Recent Results on Indistinguishability Obfuscation

Prastudy Fauzi (supervised by Dominique Unruh)

University of Tartu, Estonia
prastudy.fauzi@ut.ee

Abstract. In this research, the student will explain a candidate construction for indistinguishability obfuscation, based on a recent paper by Garg et al. (FOCS 2013). This is done by first describing the main building blocks, then how these are used to build indistinguishability obfuscation for NC^1, and expanded to $P/poly$. Security and alternative constructions are very briefly discussed.

Keywords: indistinguishability obfuscation, NC^1, polynomial size circuits

1 Introduction

Obfuscation is an important concept in computer programs, as it enables a program to work as intended, but makes it unreadable in its publicly available form, making reverse engineering difficult. This is important if a company needs to hide some implementation ideas of a product, while still giving the product to its customers. However, it has been known that for general programs, obfuscation is difficult to do.

This leads to an alternative notion called indistinguishability obfuscation, where any two equivalent circuits must have computationally indistinguishable obfuscations. We will do a summary of a recent paper by [GGH$^+$13b] that gives a candidate construction for indistinguishability obfuscation.

Scope. We will explain the construction in a relatively high level, and refer to the original paper for details. Also, while the original paper explains both indistinguishability obfuscation and functional encryption, we will focus only on indistinguishability obfuscation. We will also leave out discussions on the various uses of indistinguishability obfuscation and functional encryption.

2 Preliminaries

We will give an overview of the basic cryptographic concepts used in the later sections.

2.1 Sets

For a positive integer k, let $[k] = \{1, 2, \cdots, k\}$ be the set of first k positive integers.

2.2 Circuits

A family of boolean circuits is *uniform* if a circuit in the family can be generated by only knowing the input length. The *depth* of a circuit is the maximum distance from an input to an output. Two circuits C_1, C_2 are equivalent if for all inputs x, it holds that $C_1(x) = C_2(x)$.

For a circuit family $\{C_\lambda\}$, let U_λ be the universal circuit such that $U_\lambda(C, m) = C(m)$ for all $C \in \{C_\lambda\}$ and inputs m.

2.3 Circuit Complexity

For a positive integer i, NC^i is the class of decision problems (i.e. the output is either 0 or 1) that can be decided by a uniform Boolean circuit with gates of fan-in 2 (i.e. at most 2 inputs for each gate) and depth $O(\log n^i)$. A decision problem is in NC if it is in NC^i for some positive integer i. The set of decision problems decidable using polynomial-sized circuits is known as $P/poly$.

Recall that P is the set of languages decidable in polynomial-time. It is easy to see that $NC \subseteq P \subseteq P/poly$. Moreover, Barrington's theorem [Bar86] states that NC^1 is equivalent to the class of branching programs.

2.4 Obfuscation

An *obfuscation* of a program $Prog$ (typically, source code or executable) is the act of obscuring the implementation details of a program as $\mathcal{O}(Prog)$ such that it will be in some sense unintelligible (typically, difficult to understand for humans reading the source code or decompiled executable), but works as intended (i.e. $\mathcal{O}(Prog)$ has the same output of the non-obfuscated code $Prog$).

Impossibility Result Barak et al. [BGI+ 12] proved that black box obfuscation, which is the most natural formulation of program obfuscation, is impossible to achieve for programs in general. Specifically, there exists low complexity unobfuscatable functions.

2.5 Indistinguishability Obfuscators

In response to finding this impossibility result, Barak et al. [BGI+ 12] came up with an alternative notion that captures a property we want in program obfuscation: For a class of circuits $\mathcal{C}$, a function $i\mathcal{O}$ is called an *indistinguishability obfuscator* if for any two equivalent circuits $C_1, C_2 \in \mathcal{C}$ we have that the distribution of $i\mathcal{O}(C_1), i\mathcal{O}(C_2)$ are computationally indistinguishable.

