Idea: Benchmarking Indistinguishability Obfuscation – A candidate implementation

Sebastian Banescu, Martín Ochoa, Nils Kunze, and Alexander Pretschner

Technische Universität München, Germany
{banescu, ochoa, nils.kunze, pretschn}@cs.tum.edu

Abstract. We present the results of preliminary experiments implementing the Candidate Indistinguishability Obfuscation algorithm recently proposed by Garg et al. [1]. We show how different parameters of the input circuits impact the performance and the size of the obfuscated programs. On the negative side, our benchmarks show that for the time being the algorithm is far away from being practical. On the positive side, there is still much room for improvement in our implementation. We discuss bottlenecks encountered and optimization possibilities. In order to foster further improvements by the community, we make our implementation public.

1 Introduction

Obfuscation of software, intended as a transformation of a program such that it is difficult for adversaries to understand details of its logic or internal variables, is of an increasing practical relevance [2]. Typically obfuscation is associated with 'security by obscurity', because of a lack of formal guarantees on the security of commonly used obfuscation operators [3]. On the theoretical side, it has been shown by Barak et al. [4] that it is impossible to construct an obfuscator such that from the obfuscated version of a program implementing a function f, an adversary can only learn the inputs and outputs to f exclusively.

Recently, Garg et al. [1] proposed a promising approach that offers formal security guarantees for *indistinguishability obfuscation*, a particular obfuscation notion that guarantees that the obfuscations of two programs implementing the same functionality are indistinguishable. As a basis for their proof, the authors show that a successful attack to their construction is also an solution to the multilinear jigsaw problem, which is believed to be computationally hard. The authors conjecture that this construction provides the expected security for the obfuscation of most programs.

Although the proposers of indistinguishability obfuscation acknowledge that their construction is not practical as of today [5], concrete details have so far not been published. The motivation of our work is thus to better understand how far is the candidate construction from real applications. To do so, we prototypically implemented the algorithm described in [1] and benchmarked its space and time performance depending on various parameters.

Our contributions can be summarized as follows: a) to our knowledge, we provide the first open source implementation of the candidate indistinguishability obfuscation candidate [1], so that the community can gradually improve on it, b) we provide reproduceable performance benchmarks, which give an upper bound on the necessary time and space for running/storing obfuscated programs and c) we discuss potential areas for improvement based on our experiments.

The paper is organized as follows: In Section 2 we give an overview of the candidate construction. We then present an overview of our implementation in Section 3 and our benchmarking results in Section 4. We conclude by summarizing our results and giving an overview of ongoing and future work in Section 5.

2 Preliminaries

This section presents the candidate indistinguishability obfuscation construction developed by Garg *et al.* [1] applied to *boolean circuits* in $\mathbf{NC}^1$ [6], preceded by the concepts needed to understand this construction.

A boolean circuit is a directed acyclic graph, where nodes are represented by conjunction, disjunction and/or negation gates with maximum 2 inputs (fan-in-2), which process only boolean values. The *size* of a circuit is equal to the total number of gates in that circuit. The *depth* of a circuit is the length of the longest path from input to output gate, in the circuit.

A uniform probabilistic polynomial-time Turing (PPT) machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if: (1) it preserves the input-output behavior of the unobfuscated circuit and (2) given two circuits $C_1, C_2 \in \mathcal{C}_\lambda$ and their obfuscated counterparts $i\mathcal{O}(\lambda, C_1), i\mathcal{O}(\lambda, C_2)$, a PPT adversary will not be able to distinguish which obfuscated circuit originates from which original circuit with significant probability (the advantage of the adversary is bounded by a negligible function of the security parameter λ).

Even though an $i\mathcal{O}$ applies to boolean circuits, internally it transforms all circuits into *linear branching programs* on which it operates. This transformation is made possible by Barrington's theorem [7], which states that any fan-in-2, depth-d boolean circuit can be transformed into an oblivious linear branching program of length at most 4^d, that computes the same function as the circuit.

