Recent Advances in Indistinguishability Obfuscation and a Generic Construction from Functional Encryption

Nishant Yadav
Anshul Bawa
16 November 2015

Abstract
In this paper, we survey some recent work done in constructing indistinguishability obfuscation from public-key functional encryption. We also demonstrate some of the promising applications of indistinguishability obfuscation.

Introduction

Indistinguishability obfuscation aims to encrypt two circuits $C_1$ and $C_2$ that have same input-output behaviour (i.e. they compute the same function $f$, though their internal logic may be different) so that users of the obfuscated circuit $iO(C_b)$ cannot discern which of the two circuits $C_1$ and $C_2$ has been obfuscated.

Informally, an indistinguishability obfuscator is an efficient probabilistic algorithm $iO$ that preserves the input-output behaviour of the circuit (from a class of circuits $C$) it obfuscates, and any polynomially-bound algorithm can distinguish between the obfuscations of two functionally-equivalent circuits $C_1, C_2 \in C$ only with a negligible probability. There is an additional prerequisite for the circuits $C_1$ and $C_2$ to be of the same size.

Definition: Indistinguishability Obfuscation
A PPT algorithm $iO$ is said to be an indistinguishability obfuscator for a class of circuits $\mathcal{C}$, if it satisfies:

1. Functionality: for any $C \in \mathcal{C}$ and security parameter $\lambda$,
   
   $$\Pr_{iO}[\forall x : iO(C_1^1)(x) = C(x)] = 1$$

2. Indistinguishability: for any poly-size distinguisher $D$ there exists a negligible function $\mu()$, such that for any two circuits $C_0, C_1 \in \mathcal{C}$ that compute the same function and are of the same size:

   $$\left|\Pr[D(iO(C_0, 1^\lambda)) = 1] - \Pr[D(iO(C_1, 1^\lambda)) = 1]\right| \leq \mu(\lambda),$$

where the probability is over the coins of $D$ and $iO[2]$.

It has been shown in [4] that using a technique called punctured programs, indistinguishability obfuscation (along with one way functions) can be used to construct many of the core
primitives of cryptography, including CCA-secure public-key encryption, non-interactive zero-knowledge proofs (NIZKs) and injective trapdoor functions.

A punctured program $P^*$ is a program $P$ with a key element (crucial for a security attack) surgically removed without altering program functionality. Indistinguishability obfuscation is used to hide this puncturing from an adversary, and also to move it to a location that is almost never functionally accessed by the program [4].

Indistinguishability obfuscation is thus a very powerful notion, and its existence implies the existence of many of the known cryptographic objects.

Some known security assumptions of IO are based on not-so-well-studied mathematical objects called multilinear maps. But as they were only introduced to cryptographic scrutiny in 2013[2], their reliability hasn’t been rigorously assessed. There is a need to base IO on stronger cryptographic foundations, and this seems to be partly achieved by [2], on whose work most of this term-paper is based. They demonstrate how to achieve indistinguishability obfuscation using public-key functional encryption.

Public-key functional encryption is a type of encryption in which possessing a secret key $FSK$ allows one to learn a function of what the ciphertext is encrypting. Roughly speaking, functional encryption supports restricted secret keys that enable a key holder to learn a specific function of encrypted data, but learn nothing else about the data[2].

**Definition: Functional Encryption Scheme**
A functional encryption scheme $FE$ for a functionality $F$ defined over $(K, X)$ from a function class $F$, is a tuple of four PPT algorithms ($Setup, Keygen, Enc, Dec$) satisfying the following correctness condition $\forall k \in K$ and $\forall x \in X$:

\[
(pk, mk) \leftarrow Setup(1^\lambda) \quad \text{(generate a public and master secret key pair).}
\]

\[
sk \leftarrow Keygen(mk, k) \quad \text{(generate secret key for $k$)}
\]

\[
c \leftarrow Enc(pk, x) \quad \text{(encrypt message $x$)}
\]

\[
y \leftarrow Dec(sk, c) \quad \text{(use $sk$ to compute $F(k, x)$ from $c$)}
\]

then we require that $y = F(k, x)$ with probability one [2].

One of the primary challenges for IO has been the speed of the obfuscation process. In all the known constructions, the amount of computation needed to achieve it really slows things down. The general strategy followed to tackle this problem is to reduce the problem of obfuscating large circuits to that of connected smaller circuits.