2.6 Multilinear Maps

Let $G_1, G_2, \cdots, G_k, G_T$ be cyclic groups of the same order p, Then a k-multilinear map $e : G_1 \times G_2 \times \cdots \times G_k \to G_T$ is a function with the following properties:

1. For elements $g_i \in G_i$, $j \in [k]$ and integer a we have that
$$e(g_1, g_2, \cdots a \cdot g_j \cdots, g_k) = a \cdot e(g_1, g_2, \cdots g_k).$$
2. If the elements $g_i \in G_i$ are all generators of their groups G_i, then $e(g_1, g_2, \cdots, g_k)$ is a generator of G_T.

Essentially, this means that a multilinear map must be linear with respect to each parameter $g_i \in G_i$, and the map itself is non-degenerate. Note that the G_i-s need not be the same. If $G_1 = G_2 = \cdots = G_k = G$, this is called the symmetric case. It has the property that for elements $g_i \in G$ we have $e(g_1, g_2, \cdots g_k) = e(g_{\tau(1)}, g_{\tau(2)}, \cdots, g_{\tau(k)})$ for any permutation τ of $[k]$.

2.7 Public Key Encryption

A public key encryption (PKE) scheme consists of three elements: key generation, encryption, and decryption.

- Key generation is the process of generating a public key and secret key pair for encryption and decryption. It requires a security parameter λ, typically the size of the resulting public key.
- Encryption is the function that maps a plaintext into a ciphertext, using a public key. The domain of this encryption function is called plaintext space.
- Decryption is the function that maps a ciphertext back into plaintext, using a secret key.

A PKE scheme with a key generation algorithm KG, encryption algorithm E and decryption algorithm D can then be written as (KG, E, D). Some well-known PKE schemes include RSA, ElGamal and Paillier. A PKE scheme is called *probabilistic* if it uses randomness such that two encryptions of the same message need not be the same ciphertext value.

2.8 Homomorphic Encryption

Let the plaintext space P have "addition" operator $+$, and "multiplication" operator $\times$, and let the ciphertext space C have "addition" operator $\oplus$, and "multiplication" operator $\otimes$. Let $E : P \to C$ be a probabilistic encryption scheme, and $D : C \to P$ the corresponding decryption scheme.
A public key cryptosystem (KG, E, D) is *homomorphic* under addition if for all $a, b \in P$

$$D(E(a) \oplus E(b)) = a + b$$

and it is called homomorphic under multiplication if for all $a, b \in P$

$$D(E(a) \otimes E(b)) = a \times b.$$

An encryption scheme is *fully homomorphic* if it is both additively and multiplicatively homomorphic, and hence can evaluate any arithmetic circuit. An encryption scheme is *leveled fully homomorphic* [Gen09] if it can evaluate arithmetic circuits only up to depth L.

3 Building Blocks

We describe the three main concepts used in the construction of indistinguishability obfuscation of [GGH$^+$13b].

3.1 Branching Programs

A branching program [Hol14] is an acyclic directed graph which has a start node and two end nodes, usually labeled 0 and 1. Every vertex except the start node has non-zero in-degree, while every node except the end nodes have out-degree two.

The width of a branching program is the maximum number of nodes of the same level, while the depth is the length of the path from the start node to the end node. In the figure 1, the branching program has width 3 and depth 6, and computes a function $f : \{0,1\}^4 \to \{0,1\}$, where for example $f(0000) = 0$ and $f(0010) = 1$.

Branching programs of n variables and width w are typically written as a set of n pairs of permutation $(w \times w)$ matrices, where each pair of matrices corresponds to the relationship between one level of the graph to another, on input 0 or input 1. Barrington's theorem specifies how branching programs are related to the class NC^1.

