An Investigation of ADP-based Candidate Witness Encryption and IO Constructions

Weihang Xu, Yanming Wan, Zijian Zhu

October 30, 2021

In this project report, we mainly investigate the paper [BIJ$^+$20], which uses affine determinant programs to construct candidate witness encryption and IO schemes.

Indistinguishability Obfuscation can trivially be applied to construct Witness Encryption Scheme, so we first introduce how to implement an ADP-based Witness Encryption, through which the intuition of transforming ADPs are explained.

As for general program obfuscation, the problem can be reduced to obfuscating branching programs. With the theorem given by [AIK06], the branching program can be encoded into ADP matrices, allowing us to apply linear algebraic transformation ideas on them.

The paper mainly introduces three methods in obfuscating Branching Programs as well as briefly mentioning some potential attacks. We summarize the constructions and attacks in logical order, which makes it more straightforward to follow the paper's idea.

1 Affine Determinant Programs: Basics

Definition 1 (Affine Determinant Program). *An affine determinant program $ADP : \{0,1\}^n \to \{0,1\}$ with parameter n, k, p consists of an affine function $\mathbf{M} : \{0,1\}^n \to \mathbb{F}_p^{k\times k}$ together with an evaluation function $Eval : \mathbb{F}_p \to \{0,1\}$.*

The affine function $\mathbf{M}$ comprises a tuple of $n+1$ matrices $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n) \in (\mathbb{F}_p^{k\times k})^{n+1}$. On input $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, $\mathbf{M}$ is computed as

$$\mathbf{M}(\mathbf{x}) = \mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i,$$

and ADP is defined as

$$ADP(\mathbf{x}) = Eval(\det(\mathbf{M}(\mathbf{x}))).$$

The first step of ADP-based program obfuscation is to translate the program into ADPs. More specifically, an algorithm $ADP - Encode$ that translates functions into ADPs takes a function $f : \{0,1\}^n \to \{0,1\}$ and security parameter λ as input, outputs an equivalent ADP: $ADP - Encode(\langle f\rangle, 1^\lambda)(\boldsymbol{x}) = f(\boldsymbol{x})$ for $\forall \boldsymbol{x}$.

The following definition characterizes the basic structure of ADP-based IO schemes.

Definition 2 (IO-security for ADP-encoding). *The encoding algorithm $ADP - Encode$ is $iO - secure$ if for any two equivalent functions $f_1 \equiv f_2$ and $|\langle f_1\rangle| = |\langle f_2\rangle|$,*

$$ADP - Encode(\langle f_1\rangle, 1^\lambda) \approx_C ADP - Encode(\langle f_2\rangle, 1^\lambda)$$

holds.

2 ADP-based Witness Encryption

Recall the witness encryption is to use a instance x of NP language L to encrypt a message $m \in \{0,1\}$. When $x \in L$, one can decrypt the message m using a witness w of x. When $x \notin L$, the encryption should hide the message m.

In this section, we consider the NPC problem VECTOR-SUBSET-SUM (abbreviated as VSS), since it can be easily formalized to the following matrix form.

Definition 3 (VECTOR-SUBSET-SUM). *The VECTOR-SUBSET-SUM language consists of instances* $(\mathbf{H}, \boldsymbol{\ell})$*, where* $\mathbf{H} \in \mathbb{Z}^{d\times n}$*,* $\boldsymbol{\ell} \in \mathbb{Z}^d$*, such that there exists* $\mathbf{w} \in \{0,1\}^n$ *satisfying* $\mathbf{H}\cdot\mathbf{w} = \boldsymbol{\ell}$*.*

Then we try to use ADP to encode a VSS instance $(\mathbf{H}, \boldsymbol{\ell})$. In other word, we hope this ADP can distinguish the witness of $(\mathbf{H}, \boldsymbol{\ell})$ from other vectors. This can be done by setting $\mathbf{A} := -\sum_{j\in[d]} \ell_j \mathbf{R}_j$, $\mathbf{B}_i := \sum_{j\in[d]} H_{j,i}\mathbf{R}_j$, where $\mathbf{R}_j$ are random matrices sampled from $\mathbb{F}_p^{k\times k}$ (we interpret $\mathbf{H}$ and $\boldsymbol{\ell}$ over $\mathbb{F}_p$, p is a prime and superpolynomial in input). And then output $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell}} = (\mathbf{A}, \mathbf{B}_1, ..., \mathbf{B}_n)$.