Definition 1. *(Oblivious Linear Branching Program [1]) Let $A_0, A_1 \in \{0,1\}^{5\times 5}$ be two distinct arbitrarily chosen permutation matrices. An (A_0, A_1) oblivious branching program of length n for circuits with ℓ-bit inputs is represented by a sequence of instructions $BP = ((inp(i), A_{i,0}, A_{i,1}))_{i=1}^{n}$, where $A_{i,b} \in \{0,1\}^{5\times 5}$, and $inp : \{1, n\} \to \{1, \ell\}$ is a mapping from branching program instruction index to circuit input bit index. The function computed by the branching program is*

$$f_{BP,A_0,A_1}(x) = \begin{cases} 0 & \text{if } \Pi_{i=1}^{n} A_{i,x_{inp(i)}} = A_0 \\ 1 & \text{if } \Pi_{i=1}^{n} A_{i,x_{inp(i)}} = A_1 \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

The family of circuits $\mathcal{C}_\lambda$ is characterized by ℓ inputs, λ gates, $O(log\lambda)$ depth and one output. $\mathcal{C}_\lambda$ has a corresponding polynomial-sized universal circuit, which is a function $U_\lambda : \{0,1\}^{f(\lambda)} \times \{0,1\}^\ell \to \{0,1\}$, where $f(\lambda)$ is some function of λ. U_λ can encode all circuits in $\mathcal{C}_\lambda$, i.e. $\forall C \in \mathcal{C}_\lambda, \forall z \in \{0,1\}^\ell, \exists C_b \in \{0,1\}^{f(\lambda)} : U_\lambda(C_b, z) = C(z)$. It is important to note that the input of U_λ is a $f(\lambda) + \ell$ bit string and that by fixing any $f(\lambda)$ bits, one obtains a circuit in $\mathcal{C}_\lambda$.

Universal circuits are part of the candidate $i\mathcal{O}$ construction, because they enable running Kilian's protocol [8], which allows two parties (V and E), to evaluate any $\mathbf{NC}^1$ circuit (e.g. U_λ) on their joint input $\mathcal{X} = (x|y)$, without disclosing their inputs to each other, where x, y are the inputs of V, respectively E. This is achieved by transforming the circuit into a branching program $BP = ((inp(i), A_{i,0}, A_{i,1}))_{i=1}^n$ by applying Barrington's theorem [7]. Subsequently V chooses n random invertible matrices $\{R_i\}_{i=1}^n$ over $\mathcal{Z}_p$, computes their inverses and creates a new *randomized branching program* $RBP = ((inp(i), \tilde{A}_{i,0}, \tilde{A}_{i,1}))_{i=1}^n$, where $\tilde{A}_{i,b} = R_{i-1} A_{i,b} R_i^{-1}$ for all $i \in \{1,n\}, b \in \{0,1\}$ and $R_0 = R_n$. It can be shown that RBP and BP compute the same function. Subsequently, V sends E only the matrices corresponding to her part of the input $\{\tilde{A}_{i,b} : i \in \{1,n\}, inp(i) < |x|\}$ and E only gets the matrices corresponding to one specific input via oblivious transfer. E can now compute the result of RBP without finding out V's input. Kilian's protocol is related to the notion of program obfuscation, if we think of V as a software vendor who wants to hide (obfuscate) a program that is going to be distributed to end-users (E). However, Kilian's protocol [8] is modified in [1], by sending all matrices corresponding to any input of E, which allows E to run the RBP with more that one input. This modified version is vulnerable to *partial evaluation attacks*, *mixed input attacks* and also non-multilinear attacks, which extract information about the secret input of V.