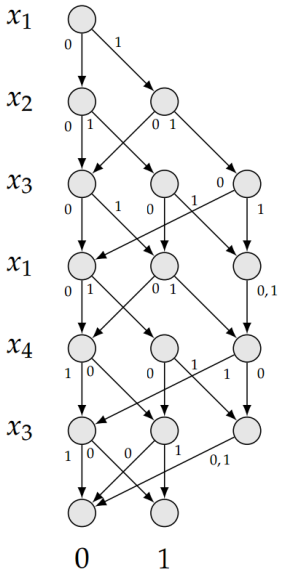

Fig. 1. An example of a branching program from [Hol14].

Theorem 1. *(Barrington's theorem [Bar86]) Let C be a circuit of depth d and 1 bit of output. There exists a branching program of width 5 and depth 4^d computing the same function as C.*

For $d = O(\log n)$, 4^d is polynomial in n, hence the following corollary is straightforward.

Corollary 1. *Any circuit in NC^1 can be computed by a branching program of width 5 and polynomial depth.*

Given any Boolean circuit $C : \{0,1\}^l \to \{0,1\}$ with fan-in 2 and depth d, Barrington's theorem also showed how to transform it into a permutation branching program $BP = \{inp(i), A_{i,0}, A_{i,1}\}$ of width 5 and length $n \leq 4^d$. Here the $A_{i,b}$ are 5×5 permutation matrices, $inp(i)$ denotes what input bit is examined in step i, and the evaluation of BP first computes $P = \prod_{i=1}^{n} A_{i,b_{inp(i)}}$, and outputs 1 if and only if $P = I$, the 5×5 identity matrix.

For a circuit C and bit string m, let $UBP(C, \cdot)$ be the universal branching program obtained from Barrington's theorem applied to a universal circuit $U_\lambda(C, \cdot)$. For a specific input circuit c, let (I_C, σ_c) be the partial assignment that fixes the bits of c in the input of UBP.

3.2 Killian's Protocol

In Killian's protocol [Kil88], two parties jointly evaluate an NC^1 circuit but keeping their inputs private. This can be adapted to randomizing a branching program. Here the branching program $BP = \{inp(i), A_{i,0}, A_{i,1}\}$ is transformed into the randomized branching program $RBP = \{inp(i), \tilde{A}_{i,0}, \tilde{A}_{i,1}\}$ by the transformation $\tilde{A}_{i,b} = R_{i-1} A_{i,b} R_i^{-1}$ for invertible matrices R_i over $\mathbb{Z}_p$. We denote this as $RBP \leftarrow \mathcal{RND}_p(BP)$. In this setting, one party would be the obfuscator that creates RBP, while the other party would be the evaluator of the randomized branching program.

Let $J \subset [l]$ and consider a partial assignment of input bits $\sigma : J \to \{0,1\}$. Denote a garbled program relative to the partial assignment (J, σ) as $GARBLE(\mathcal{RND}_p(BP), (J, \sigma))$. This removes all matrices $D_{i,b}$ that don't match the partial assignment σ. If the original program is computing the function F, then we denote as $F|_\sigma$ the function that is computed by this garbled program.

Functionally Equivalent Assignments. For a function $F : \{0,1\}^l \to \{0,1\}$, two partial assignments $(J, \sigma_0), (J, \sigma_1)$ over the input variables are *functionally equivalent relative to F* if $F|_{\sigma_0} = F|_{\sigma_1}$.

3.3 Multilinear Jigsaw Puzzles

Jigsaw Specifier. A *Jigsaw Specifier* is a tuple (k, l, A) where $k, l \in \mathbb{Z}^+$ and A is a probabilistic circuit that, for input prime p, outputs $A(p) = (p, (S_1, a_1), (S_2, a_2), \cdots, (S_l, a_l))$ where $a_i \in \mathbb{Z}_p$ and $S_i \subseteq [k]$.

