing heuristics

Fig. 4 illustrates the trade-off between correlation reduction and computation time. The baseline for computation time is random selection as it is the simplest method, thus the fastest to compute. The most efficient heuristics are closer to the origin, since they are the fastest to compute and the most efficient at reducing correlation.

Other heuristics can be broadly classified into two categories. First, fault-analysis based selection [14] can reduce correlation significantly, down to 0.2. However, this selection

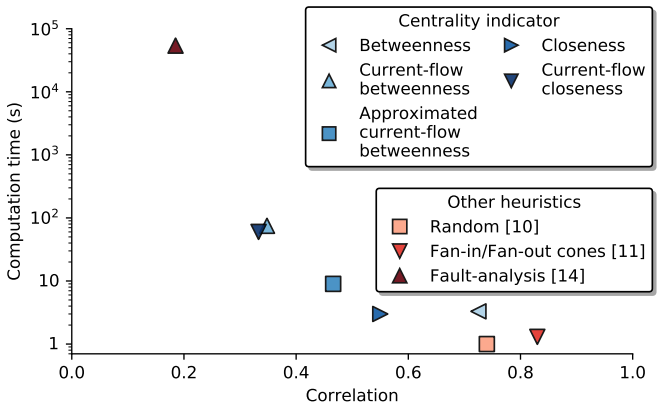


Fig. 4. Trade-off between computation time and correlation reduction. The logic resources overhead is 5-6%. The correlation and computation time values are obtained after averaging over all benchmarks, except for [14] for which only three small benchmarks from the original article are considered.

heuristic is very computationally expensive. Authors report that "*This method took two hours to encrypt the C7552 circuit*". This circuit only has 3,500 nodes. Therefore, fault-analysis based selection is highly impractical and cannot cope with large netlists, which are typically the ones worth protecting against counterfeiting. Emulation has been proposed in [25] to speed-up the process but it requires a very large FPGA for implementation since it increases the size of the original a lot. Moreover, the correlation value of 0.2 is obtained from only three combinational benchmarks, which are relatively small. Nothing shows that this low correlation would be observed on larger benchmarks such as the ones we used. On the other hand, random [10] and fan-in/fan-out cones [11] methods are rapidly computed. However, as visible in Fig. 4, the correlation remains very high. Therefore, they do not achieve efficient masking.

Overall, existing heuristics are either efficient at reducing correlation, but complex to compute, or easy to compute but inefficient at reducing correlation. In contrast, centrality indicators can reduce correlation down to 0.4 on average. Moreover, they are much more computationally efficient than fault-analysis based selection, since they run 1,000 times faster on average. Among centrality indicators, the most efficient are the ones based on current flow. They are the closest to the origin, and reduce correlation efficiently while being computationally practical. Therefore, centrality indicators offer a better trade-off than state-of-the-art heuristics between masking efficiency and computational complexity.

VI. Discussion

A. Impact on maximum operating frequency

When choosing the nodes to modify, critical paths can be excluded. This way, the impact of logic masking on the operating frequency is minimised. Masking efficiency would not be affected much since critical paths are marginal.

B. Sequential circuits

When masking sequential circuits, combinational parts must be isolated. Flip-flop inputs are converted to graph output nodes, and the flip-flop outputs to graph input nodes [4].

C. Scalability

The results we provide here for computation time are obtained on a standard desktop workstation. In order to improve performances, a dedicated server with more memory could be used to provide more computing power and analyse larger netlists. Another option to improve scalability is to compute the centrality indicator in parallel. Recent research [24] highlight the fact that current flow-based centrality indicators, which are usually the most efficient for logic masking, could be computed faster. Other heuristics based on interference graph [12] or fault analysis [14] are intrinsically sequential since they require the masking metric to be recomputed every time a node is modified.

D. Controllability and distance to inputs and outputs

For most of the modified benchmarks, inspection shows that the inserted gates are as close to the inputs as they are to the outputs. They are then approximately in the middle of the netlist. This is a good point against reverse-engineering. Indeed, if the extra gates are embedded deeper in the netlist, they are harder to uniquely identify and disable.