To prevent partial evaluation attacks Garg *et al.* [1] transform the 5×5 matrices of BP into higher order matrices, having dimension $2m + 5$, where $m = 2n+5$ and n is the length of BP. Subsequently, they add 2 *bookend* vectors of size $2m + 5$ in order to neutralize the multiplication with the random entries in the higher order matrices. To prevent mixed input attacks a multiplicative bundling technique is used, which leads to an *encoded* output of BP. To decode the output of the BP an additional branching program of equal length with BP, that computes the constant 1 function is generated and the same multiplicative bundling technique is applied to it. Subtracting the results of the two branching programs executed on the same inputs, will decode the output of BP. To prevent non-multilinear attacks, the candidate construction of Garg *et al.* [1] employs the *multilinear jigsaw puzzle* (MJP).

An overview of MJP is illustrated in Figure 1 and consists of two entities, i.e. the *Jigsaw Generator* (JGen) and the *Jigsaw Verifier* (JVer). The JGen is part of the *circuit obfuscator*. It takes as input a security parameter (λ), a universal circuit (U_λ) and the number of input bits (ℓ) of any circuit simulated by U_λ. JGen first applies Barrington's theorem [7] to transform U_λ into a universal branching program UBP of length n. Subsequently, the *Instance Generator* takes λ and the multilinearity parameter ($k = n+2$) as inputs and outputs a prime number p and

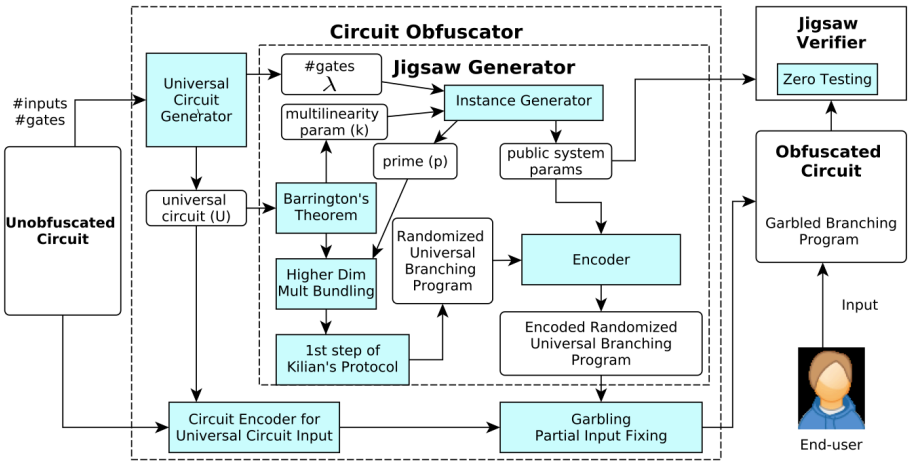

Fig. 1. Overview of the candidate construction for indistinguishability obfuscation

a set of public system parameters (including a large random prime q and a small random polynomial $g \in \mathcal{Z}[X]/(X^m + 1)$). Afterwards, UBP is transformed into a randomized branching program by: (1) transforming the branching program matrices into higher order matrices, (2) applying multiplicative bundling and (3) the first step of Kilian's protocol. The output of JGen is a set of public system parameters and the randomized universal branching program ($\widehat{\mathcal{RND}}(UBP_\lambda)$) with all matrices encoded by the *Encoder* component.

The output of JGen can be used to obfuscate a circuit $C \in \mathcal{C}_\lambda$ by fixing a part of the inputs (garbling) of $\widehat{\mathcal{RND}}(UBP_\lambda)$ such that it encodes C for all $z \in \{0,1\}^\ell$. Garbling is done by discarding the matrices of $\widehat{\mathcal{RND}}(UBP_\lambda)$ which correspond to values not chosen for the fixed input bits. The result of this step is $i\mathcal{O}(\lambda, C)$, the candidate of Garg *et al.* [1]. It is sent to an untrusted party which evaluates it by fixing the rest of its inputs and providing it as input to the JVer. The JVer outputs 1 if the evaluation of $i\mathcal{O}(\lambda, C)$ is successful and 0, otherwise.