Multilinear Form. A *Multilinear Form* is a tuple $\mathcal{F} = (k, l, \Pi, F)$ where $k, l \in \mathbb{Z}^+$ and Π is a circuit with l input wires, consisting of binary addition $\oplus$ and multiplication $\otimes$ gates, along with unary negation $\ominus$ and empty (zero out-degree) $\square$ gates. Moreover, F is an assignment of a set $I \subseteq [k]$ to every wire in Π, with the following rules:

1. For every $\oplus$ or $\ominus$ gate, assign the same set I to all input and output wires of that gate.
2. For every $\otimes$ gate, assign two disjoint sets $I_1, I_2 \subseteq [k]$ to the input wires, and the union $I_1 \cup I_2$ to the output wire.
3. Assign $[k]$ to the final output wire.

Multilinear Forms can be evaluated on the output of a Jigsaw Verifier, which is defined later in this section.

Multilinear Jigsaw Puzzle. A *Multilinear Jigsaw Puzzle* scheme consists of two probabilistic polynomial time (PPT) algorithms $(JGen, JVer)$ described as follows:

- Jigsaw Generator $JGen = (InstGen, Encode)$ is given a Jigsaw Specifier (k, l, A) and does the following:
  1. Generate the instance $(p, prms, s) \leftarrow InstGen(1^\lambda, 1^k)$, where λ is the security parameter,
  2. Run the Jigsaw Specifier algorithm A on input p to get
  $$A(p) = (p, (S_1, a_1), (S_2, a_2), \cdots, (S_l, a_l)).$$
  3. Encode all plaintext elements to get $(S_i, u_i) = Encode(prms, s, S_i, a_i)$ for all $i \in [l]$.

  If we denote $X = (p, (S_1, a_1), \cdots, (S_l, a_l))$ and $puzzle = (prms, (S_1, u_1), \cdots, (S_l, u_l))$, then we can denote all this as $(p, X, puzzle) \leftarrow JGen(1^\lambda, k, l, A)$, where X is the private output and $puzzle$ is the public output.
- Jigsaw Verifier $JVer$ has input $puzzle$ and Multilinear Form $\mathcal{F} = (k, l, \Pi, F)$ and outputs 0 or 1. $JVer$ is correct relative to $(p, puzzle, \mathcal{F}, X)$ if either $\mathcal{F}(X) = ([k], 0) \wedge JVer(puzzle, \mathcal{F}) = 1$ or $\mathcal{F}(X) \neq ([k], 0) \wedge JVer(puzzle, \mathcal{F}) = 0$. It is required that with all but negligible probability, $JVer$ is correct for all forms $\mathcal{F}$.

Multilinear Jigsaw Puzzles are a subset of the Garg-Gentry-Halevi multilinear encoding scheme [GGH13a]. The main difference is that encoding can only be done by one party, in this case the Jigsaw Generator. The security follows from the security of the Garg-Gentry-Halevi multilinear encoding scheme.

The Jigsaw Generator can be seen to do an encryption of private plaintexts X into a public set of ciphertexts $puzzle$, where each encryption is done by the $Encode$ function. In this regard, the Jigsaw Verifier can be seen to do a homomorphic evaluation of a circuit using the encrypted inputs, with an added property that he can also see the output (0 or 1).

4 Indistinguishability Obfuscation: The Construction

In this section, we descuss how Garg et al. used the above building blocks to get indistinguishability obfuscation. The construction starts with creating indistinguishability obfuscation for NC^1, and then uses this and some tricks from fully-homomorphic encryption to get indistinguishability obfuscation for all polynomial-sized circuits.

4.1 Indistinguishability Obfuscation for NC^1

Indistinguishability obfuscation for NC^1 is achieved in three steps, related to the building blocks we have discussed:

1. Convert circuits in NC^1 to an equivalent matrix branching program, using Barrington's theorem.
2. Randomize / garble the matrix branching program using full rank randomizer matrices we get from Killian's protocol.
3. Encode and hide the randomizer matrices using Multilinear Jigsaw Puzzles.