So we have

$$\mathbf{M}_{\mathbf{H},\boldsymbol{\ell}}(\mathbf{x}) = \mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i = \sum_{j\in[d]} (-\boldsymbol{\ell} + \mathbf{H}\cdot\mathbf{x})_j \mathbf{R}_j$$

When $\mathbf{H}\cdot\mathbf{x} = \boldsymbol{\ell}$, $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell}}(\mathbf{x}) = \mathbf{O}$. Otherwise, $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell}}(\mathbf{x})$ is the sum of some random matrices $\mathbf{R}_j$.

2.1 An Insecure Witness Encryption

From the ADP encoding for VSS, we can easily design a trivial encryption scheme. To encrypt a bit b with respect to an instance $(\mathbf{H}, \boldsymbol{\ell})$, we just add $b\mathbf{S}$ to matrix $\mathbf{A}$ of the ADP encoding, where $\mathbf{S}$ is a random matrix sampled from $\mathbb{F}_p^{k\times k}$. So the ADP we get is $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell},b} = \mathbf{M}_{\mathbf{H},\boldsymbol{\ell}} + (b\mathbf{S}, 0, ..., 0)$.

To decrypt a message by using witness $\mathbf{w}$, we just calculate $b = \mathrm{Eval}_{\neq 0}(\det(\mathbf{M}_{\mathbf{H},\boldsymbol{\ell},b}(\mathbf{w})))$ (since $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell},b}(\mathbf{w}) = b\mathbf{S}$).

However, this construction is insecure. Given an instance $(\mathbf{H}, \boldsymbol{\ell}) \notin$ VSS, an adversary can find $\mathbf{x} \in \mathbb{F}_p^n$ by solving linear equations in $\mathbb{F}_p$ such that $\mathbf{H}\cdot\mathbf{x} = \boldsymbol{\ell}$ (note that $\mathbf{x}$ may not in $\{0,1\}^n$), so the encrypted bit b can be recovered by performing evaluation on $\mathbf{x}$.

2.2 Preventing Invalid Evaluations

So we want to prevent the adversary from performing invalid evaluations. That is, when the adversary evaluates on some invalid inputs, we add a 'noise' to $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell},b}$, so that the adversary can't gain information on non-binary inputs. And for valid inputs, the 'noise' shouldn't interfere with the evaluation. To formalize this idea, the 'noise' should take the form of another ADP $\mathbf{M}_{noise}$, which satisfies the following properties:

- For all $\mathbf{x} \in \{0,1\}^n$, $\det(\mathbf{M}_{noise}(\mathbf{x})) = 0$.
- For all $\mathbf{x} \in \mathbb{F}_p\backslash\{0,1\}^n$, $\Pr[\det(\mathbf{M}_{noise}(\mathbf{x})) = 0] = \mathrm{negl}(n)$.

2.3 Constructions of AllAcc.Gen

Let AllAcc.Gen is a randomized procedure that generates $\mathbf{M}_{noise}$. The main idea of constructing AllAcc.Gen is that we need $\mathbf{M}_{noise}$ to be rank-deficient on binary inputs and to be full-rank on non-binary inputs. The paper gives two constructions of AllAcc.Gen. For simplicity, we only introduce one of them, which is equivalent to the ADP encoding of boolean formula $f(x_1, ..., x_n) = (x_1 \vee \neg x_1) \wedge (x_2 \vee \neg x_2) \wedge ... \wedge (x_n \vee \neg x_n)$.

