Cryptanalysis of Candidate Obfuscators for Affine Determinant Programs

Li Yao[1], Yilei Chen[2,3], and Yu Yu[1,3]

[1] Shanghai Jiao Tong University, Shanghai, China 200240
[2] Tsinghua University, Beijing, China 100084
[3] Shanghai Qi Zhi Institute, Shanghai, China 200232
{pegasustianma,chenyilei.ra,yuyuathk}@gmail.com

Abstract. At ITCS 2020, Bartusek et al. proposed a candidate indistinguishability obfuscator ($i\mathcal{O}$) for affine determinant programs (ADPs). The candidate is special since it directly applies specific randomization techniques to the underlying ADP, without relying on the hardness of traditional cryptographic assumptions like discrete-log or learning with errors. It is relatively efficient compared to the rest of the $i\mathcal{O}$ candidates. However, the obfuscation scheme requires further cryptanalysis since it was not known to be based on any well-formed mathematical assumptions.

In this paper, we show cryptanalytic attacks on the $i\mathcal{O}$ candidate provided by Bartusek et al. Our attack exploits the weakness of one of the randomization steps in the candidate. The attack applies to a fairly general class of programs. At the end of the paper we discuss plausible countermeasures to defend against our attacks.

Keywords: indistinguishability obfuscation · cryptanalysis · affine determinant program.

1 Introduction

Indistinguishability Obfuscation ($i\mathcal{O}$) [BGI+01] is a probabilistic polynomial-time algorithm that transforms a circuit C into an obfuscated circuit $C' = i\mathcal{O}(C)$ while preserving the functionality. In addition, for any functionally equivalent circuits C_1 and C_2 of the same size, $i\mathcal{O}(C_1)$ and $i\mathcal{O}(C_2)$ are computationally indistinguishable. $i\mathcal{O}$ is a powerful cryptographic primitive with a wide variety of applications in cryptography and complexity theory. Indeed indistinguishability obfuscation, when combined with a minimal cryptographic primitive (one-way functions), is generally regarded as "crypto complete", implying almost all cryptographic applications currently known (e.g., [SW14, KLW15, BPR15, CHN+16]).

Despite the remarkable success in basing cryptographic applications on $i\mathcal{O}$, constructing efficient and provably secure $i\mathcal{O}$ remains a long-standing open problem in cryptography. While still far from the ultimate goal, many $i\mathcal{O}$ candidates have been provided in the past eight years. They can be generally classified as follows:

Candidates from multilinear maps. The initial $i\mathcal{O}$ candidates are built based on multilinear maps (a.k.a. graded encodings) [GGH13a, CLT13, GGH15]. Starting from the first $i\mathcal{O}$ candidate of Garg et al. [GGH$^+$13b], these candidates have gone through several rounds of break-and-repair. (see, e.g. [BGK$^+$14, BR14, CHL$^+$15, HJ16, MSZ16, CGH17]). To date, some variants of the original candidate of Garg et al. [GGH$^+$13b] remain secure, but no security proofs were known for any of those variants without using strong idealized models.

Candidates from succinct functional encryption. A remarkable line of works has been dedicated to building $i\mathcal{O}$ from succinct functional encryption schemes, which can then be based on well-founded assumptions, including LPN, DLIN in pairing and PRG in NC^0 [AJ15, BV15, Lin16, LV16, Lin17, LT17, AJL$^+$19, JLMS19, GJLS20, JLS21a, JLS21b]. They build $i\mathcal{O}$ via a series of reductions and take advantage of many cryptographic primitives, including attribute-based encryption, fully homomorphic encryption, FE for quadratic functions, homomorphic secret-sharing, universal circuits, etc. The downside of those candidates is that the overall constructions are complicated and far from efficient.

Candidates based on non-standard lattice assumptions. We also have lattice-based candidates without using pairing or multilinear maps. [BDGM20a, GP21, BDGM20b] construct candidates based on a strong circular-security assumption. [WW21] shows that oblivious LWE sampling implies $i\mathcal{O}$ and gives a candidate based on a circularity-like conjecture. Unfortunately, [HJL21] provides counterexamples to both assumptions. Apart from the circularity-based candidates, some works try to base $i\mathcal{O}$ on Noisy Linear FE [Agr19, AP20]. Recent work [DQV$^+$21] improves on [WW21] by basing iO on succinct LWE sampling, a weaker notion. It presents a candidate whose security is related to the hardness of solving systems of polynomial equations. The security of all these candidates relies on non-standard assumptions.