Let C be a circuit in NC^1, which is transformed using Barrington's theorem to a matrix branching program $\{A_{i,b}\}_{i\in[n],b\in\{0,1\}}$. For additional garbling, we add a random large diagonal matrices $E_{i,b}, E'_{i,b}$ and random non-zero scalars $\alpha_{i,b}, \alpha'_{i,b}$ creating $D_{i,b} = \begin{Bmatrix} E_{i,b} & 0 \\ 0 & \alpha_{i,b}A_{i,b} \end{Bmatrix}$ and $D'_{i,b} = \begin{Bmatrix} E'_{i,b} & 0 \\ 0 & \alpha'_{i,b}I_{5\times 5} \end{Bmatrix}$. Hence we get a garbled branching program $BP = \{D_{i,b}\}_{i\in[n],b\in\{0,1\}}$ and a parallel garbled branching program $BP' = \{D'_{i,b}\}_{i\in[n],b\in\{0,1\}}$.

From Killian's protocol, we can define the randomized branching program with matrices $\tilde{D}_{i,b} = R_{i-1}D_{i,b}R_i^{-1}$ as follows:

$$\mathcal{RND}_p(BP) = \left\{ \begin{matrix} \tilde{s} = sR_0{}^{-1}, \tilde{t} = R_n t, & \tilde{s}' = s'R'_0{}^{-1}, \tilde{t}' = R'_n t' \\ \{\tilde{D}_{i,b} = R_{i-1}D_{i,b}R_i^{-1}\}_{i\in[n],b\in\{0,1\}}, & \{\tilde{D}'_{i,b} = R'_{i-1}D'_{i,b}R'^{-1}_i\}_{i\in[n],b\in\{0,1\}} \end{matrix} \right\}.$$

The randomization here is essentially two parallel programs, one which contains the original program BP, while the other is an arbitrary program of the same length, containing just identity matrices in the same position.

The branching program is now randomized, but may still be distinguishable using some attack patterns, e.g. by using partial evaluations (computing two products of $\tilde{D}_{i,b}$-s, but not necessarily full evaluations, and comparing them to each other) or miscellaneous use of the permutation matrices (such as computing matrix inverse). Hence we need Multilinear Jigsaw Puzzles to encode the branching program (using the Jigsaw Generator), but still be able to evaluate the branching program (using the Jigsaw Verfier).

If we regard $(p, \mathcal{RND}_p(BP))$ as the private output of a Jigsaw Generator, then the corresponding public output $\widehat{\mathcal{RND}}_p(BP)$ would be in a similar form but with encodings of all elements.

With these definitions and above building blocks, we can construct the candidate indistinguishability obfuscator $i\mathcal{O}(C) = GARBLE(\widehat{\mathcal{RND}}_p(UBP), (I_C, \sigma_C))$.

The security of this scheme relies on the following assumption:

Equivalent Program Indistinguishability (EPI) For a length-n branching program that computes a function $F : \{0,1\}^l \to \{0,1\}$, and any two partial assignments $(J, \sigma_0), (J, \sigma_1)$ functionally equivalent relative to F, the two corresponding garbled programs are computationally indistinguishable

$$GARBLE(\widehat{RND}_p(BP), (J, \sigma_0)) \approx GARBLE(\widehat{RND}_p(BP), (J, \sigma_1)).$$

The security assumption essentially states that using the $GARBLE$ function on a branching program will create indistinguishable output for any two different partial assignments of the program's variables. We cite the following theorem [GGH+13b] on security of the scheme.

Theorem 2. *The above construction* $i\mathcal{O}(c) = GARBLE(\widehat{\mathcal{RND}}_p(UBP), (I_c, \sigma_c))$ *is an efficient indistinguishability obfuscator under the EPI assumption.*

4.2 Indistinguishability Obfuscation for all polynomial-sized circuits

Assume we have a (leveled) fully homomorphic encryption scheme Enc_{FHE} with homomorphic evaluation function $Eval_{FHE}$, and decryption in NC^1. Intuitively, we can use the bootstrapping process of fully homomorphic encryption to reduce the circuit depth of any circuit to just logarithmic depth, as long as decryption in NC^1.