2.3.1 An All-Accept ADP from a Boolean Formula Encoding

AllAcc.Gen$^{\text{FORM}}(n, \mathbb{F}_p)$: Let $\mathbf{A} := \sum_{i\in[n]} \mathbf{u}_i\mathbf{v}_i^\top, \mathbf{B}_i := -\mathbf{u}_i\mathbf{v}_i^\top + \mathbf{s}_i\mathbf{t}_i^\top + \sum_{j\in[n]} c_i^{(j)}\mathbf{u}_j\mathbf{t}_j^\top$ (where $\{\mathbf{u}_i\}_{i\in[n]}, \{\mathbf{v}_i\}_{i\in[n]}, \{\mathbf{s}_i\}_{i\in[n]}, \{\mathbf{t}_i\}_{i\in[n]}$ are 4 sets of random vectors from $\mathbb{F}_p^{n+1}$, $\{c_j^{(i)}\}_{i,j\in[n]}$ are random numbers from $\mathbb{F}_p$). Output $\mathbf{M} = (\mathbf{A}, \mathbf{B}_1, ..., \mathbf{B}_n)$.

It's not hard to verify the correctness of this construction by discussing on the possible values of x_i.

However, for the SUBSET-SUM instance with scarce non-zero elements, most of $\mathbf{A}$ and $\mathbf{B}_i$ of $\mathbf{M}_{noise}$ are clear to the adversary. Then the adversary can potentially extract some information about the sampling process (If we can recover $\mathbf{u}_i$, the process of sampling $\mathbf{u}_i$ is meaningless). As a concrete example, consider the instance $(\mathbf{h} = (1, 0, ..., 0), \ell = 2)$. Obviously, $(\mathbf{h}, \ell) \notin$ SUBSET-SUM, but all $\mathbf{B}_2, ..., \mathbf{B}_n$ of $\mathbf{M}_{noise}$ are clear to the adversary.

This paper gives two attacks to AllAcc.Gen$^{\text{FORM}}$. One of them based on the rank-deficiency of matrices $\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i$. And the other is based on the construction of low rank matrix consists of unknown variables which gives a system of non-linear equations. Both of them can recover some of $\{\mathbf{u}_i\}_{i\in[n]}, \{\mathbf{v}_i\}_{i\in[n]}, \{\mathbf{s}_i\}_{i\in[n]}, \{\mathbf{t}_i\}_{i\in[n]}$ up to a scalar.

These attacks show the importance of hiding the outputs of AllAcc.Gen.

2.4 A Method to Hide $\mathbf{B}_i$

Given an instance $(\mathbf{h} \in \mathbb{Z}^n, \ell \in \mathbb{Z})$ of SUBSET-SUM, we can convert it to the following equivalent instance $(\mathbf{H} \in \mathbb{Z}^{(n+1)\times 2n}, \boldsymbol{\ell} \in \mathbb{Z}^{n+1})$ of VECTOR-SUBSET-SUM, where

$$\mathbf{H} := \begin{pmatrix} h_1 & h_2 & \cdots & h_n & 0 & 0\cdots & 0 \\ 1 & 0 & \cdots & 0 & 1 & 0\cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1\cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots\ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0\cdots & 1 \end{pmatrix}, \boldsymbol{\ell} := \begin{pmatrix} \ell \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Notice that every column of $\mathbf{H}$ is non-zero, which will avoid all matrices of $\mathbf{M}_{noise}$ from being given in clear by adding a random matrix to each of them.

2.5 Candidate Witness Encryption Scheme

To conclude this section, we give the final candidate witness encryption scheme from ADP.
WE.Enc$(b, (\mathbf{H}, \boldsymbol{\ell}))$: (Let $d \times n$ be the dimension of $\mathbf{H}$.)