3 Implementation

Our proof-of-concept implementation was done in Python, leveraging the SAGE computer algebra system and can be downloaded from the Internet[1]. It consists of the following modules, corresponding to the light blue rectangles from Figure 1: (1) building blocks for universal circuit creation, (2) Barrington's theorem for transforming boolean circuits to branching programs, (3) transformation from branching program matrices into higher order matrices and applying multiplicative bundling (4) 1st step of Kilian's protocol for creating randomized branching

[1] `https://github.com/tum-i22/indistinguishability-obfuscation`

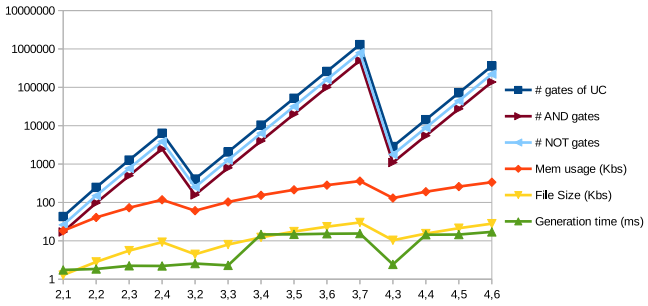

Fig. 2. Generation of UCs (X-axis: no. inputs (ℓ), no. gates of input circuit (λ))

programs from branching programs, (5) instance generator for MJP, (6) encoder for MJP, (7) circuit encoder into input for universal circuit, (8) partial input fixer for random branching programs, and (9) zero testing of jigsaw verifier.

Technical challenges faced Although commonly used in the literature, we could not find a readily available implementation of Universal Circuits (UC) that was easily adaptable to our setting. Therefore we decided to implement our own UC component, following the less performant algorithm of [9]. For the sake of performance, this component can be improved by following for instance the more performant (but more complex) algorithm suggested in [9] or [10].

Challenges interpreting [1] We also faced some challenges while interpreting the candidate construction description, in particular their suggested encoding function. For instance it was difficult to come up with concrete values for some parameters, since the relation between them is given using the big O notation. On the other hand, the Encoder function requires to reduce an element $a \in \mathcal{Z}_p$ modulo a polynomial g of degree ≥ 1. We could not think of a better canonical representative for this reduction than a itself, which makes us believe that either the modulo reduction is redundant or the authors had another canonical representative in mind (a polynomial) which is unclear how to compute.

Summary of current status Currently, our implementation can perform most steps of the candidate construction, with the exception of the zero test. We believe this is a result of an incorrect choice of the canonical representative of a modulo g or/and of the concrete parameters as discussed above. We have raised these issues in popular mathematics and cryptography forums and contacted the authors for clarification with no success at the moment of elaborating this document. However, note from Figure 1 that the improper functioning of the zero test does not affect the results of benchmarking the *Circuit Obfuscator* presented in the next section, because the it is part of the *Jigsaw Verifier*.

4 Benchmarking

We executed our experiments on a virtual machine (VM) with 4 cores and 64 GB of memory. The first experiment aims to investigate the resources required

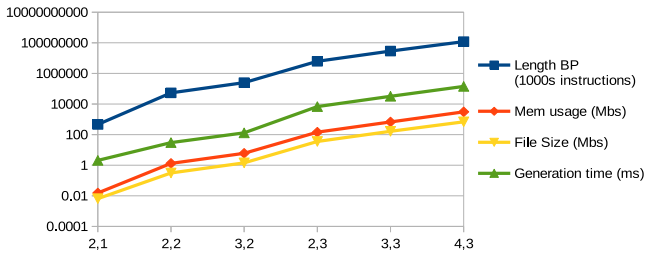

Fig. 3. Generation of BPs (X-axis: no. inputs (ℓ), no. gates of input circuit (λ))

to obfuscate a circuit consisting only of AND gates as a function of its number of inputs and gates. As illustrated in Figure 1 the first step of obfuscation consists of generating the UC, corresponding to the first step of our experiment. The number of circuit inputs were varied between 2 and 4, while the number of gates between 1 and 10. The recorded outputs are shown in Figure 2 and consist of the: number of gates, memory usage, output file and generation time needed for the UC. Observe that increasing the number of inputs causes a linear increase in each measured output of the experiment, while increasing the number of gates causes an exponential increase. The memory usage is around one order of magnitude higher than the file size due to the compression algorithm we use to store UCs.