The distance to inputs and outputs is closely related to the controllability of the nodes. In order to make sure that the modified nodes are hard to control, one can set a threshold on their controllability. The controllability value can be computed very fast. By ensuring that the controllability of the selected nodes is high enough, the key value is much harder to reveal on the outputs of the circuit by sensitisation attack [13].

VII. Conclusion

We proposed to use centrality indicators to select the nodes modified by logic masking. On the one hand, it reduces correlation effectively and is faster to compute than state-of-the-art effective heuristics. On the other hand, compared to other computationally-efficient heuristics, it reduces correlation significantly more. Overall, it provides a better trade-off between masking efficiency and computational complexity, and is the only realistic candidate for integration in EDA tools dealing with large and complex netlists.

Acknowledgements

The work presented in this paper was realised in the frame of the SALWARE project number ANR-13-JS03-0003 supported by the French "Agence Nationale de la Recherche" and by the French "Fondation de Recherche pour l'Aéronautique et l'Espace", funding for this project was also provided by a grant from "La Région Rhône-Alpes".

References

[1] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.

[2] B. Colombier and L. Bossuet, "Survey of hardware protection of design data for integrated circuits and intellectual properties," *IET Computers & Digital Techniques*, vol. 8, no. 6, pp. 274–287, Nov. 2014.

[3] B. Colombier, L. Bossuet, and D. Hély, "From secured logic to IP protection," *Elsevier Microprocessors and Microsystems*, vol. 47, pp. 44–54, 2016.

[4] S. M. Plaza and I. L. Markov, "Protecting integrated circuits from piracy with test-aware logic locking," in *International Conference on Computer Aided Design*, San Jose, CA, USA, Nov. 2014.

[5] ——, "Solving the third-shift problem in IC piracy with test-aware logic locking," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust*, Washington, DC, USA, May 2015, pp. 137–143.

[7] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2015.

[8] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "Sarlock: SAT attack resistant logic locking," in *IEEE International Symposium on Hardware Oriented Security and Trust*, McLean, VA, USA, May 2016, pp. 236–241.

[9] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, Santa Barbara, CA, USA, Aug. 2016, pp. 127–146.

[10] J. A. Roy, F. Koushanfar, and I. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[11] R. S. Chakraborty and S. Bhunia, "HARPOON: an obfuscation-based soc design methodology for hardware protection," *IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Annual Design Automation Conference*, San Francisco CA, USA, Jun. 2012, pp. 83–89.

[13] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *ACM Conference on Computer & communications security*, Berlin, Germany, Nov. 2013, pp. 709–720.

[14] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2015.

[15] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, no. 4, pp. 581–603, 1966.

[16] J. M. Anthonisse, "The rush in a directed graph," *Mathematische Besliskunde*, no. BN 9/71, pp. 1–10, 1971.

[17] U. Brandes and D. Fleischer, "Centrality measures based on current flow," in *Annual Symposium on Theoretical Aspects of Computer Science*, vol. 3404, Stuttgart, Germany, Feb. 2005, pp. 533–544.

[18] M. E. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39–54, 2005.

[19] K. Stephenson and M. Zelen, "Rethinking centrality: Methods and examples," *Social Networks*, vol. 11, no. 1, pp. 1–37, 1989.

[20] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal Complex Systems*, vol. 1695, no. 5, pp. 1–9, 2006.

[21] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Python in Science Conference*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[22] S. Davidson, "ITC'99 benchmark circuits - preliminary results," in *IEEE International Test Conference*, Atlantic City, NJ, USA, Sep. 1999, p. 1125.

[23] L. Amarú, P.-E. Gaillardon, and G. D. Micheli, "The EPFL combinational benchmark suite," in *International Workshop on Logic & Synthesis*, Mountain View, CA, USA, Jun. 2015.

[24] A. Lulli, L. Ricci, E. Carlini, and P. Dazzi, "Distributed current flow betweenness centrality," in *International Conference on Self-Adaptive and Self-Organizing Systems*, Cambridge, MA, USA, Sep. 2015, pp. 71–80.

[25] S. Gören, C. C. Gürsoy, and A. Yildiz, "Speeding up logic locking via fault emulation and dynamic multiple fault injection," *Journal of Electronic Testing*, vol. 31, no. 5-6, pp. 525–536, 2015.