1. Sample $(\mathbf{M}_{noise}, k) \leftarrow$ AllAcc.Gen$(n, \mathbb{F}_p)$, $\mathbf{M}_{\mathbf{H},\boldsymbol{\ell}} \leftarrow$ VSS.Encode$((\mathbf{H}, \boldsymbol{\ell}), \mathbb{F}_p, k)$, $\mathbf{S} \leftarrow \mathbb{F}_p^{k\times k}$.
2. Output $\mathbf{M}_{noise} + \mathbf{M}_{\mathbf{H},\boldsymbol{\ell}} + (b\mathbf{S}, \mathbf{0}, ..., \mathbf{0})$.

3 Encoding Branching Programs as ADPs: Method and Motivation

Definition 4 (Branching Program). *A branching program with input variables $(x_1, \ldots, x_n), x_i \in \{0, 1\}$ is a DAG $G = (V, E)$ with source s and sink t where every edge (u, v) has a label $\phi(u, v) = x_i$ or $\neg x_i, i \in [n]$.*

On a set of given input values $(x_1, \ldots, x_n)$, the branching program outputs the number of $s - t$ paths where the label ϕ on each edge of the path has value 1.

Theorem 5. *For a branching program $f : \{0,1\}^n \to \mathbb{N}$ on DAG $G = (V, E)$, first assign a topological order to the vertices: $V = \{s = v_1, v_2, \ldots, t = v_m\}$. Construct a $m \times m$ matrix A satisfying*

$$A_{ij} = \begin{cases} \phi(v_i, v_j) & (v_i, v_j) \in E \\ 0 & else \end{cases},$$

suppose L is the matrix obtained by deleting the first column and last row of $A - I$, then $\det(L(\boldsymbol{x})) = f(\boldsymbol{x})$ for $\forall \boldsymbol{x} \in \{0,1\}^n$.

Proof [AIK06]: since s and t are ordered as v_1, v_m, the number of $s - t$ paths of length i is $(A(\boldsymbol{x})^i)_{m,1}$. Note that $A(\boldsymbol{x})^m = O$ since no path in G has length longer than $n - 1$, therefore

$$\begin{aligned} f(\boldsymbol{x}) &= (\sum_{i=0}^{m-1} A(x)^i)_{m,1} = ((I - A(\boldsymbol{x}))^{-1}(I - A(\boldsymbol{x})^m))_{m,1} \\ &= (I - A(\boldsymbol{x}))^{-1}_{m,1} = (-1)^{m+1}\frac{\det(-L(\boldsymbol{x}))}{\det(I - A(\boldsymbol{x}))} = \det(L(\boldsymbol{x})) \end{aligned}$$

□

This theorem immediately yields an ADP for any branching program since L could be rewritten in the form of an affine function: $L(\boldsymbol{x}) = A + \sum_{i\in[n]} x_i B_i$. In fact, now we show that if we could obfuscate such an ADP L (which would become our major concern later), then (under the assistance of some other cryptographical primitives) the obfuscation could be generalized to any polynomial-sized circuit.

By Barrington's Theorem, the computation power of branching programs is no weaker than that of circuits in NC^1, so obfuscation for branching programs naturally provides an obfuscation for circuits in NC^1.

Now we introduce an interesting (conditional) reduction from general IO to IO in NC^1 provided in [GGH+13]. Combining this reduction with Barrington's Theorem, the obfuscation for branching programs could be converted into an IO scheme for any polynomial-size circuit.

Suppose we have an indistinguishable obfuscator iO_{NC^1} for circuits in NC^1, also an fully homomorphic encryption scheme $FHE = (Gen, Enc, Dec, Eval)$ where the evaluation function $Eval$ satisfies that for any function f, key pair $(pk, sk) \leftarrow Gen(1^\lambda)$, messages $m_1, \ldots, m_n$,

$$Dec_{sk}(Eval(pk, f, c_1, \ldots, c_n)) = f(m_1, \ldots, m_n).$$

Here $c_1, \ldots, c_n$ denotes the ciphertexts of $m_1, \ldots, m_n$: $c_i = Enc_{pk}(m_i), \forall i$. Also, suppose the decryption algorithm Dec is in NC^1, we construct an indistinguishable obfuscator for any polynomial-sized circuit C.