In [2] too, the entire circuit for the FE scheme is not obfuscated, but only the encryption algorithm $Enc$ is. $Enc$ acts as a token-generator, and obfuscating it gives a token-based obfuscation, wherein to evaluate obfuscation of an input $x$, a token needs to be obtained first.

In the setting of private-key functional encryption, the encryption of input by the secret-key owner is the said token. It has been shown that it is always possible to guarantee function-hiding in private-key functional encryption schemes, by treating the function-hiding function secret key as the obfuscation for $f$. But the drawback of private-key FE is that the tokens
cannot be generated publicly without interaction with the secret-key owner. So this does not achieve IO in its full spirit, but does progress in that direction.

Private-key FE schemes can not be used for purpose of obfuscating the encryption algorithm because indistinguishability must hold even when the circuit is available publicly, and there exist schemes where access to the encryption circuit can lead to devastating attacks. Coming back to public-key functional encryption then, the central approach followed can be broadly divided into two steps:

- Reducing the problem to a simpler one: This is the idea of using token-based obfuscation wherein only the obfuscation of the encryption algorithm suffices in obfuscating $f$.

- Bootstrapping IO: This addresses the problem of obfuscating the encryption algorithm in the first place. Here, IO for an $n$-bit-encrypting circuit $Enc_n$ is recursively reduced to an instance of functional encryption and IO for an $(n-1)$-bit-encrypting circuit $Enc_{n-1}$. At the base of this recursion, we obfuscate one bit input circuits by merely writing their respective outputs. (In fact, the base case of recursion need not be one bit input circuits but any circuit that takes a constant number of input bits.)

Employing recursion to achieve obfuscation of the encryption can potentially double the size of the circuit to be obfuscated at each step, leading to a blowup in the size of the circuit to be obfuscated. In the construction mentioned in [2], the size of the circuit to be obfuscated stays proportional to the input size, which reduces gradually, and hence an exponential growth in size is avoided.

One of the primitives used in their construction is that of a puncturable PRF family.

A puncturable PRF is a special case of constrained PRF where a constrained key associated with an element $x'$ allows for evaluation of the PRF at all points but $x'$. Even given such a key, the output of the PRF at $x'$ cannot be distinguished from a random string by any poly-time distinguisher with non-negligible advantage [4].

The idea of using punctured schemes comes up more evidently when we study construction of full-domain hash functions using IO in a later section.

**Construction**

We now formally state the primitives used in the construction, followed by the construction itself [2].

- A $(2^{\tilde{\lambda}})$-secure single-key, selectively-secure, public-key functional encryption scheme $FE$ for $P/poly$, for a single key with (fully or weakly) succinct encryption.

- A $(2^{\tilde{\lambda}})$-secure one-time symmetric encryption scheme $Sym$.

- A $(2^{\tilde{\lambda}})$-secure puncturable pseudo-random function family $\mathcal{PRF}$

where $\tilde{\lambda}$ is the security parameter and $\epsilon < 1$
Analysis

We first demonstrate that this obfuscation preserves functionality of the obfuscated circuit. The recursive evaluation procedure $rO.Eval(i, E_i, x_i)$. The recursion starts folding at its base case of returning $FCT_{i+1}^1$ which is the function ciphertext corresponding to the first bit, and is simply $C_1(x_1)$. In all other steps $i$, given encryptions of both $(x_{i-1}, 0)$ and $(x_{i-1}, 1)$, the correct $FCT_{i+1}^x$ is decrypted with $FSK_i$ to produce $FCT_{i+1}^0$ and $FCT_{i+1}^1$ which are encryptions of $(x_i, 0)$ and $(x_i, 1)$ respectively. In the final step, this evaluates to the circuit itself, $C(x_n)$.

We now analyse the security guarantee of inability to distinguish between obfuscations of two circuits $C_0$ and $C_1$ of same size and functionality.

First we show that irrespective of what circuit is being encrypted, hardwiring either of the two symmetric encryptions $SK_0^0$ and $SK_1^1$ works, and in fact, these two worlds would be computationally indistinguishable.

The difference would lie in the obfuscated $E_{i-1}$, which would put $SK_0^0$ in the plaintext in one world and $SK_1^1$ in another. The key to indistinguishability is that the output of the
two circuits on any point \( x_{i-1} \in \{0, 1\}^{i-1} \) is indistinguishable even given the two circuits, as long as the randomness used to generate the output is not revealed. The FE guarantee does not therefore allow distinguishing between the two cases.