Candidate for affine determinant program. Finally, a special candidate obfuscator, which is the focus of this work, is provided by Bartusek et al. [BIJ$^+$20] for obfuscating affine determinant programs. An affine determinant program (ADP): $\{0,1\}^n \to \{0,1\}$ is specified by a tuple of square matrices $(\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n)$ over $\mathbb{F}_q$ and a function $\mathsf{Eval} : \mathbb{F}_q \to \{0,1\}$. It evaluates on input $\mathbf{x} \in \{0,1\}^n$ and produces an output $\mathsf{Eval}(\det(\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i))$. Non-uniform log-space computations (denoted by $\mathsf{L/poly}$) can be transformed into polynomial-size ADPs. Since $\mathsf{NC}^1 \subseteq \mathsf{L/poly}$, an obfuscator for such ADPs can serve as an obfuscator for NC^1 circuits, which implies general purpose $i\mathcal{O}$ additionally assuming the existence of fully homomorphic encryption [GGH$^+$13b].

The obfuscation candidate based on ADP is unique since it is the only unbroken candidate to date that does not rely on any traditional cryptographic assumptions like discrete-log or LWE. The candidate is also relatively simple to describe. In addition, the current quantum techniques do not seem to show special advantage in breaking the ADP-based candidate. So if LWE is broken by

a quantum algorithm in future, the obfuscation candidate for ADP might be the only living $i\mathcal{O}$ candidate against quantum computers.

The idea of using the ADP program model for obfuscation was also used in the earlier paper of Bartusek et al. [BLMZ19] for obfuscating conjunctions, where they can achieve provable security based on standard cryptographic assumptions. However, obfuscating a general program requires significantly different ideas. Indeed, the lack of security reduction from any well-formed assumption also means that the security of the candidate in [BIJ+20] requires more investigation.

1.1 Main Result

In this work, we show cryptanalytic attacks against the $i\mathcal{O}$ candidate of Bartusek et al. [BIJ+20]. Our attack can be seen as a variant of the "mod 4 attack" mentioned in [BIJ+20, Section 9.3]. The "mod 4 attack" was originally discussed in [BIJ+20, Section 9.3] as an attack for breaking a simpler version of the obfuscation scheme. It was also the motivation of adding a layer of randomization called Random Local Substitutions (RLS). However, we show that even with the RLS they provide, we can still manipulate the "mod 4 attack" in some other way to break the $i\mathcal{O}$ candidate.

To explain what kind of programs our attack applies to, let us describe the necessary and sufficient conditions separately. The *necessary* condition of the programs where our attack applies is that we can efficiently find four inputs of the form $\mathbf{x}_1 = \mathbf{a0b0c}, \mathbf{x}_2 = \mathbf{a1b0c}, \mathbf{x}_3 = \mathbf{a0b1c}, \mathbf{x}_4 = \mathbf{a1b1c}$ s.t. the program outputs the same value on these inputs, where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are some fixed strings of arbitrary length. Our attack will exactly run on those four inputs. The *sufficient* conditions are more complex to describe in their general forms. They deal with the minors of the matrices used in the ADPs to be obfuscated. Here let us mention a simple sufficient condition, that is, if two of the matrices among the n are all zero. Namely, for $1 \leq i < j \leq n$, $\mathbf{B}_i = \mathbf{B}_j = \mathbf{0}$. If so then we can distinguish the obfuscated version of such programs with those of functionally equivalent ADPs with $\mathbf{B}_i \neq \mathbf{0}$ or $\mathbf{B}_j \neq \mathbf{0}$. The general sufficient conditions relax the constraint such that we do not require the underlying branching program to contain all-zero matrices. This makes our attack work for a fairly general class of programs. However, as mentioned, the precise conditions on which our attack applies is a bit complicated. We refer readers to Section 5.3 for details.

At the end of the paper we provide some revisions of the RLS randomization which plausibly defends the obfuscation scheme against our attack. Let us also remark that the witness encryption candidate in [BIJ+20] is constructed via a somewhat different methodology, to which our attack does not apply.