First consider a gadget: an universal circuit U such that for any circuit C and massage m, U takes (C, m) as input and outputs $U(C, m) = C(m)$.[1] To obfuscate a polynomial-sized circuit C, the obfuscator Obf does the following:

1. Sample two pairs of keys $(pk_1, sk_1), (pk_2, sk_2) \leftarrow Gen(1^\lambda)$ independently.
2. Encrypt the circuit C with two key pairs respectively and get $C_1 = Enc_{pk_1}(C), C_2 = Enc_{pk_2}(C)$.

[1] In fact, since the input width of a circuit is fixed, what we should be really considering here is an universal circuit **family** $\{U_n\}$, each member of the family handles all circuits of a fixed input width n. But for simplicity here we only consider one single circuit U.

3. Construct program P:

   P

   (a) On input (m, t_1, t_2, ϕ), check whether ϕ is a proof for the NP statement

   $$t_1 = Eval(pk_1, U(\cdot, m), C_1) \wedge t_2 = Eval(pk_2, U(\cdot, m), C_2)$$

   (b) If the check fails, output $\perp$, otherwise output $Dec_{sk_1}(t_1)$.

4. Obfuscate P using the NC^1 obfuscator and get $iO_{NC^1}(P)$. Output the obfuscated version of C as

   Obf(C)

   (a) On input m, compute $t_1 = Eval(pk_1, U(\cdot, m), C_1)$, $t_2 = Eval(pk_2, U(\cdot, m), C_2)$.

   (b) Compute a proof ϕ proving that t_1, t_2 are computed correctly.

   (c) Run $iO_{NC^1}(P)(m, t_1, t_2, \phi)$ and output the result.

Remark 6. *The main idea of the construction is that, since the encryption scheme is fully homomorphic, we could compute the (encrypted version of) output of C on input m using the encrypted version of C: $Eval(pk_1, U(\cdot, m), C_1) = Enc_{pk_1}(U(C, m)) = Enc_{pk_1}(C(m))$. The way of decrypting $Enc_{pk_1}(C(m))$ is to construct the decryption-program P and obfuscate it, the secret keys are hidden during the process of obfuscation.*

Remark 7. *To use the obfuscator iO_{NC^1}, it has to be guaranteed that $P \in NC^1$. Therefore P cannot check whether t_1, t_2 are legal inputs directly. Instead, P requires the proof ϕ about the legality of t_1, t_2 as input, and check the correctness of ϕ.*

Remark 8. *The setting of using two pairs of keys $(pk_1, sk_1), (pk_2, sk_2)$ might seemed counter-intuitive. This actually was designed to accommodate the definition of IO, and to swap between the two pairs of keys using hybrid arguments during the security proof. (So maybe this* ***is*** *sort of counter-intuitive?)*

The security of the above construction can be proven by a rather straightforward hybrid argument, and details are omitted for simplicity.

4 ADP-based Obfuscation Scheme

Recall the **Theorem 5**, we are going to obfuscate the ADP matrices of branching programs. The paper mentions the concept of One-Time Security[2], which is much weaker than iO Security. But achieving One-Time Security can help us understand the goal of obfuscation.