Let $\{C_\lambda\}$ be a family of polynomial-size circuits, and let $\{U_\lambda\}$ be the corresponding universal circuit family. Let $C \in \{C_\lambda\}$ be a circuit we wish to obfuscate. Obfuscation is done as follows:

1. Generate two different public key and private key pairs (PK_1, SK_1), (PK_2, SK_2) for FHE (and set the level $l = depth(U_\lambda)$ if applicable).
2. Compute encryptions $g_1 = Enc_{FHE}(PK_1, C)$ and $g_2 = Enc_{FHE}(PK_2, C)$.
3. Consider the program $P_1(SK_1, g_1, g_2)$. This is in NC^1, so we can define the obfuscation $P = i\mathcal{O}(P1)$.

Now, given a program input m, evaluation is done as follows:

1. Using the homomorphic evaluation function we can compute the encryption of $C(m)$ as $e_i = Eval_{FHE}(PK_i, U_\lambda(\cdot, m), g_i)$ for $i \in \{1, 2\}$.
2. These are evaluations of the circuit C, so we can have a low-depth proof ϕ that e_1, e_2 were computed correctly.
3. Finally, the obfuscation program we want is just $P(m, e_1, e_2, \phi)$.

We omit the details of the zero knowledge proof used in evaluation step 2. At a high level, the proof can be verified by a circuit in NC^1. Correctness is not evident, but the main thing required is for the FHE decryption circuit to be in NC^1. For the security proof, see [GGH+13b].

5 Discussion

The construction above is novel: there are no previously known constructions of indistinguishability obfuscation. However, the security of the construction has not been studied in detail. The main hardness assumption, EPI, is novel and covers all the known attacks. Although this gives us some confidence of its feasibility, it is by no means a proof that the assumption is sound, and needs more intense scrutiny to see if it is feasible. This is the reason the authors only state their construction to be a candidate construction.

The construction of Multilinear Jigsaw Puzzles is interesting, as if we have two "equivalent" private outputs X_1, X_2 then no multilinear form $\mathcal{F}$ can distinguish them, meaning that the corresponding public outputs $puzzle_1, puzzle_2$ would also be indistinguishable. This suggests that if we can have a nice definition for equivalency for the private outputs, we can get indistinguishability obfuscation directly from Multilinear Jigsaw Puzzles.

6 Summary

In this article, we analyzed the candidate construction of indistinguishability obfuscation by Garg et al. [GGH+13b]. We described the building blocks used to achieve this, and how they are used to get indistinguishability obfuscation for NC^1. We then described how this can be extended to indistinguishability obfuscation for $P/poly$.

While we did not argue much about the security, we discussed how the security of this construction relies on a hardness assumption that is not yet very well studied. This is an interesting topic for future research.

References

Bar86. David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. In Juris Hartmanis, editor, *STOC 1986*, pages 1–5, Berkeley, California, USA, May 28–30, 1986.

Gen09. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, September 2009.

GGH13a. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate Multilinear Maps from Ideal Lattices and Applications. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881, pages 1–17, Athens, Greece, April 26–30, 2013. Springer, Heidelberg.

GGH+13b. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013*, pages 40–49, Berkeley, CA, USA, October, 26–29 2013. IEEE, IEEE Computer Society Press.

Hol14. Thomas Holenstein. Branching Programs and Barringtons Theorem. Complexity Theory (FS14) lecture notes, Chapter 10, 15 December 2014. Available from `http://wwww.complexity.ethz.ch/education/Lectures/ComplexityFS14`.

Kil88. Joe Kilian. Founding Cryptography on Oblivious Transfer. In *STOC 1988*, pages 20–31, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.