It is also true that indistinguishability holds across all the worlds where \( CT_0 \) and \( CT_1 \) each encrypt either of \( C_0 \) and \( C_1 \) (irrespective of what the other encrypts). Because the symmetric encryption is one-time secure, and because \( SK_0 \) is independent of \( SK_1 \), looking at \( CT_0 \) and \( CT_1 \) does not reveal information about correspondence to either \( C_0 \) or \( C_1 \).

In all cases, a punctured key is used to generate all encryptions other than the hardwired one.

Applications - Full Domain Hash Functions from Indistinguishability Obfuscation

To be able to instantiate a Random Oracle with a concrete hash function family has been one of the open questions in cryptography [6]. This would allow proving correctness of existing schemes which have already been proven to be secure in the Random Oracle model, without having to modify the underlying scheme at all.

In the paper [6], they give a positive answer to this by leveraging the idea of indistinguishability obfuscation. Because their construction makes use of IO and is therefore inherently a non-black-box, they bypass a known impossibility result [1] that there can be no black box construction of hash functions that allow for full-domain hash signatures to be based on trapdoor permutations.

Selectively Secure Full Domain Hash Functions

The security of Bellare-Rogaway scheme was earlier proven using the random oracle model which in practice is replaced by a hash function. In contrast, the construction in [6] bases its security on security of the indistinguishability obfuscator, the security of a puncturable PRF family and security of an injective TDP family.

One of the natural questions that arises is why do we need to obfuscate the circuits of the hash functions at all? This is important primarily because the behaviour of an adversary against the signature scheme could be different when the hash function used
is FDH than when it is FDH*, because the circuit of the hash function used is given out as part of the public key and from which an adversary can recover the secret key of the PRF.

Now since FDH* uses a constrained key, even after learning the key, the adversary can learn nothing about the signature for the punctured message, \( m^* \). But in case of FDH, the secret key leaked by the circuit allows for evaluation at all points in the domain, in particular \( m^* \). Thus an adversary can easily forge a signature.

IO ensures that the behaviour of the adversary is only negligibly different when FDH* is used instead of FDH, as otherwise the security of the IO scheme being used gets violated (because FDH and FDH* agree on all but a negligible fraction of inputs).

Basing our argument on the security of the PRF being used, we can replace \( F(k, m^*) \) by \( t^* \) (picked at random from \( \{0, 1\}^n \)) and argue that this replacement causes at most a negligible gain in advantage for the adversary.

Finally, the security of the injective trapdoor permutation helps us argue that any PPT adversary gains at most a negligible advantage in breaking the security of the latter world (where \( F(k, m^*) \) was replaced by a random \( t^* \)).

**Adaptively Secure RSA Signature**

In this setting, there are two modes for hash functions:

- **Normal Mode**: The hash function parameters are chosen at random
- **Partitioning Mode**: The hash function parameters are chosen according to a special distribution that allows for computation of inverse.

IO is used here for the purpose of hiding this partition which is otherwise leaked by the hash function parameters, even though the functionality of the hash function is identical in both the modes.

**IO with linear overhead**

Combining various results from literature, the construction mentioned in [2] implies that, assuming LWE (learning with errors), any IO scheme can be turned into IO scheme where
size of the obfuscation of circuit $C$ of depth $d$ is $2|C| + \text{poly}(d, n, \lambda)$.

At every recursive call to the function $rO.\text{Obf}(n, C, 1^\lambda)$, primarily three operations are performed which are

- Obfuscation of $E_{n-1}$,
- Generating FE encryption of inputs, and
- Computation of $FSK_n$ to perform evaluation of the circuit $C$.

Here, the size of the circuit $E_{n-1}$ is dominated by the complexity of FE encryption and the function $f_n$ can be represented by $2|C|$ bits, consisting of two one-time encryptions of $|C|$. (For example, using a PRG that expands $\lambda$ bits to $|C|$ bits as a one-time pad.)

**Related Work**

In an independent piece of work [3], it has been shown how to construct IO from sub-exponentially secure public-key functional encryption. A transformation from public key FE to multi-input functional encryption has been demonstrated and is shown to imply IO in another work by [5]. The question of whether private-key FE can be used to achieve construction of IO remains unanswered. However the notion of puncturable private-key FE suffices for the transformation mentioned in [2]. But achieving this without relying on public-key schemes is still an open question.

**References**


- Figures 1 and 2 from [2]
- Figures 3 and 4 from [6]