The second step of our experiment consisted of transforming the previously generated UCs into branching programs (BPs) using our implementation of Barrington's theorem [7]. However, it was infeasible to transform all the previously generated UCs because of the fast polynomial increase in memory usage and file size, illustrated in Figure 3. We estimated the size of generating a BP for a UC which encodes a circuit by applying following recursive formula (corresponding to our implementation), to the output gate of a UC:

$$l(\mathsf{gate}) = \begin{cases} 1 & \text{if type}(\mathsf{gate}) = \mathsf{Input} \\ l(\mathsf{gate.input}) & \text{if type}(\mathsf{gate}) = \mathsf{NOT} \\ 2l(\mathsf{gate.input1}) + 2l(\mathsf{gate.input2}) & \text{if type}(\mathsf{gate}) = \mathsf{AND} \end{cases}$$

The estimated memory usage of a universal BP which encodes 4 inputs and 6 gates, corresponding to the largest UC we show in Figure 2, is over 4.47 Peta Bytes, which is infeasible to generate on our VM.

The third step of our experiment was to transform the BPs generated previously into randomized branching programs (RBPs) by transforming the BP matrices into higher order matrices, applying multiplicative bundling and the first step of Kilian's protocol [8]. The results of this experiment are shown in Figure 4. Additionally to the number of inputs and gates, in this experiment we also have the matrix dimension increase (m) and the choice of the prime (p) corresponding to $\mathcal{Z}_p$ in which Kilian's protocol operates. The choice of m influences both the generation time and the file size polynomially. Observe that the memory usage remains constant for different values of m. This is due to compatibility issues between SAGE and our memory profiler. However, we observer

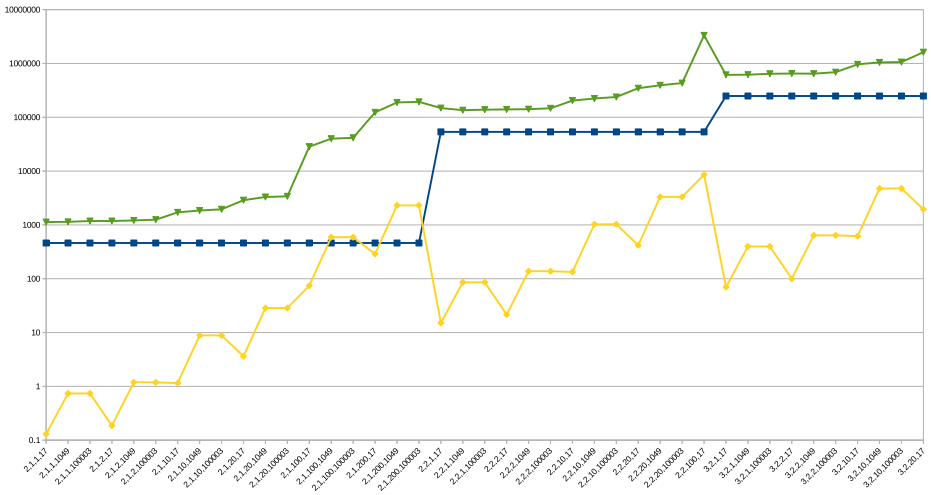

Fig. 4. Generation of RBPs (X-axis: no. inputs (ℓ), no. gates of input circuit (λ), matrix dimension (m), prime number (p)). Legend is the same as Figure 3.

that the actual memory usage is still one order of magnitude higher than the file size. p influences the generation time linearly, however, the memory usage and file size are affected only if the data type width of p grows. Note that, the memory usage is not shown in Figure 4 since it could not be measured reliably due to technical limitations of our memory profiler. We estimate that the memory usage is approximately one order of magnitude higher than the file size.