4.1 Obfuscating ADP with Random Matrices

For an ADP $\mathbf{M} = (\mathbf{A}, \mathbf{B}_1, ..., \mathbf{B}_n)$, a trivial idea is to sample two random matrices $\mathbf{U}, \mathbf{V}$ such that $\det(\mathbf{U}) = \det(\mathbf{V}) = 1$ and let $\mathbf{M}' = (\mathbf{A}' = \mathbf{UAV}, \mathbf{B}'_1 = \mathbf{UB}_1\mathbf{V}, ..., \mathbf{B}'_n = \mathbf{UB}_n\mathbf{V})$. The

[2]The definition of One-Time Security is not quite important in this paper, so we omit it.

correctness is easy to check since

$$\det(\mathbf{A}' + \sum x_i \mathbf{B}'_i) = \det(\mathbf{U}) \det(\mathbf{A} + \sum x_i \mathbf{B}_i) \det(\mathbf{V}) = \det(\mathbf{A} + \sum x_i \mathbf{B}_i)$$

The motivation of this simple obfuscation is to randomized the ADP matrices while not violating the correct output on ***binary*** inputs. Here comes an important weakness. Determinant is a multi-linear polynomial function of $x_1, x_2, \ldots, x_n$, and the attacker is not restricted to only evaluating on binary inputs.

The paper use the ADP-encoding of a BP for point function to stress this ***Polynomial Extension Attack***. The determinant to be evaluated is $\prod_i (1 + 2v_i x_i - v_i - x_i)$, assigning $x_i = 0.5$ would lead to fixed value 0.5 for that term, eliminating the effect of v_i. Therefore, an input $(0, 0.5, 0.5, \ldots, 0.5)$ actually decouples the effect of v_i's, allowing the attacker to get v_1.

4.2 Avoiding Invalid (non-binary) Inputs

In previous parts, the determinant output is either 0 or 1 for binary function on binary inputs, so that the evaluation can simply check if it's zero or nonzero. Forcing a determinant to be zero is much easier on non-binary inputs, thus a direct idea is to "avoid" zero-valued determinants. This leads to the idea of ***Adding Even-valued Noise***. We can sample integral-valued error matrices Err_i and construct the obfuscation as

$$\mathbf{M}' = (\mathbf{A}' = \mathbf{U}(\mathbf{A} + 2Err_0)\mathbf{V}, \mathbf{B}'_1 = \mathbf{U}(\mathbf{B}_1 + 2Err_1)\mathbf{V}, ..., \mathbf{B}'_n = \mathbf{U}(\mathbf{B}_n + 2Err_n)\mathbf{V})$$

In this way, the original determinant is shifted by an even value, indicating that the evaluation function should now check the *parity* of output instead.

Unlike zero/non-zero evaluation, the parity can be more controllable to protect the scheme from invalid inputs. By restricting the sampling space χ of error matrices and choosing appropriate modulus prime p. The paper also argues that the error should be super-polynomial. There may exist distinguisher through estimating the magnitudes of determinants if the errors are of small size. Under the setting

$$\chi = [-B(l), B(l)],\ B(l) = l^{\omega(1)},\ p = \Theta((n \cdot B(l) \cdot \sqrt{l})^l),$$

the obfuscation manages to satisfy

- On binary inputs, the magnitude of determinant is smaller than p w.h.p.
- On non-binary inputs, the magnitude of determinant is greater than p w.h.p., so that the parity information is hidden over modulo p process.

The above construction still fail to treat short but non-binary inputs, leading to another interesting transformation idea by ***Adding Blocks on Diagonal***. Briefly, it also provide the non-binary inputs with large and random result. We need to independently sample $2n$ small matrices $\{G_i\}, \{H_i\}$ in same size, all with determinant 1. The transformation pad the original ADP matrices to larger ones as follows:

$$\begin{aligned}\mathbf{A}' &= \mathbf{U} \cdot diag(\mathbf{A}, \mathbf{G_1}, \ldots, \mathbf{G_n}) \cdot \mathbf{V},\\ \mathbf{B'_i} &= \mathbf{U} \cdot diag(\mathbf{B_i}, 0, \ldots, 0, \mathbf{H_i} - \mathbf{G_i}, 0, \ldots, 0) \cdot \mathbf{V}.\end{aligned}$$

If the input is binary, all blocks on the diagonal should be either G_i or H_i (with determinant 1), so that the result is not affected. But if the input in non-binary, there would be some block in the form $aG_i + bH_i$ with nonzero a and b, which incurs a large and random determinant with high probability.