5 Conclusions and Future Work

In this paper we have presented a non-trivial upper bound on the size and performance of the obfuscated versions of small circuits. To give an idea about the practicality of this construction, consider a 2-bit multiplication circuit. It requires 4 inputs and between 1 and 8 AND gates for each of its 4 output bits. An obfuscation would be generated in about 10^{27} years on a 2,6 GHz CPU and would require 20 Zetta Bytes of memory for $m = 1$ and $p = 1049$. Executing this circuit on the same CPU would take 1.3×10^8 years. This clearly indicates that for the time being the candidate construction is highly unpractical.

However, this upper bound can still be tightened (perhaps even dramatically) by improving upon our preliminary implementation. In particular, there exist better algorithms for the generation of universal circuits, which directly affect the size of the obfuscation [9, 10]. There is an inherent limitation for this improvement due to the fact that the output of gates in UCs are reused by other gates, which causes duplication of matrices in BPs when using Barrington's theorem [7]. Therefore, one improvement is to avoid using Barrington's theorem as suggested by Ananth et al. [11]. On the other hand, we have only implemented the construction for $\mathbf{NC}^1$ circuits: the candidate construction in-

cludes an extension to cope with bigger circuit classes, that includes the use of fully homomorphic encryption. To this date, there exists no practical implementations of fully homomorphic encryption, although progress has been made since the original algorithm was proposed [12].

As research advances towards practical fully homomorphic encryption, we expect our initial and open implementation of the candidate indistinguishability obfuscation algorithm to foster improvements by the community. Being open, our implementation is amenable to adaptations to new algorithms based on the MJP complexity assumption.

At the moment of submission of this manuscript, we are working to make our implementation fully functional. Avenues for future work include: (1) improving the UC generation procedure according to [9, 10], (2) engineering more efficient representations for the matrices and polynomials in memory and disk, (3) improving our optimization technique to reduce obfuscated circuit generation time, (4) experimenting with various compression techniques and (5) implementing the technique of Ananth et al. [11], to avoid Barrington's theorem.

References

1. S. Garg, C. Gentry, S. Halevi, M. Raykova, A Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. of the 54th Annual Symp. on Foundations of Computer Science*, pages 40–49, 2013.
2. W. Zhou, Z. Wang, Y. Zhou, and X. Jiang. Divilar: Diversifying intermediate language for anti-repackaging on android platform. In *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy*, pages 199–210. ACM, 2014.
3. C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand, 1997.
4. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. In *CRYPTO 2001*, pages 1–18. Springer, 2001.
5. C. Edwards. Researchers probe security through obscurity. *Communications of the ACM*, 57(8):11–13, 2014.
6. S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
7. D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *Proc. of the 18th Annual ACM Symp. on Theory of Computing*, STOC '86, pages 1–5, New York, NY, USA, 1986. ACM.
8. J. Kilian. Founding crytpography on oblivious transfer. In *Proc. of the 20th Annual ACM Symp. on Theory of Computing*, pages 20–31. ACM, 1988.
9. T. Schneider. Practical secure function evaluation. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2008.
10. L. G. Valiant. Universal circuits (preliminary report). In *Proc. of the 8th Annual ACM Symp. on Theory of Computing*, pages 196–203. ACM, 1976.
11. P. Ananth, D. Gupta, Y. Ishai, and A. Sahai. Optimizing obfuscation: Avoiding barrington's theorem. *IACR Cryptology ePrint Archive*, 2014:222, 2014.
12. M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proc. of the 3rd ACM Workshop on Cloud Computing Security*, pages 113–124. ACM, 2011.