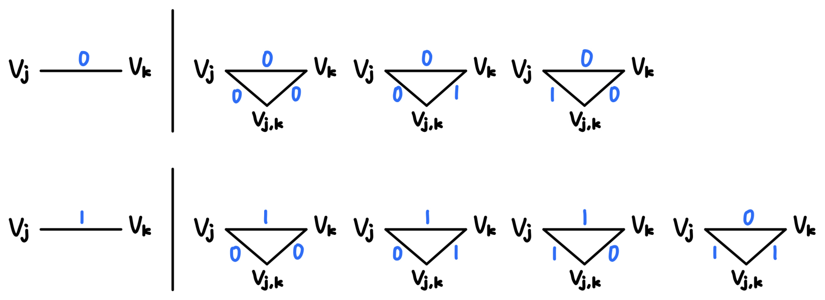
4.3 Adding More Randomness into Original DAG

Things can be more complicated even if the inputs are valid. If the ADP function $L(\mathbf{x})$ is fixed and known, attacker can actually recover the error matrices drawn in **AddNoise** operation. Suppose the (j,k) entry of Err_i is $e^{(i)}_{j,k}$. Then for any input $\mathbf{x} \in \{0,1\}^n$,

$$\det(\mathbf{A}' + \sum_i x_i \mathbf{B}'_\mathbf{i}) = \det(L(\mathbf{x})) + \sum_{j,k} (\sum_i x_i 2 e^{(i)}_{j,k}) \det(L(\mathbf{x})_{(j,k)}) \mod 4$$

The only unknown terms in the equation above are $e^{(i)}_{j,k}$'s. There are polynomial such variables, but we have exponential equations (corresponding to all valid inputs). With high probability there are sufficient linearly independent ones and the attacker can solve the equation system. Recovering all the sampled noise make our obfuscation operation meaningless.

To treat this problem, we can turn back to original DAG of the branching programs and inject some randomness. While maintaining the functionality, each edge $v_j v_k$ (corresponding to width-1 ADP) can be replaced by a "triangle" (corresponding to width-2 ADP) by adding a new vertex $v_{j,k}$. The intuition can be explained by the following figure.



The replaced graphs above give 3 matrices $ZERO_c$ and 4 matrices ONE_d in ADP. An edge labelled with 0 or 1 can simply assign $\mathbf{A}'$ with one of three or four matrices, leaving all $\mathbf{B}'_\mathbf{i}$ zero. As for x_i, we can assign $\mathbf{A}' = ZERO_c$ and $\mathbf{B}'_\mathbf{i} = ONE_d - ZERO_c$; it's similar for $1 - x_i$, and there are 12 replacement choices.

This idea is called ***Random Local Substitution***. Intuitively, the attacker can no longer know the exact $L(\mathbf{x})$ for sure or even guess, since there are exponential possible choices of replacement. But the functionality doesn't change since the number of all-1 paths keeps the same.

Based on all above construction, the final candidate raised by the paper can simply be written as

$$\mathbf{AddBlockDiagonals}(\mathbf{AddNoise}(\mathbf{RLS}(\mathrm{ADP})))$$

Though the iO security of this obfuscation cannot be proved, it can defend commonly seen attacks mentioned in this section.

Further Comments

Affine Determinant Programs theory is introduced to encode the Branching Programs, which transforms the program obfuscation problem into a linear algebraic topic on matrices. The paper [BIJ$^+$20] formalize the ADP-based problem, and give candidate schemes to defend some commonly-seen potential attacks. The attacks in the paper are not quite in detail, for many are only explained by some intuitive toy examples; and the candidate scheme's security (within cryptography standard) hasn't been proved explicitly. Nevertheless, this paper takes an significant step to pave the new way for program obfuscation. We are looking forward to further research in more concrete constructions and security proofs.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[BIJ$^+$20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 82:1–82:39. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 479–499. Springer, 2013.