1.2 Our Ideas in a Nutshell

To obfuscate an ADP, Bartusek et al. [BIJ+20] sample independent even noises and add them to each entry of $\{\mathbf{A}, \mathbf{B}_{i\in[n]}\}$. However, they also notice that the adversary can extract the parities of the noises by computing the determinant

first and then computing the result mod 4 after adding noises. The coefficients of the parities are minors of $\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}$, which are known to the adversary.

To defend against the attack, they introduce Random Local Substitutions, aiming to substitute the ADP P chosen by the adversary with another ADP $P' = \mathsf{RLS}(P)$, while preserving the functionality. The intuition that how the RLS comes to rescue is that by applying RLS to P the adversary cannot learn minors of $\mathbf{A}' + \sum_{i\in[n]} x_i\mathbf{B}'_i$, where $\{\mathbf{A}', \mathbf{B}'_{i\in[n]}\}$ are matrices of $\mathsf{RLS}(P)$. However, as we will show in this paper, it is not necessary to learn the coefficients of the parities to carry out the attack. We sketch the idea of our attack below.

Our attack starts from a well-crafted kind of ADP. Consider the simplest case where $n = 2$. We observe that if for $i \in \{1, 2\}$, $\mathbf{B}_i$ of P is a zero matrix, then $\mathbf{B}'_i$ of $P' = \mathsf{RLS}(P)$ will also be a zero matrix. Therefore, if for all $i, \mathbf{B}_i$ is a zero matrix, then for all $\mathbf{x}$, the minors of $\mathbf{A}' + \sum_{i\in[n]} x_i\mathbf{B}'_i$ remain the same. Therefore, we can add four parity equations together to cancel out the unknown coefficients (the equal minors), i.e., $\forall x : 4x \equiv 0 \mod 4$. We refer to Section 5.1 for how we cancel out the coefficients and other details about the attack.

We further generalize the above attack by relaxing the limitation that $\mathbf{B}_i$ of P are all zero matrices. By comparing the minors before and after the RLS, we notice that the RLS may not bring much uncertainty to the minors of $\mathbf{A}' + \sum_{i\in[n]} x_i\mathbf{B}'_i$, especially when $\mathbf{B}_{i\in[n]}$ are sparse matrices. In Section 5.2, we figure out the exact condition on which the minors of $\mathbf{A}' + \sum_{i\in[n]} x_i\mathbf{B}'_i$ remain the same for different $\mathbf{x}$, regardless of the randomness injected by the RLS. Thus, our attack is similarly applicable to all ADPs satisfying the condition.

2 Preliminaries

Let $\mathbb{Z}, \mathbb{N}^+$ be the set of integers and positive integers respectively. For $n \in \mathbb{N}^+$, we let $[n]$ denote the set $\{1, \ldots, n\}$. For $p \in \mathbb{N}^+$, We denote $\mathbb{Z}/p\mathbb{Z}$ by $\mathbb{Z}_p$ and denote the finite field of prime order p by $\mathbb{F}_p$. A vector $\mathbf{v} \in \mathbb{F}_p^n$ (represented in column form by default) is written as a bold lower-case letter and we denote its i-th element by $v_i \in \mathbb{F}_p$. A matrix $\mathbf{A} \in \mathbb{F}_p^{n\times m}$ is written as a bold capital letter and we denote the entry at position (i, j) by $(\mathbf{A})_{i,j}$. For any set of matrices $\mathbf{A}_1, \ldots, \mathbf{A}_n$ of potentially varying dimensions, let $\mathsf{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ be the block diagonal matrix with the $\mathbf{A}_i$ on the diagonal, and zeros elsewhere.

We use the usual Landau notations. A function $f(\cdot)$ is said to be negligible if $f(n) = n^{-\omega(1)}$ and we denote it by $f(n) = \mathsf{negl}(n)$. We write $D_1 \approx_C D_2$ if no computationally-bounded adversary can distinguish between D_1 and D_2 except with advantage negligible in the security parameter.

2.1 Indistinguishability Obfuscation

Definition 1 (Indistinguishability Obfuscator [BGI+01]). *A uniform PPT machine $i\mathcal{O}$ is an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:*

- *(Strong Functionality Preservation) For all security parameters $\lambda \in \mathbb{N}^+$, for all $C \in \mathcal{C}_\lambda$,*

$$\Pr_{C' \leftarrow i\mathcal{O}(\lambda, C)} [\forall x, C'(x) = C(x)] \geq 1 - \mathrm{negl}(\lambda).$$

- *For any non-uniform PPT distinguisher D, there exists a negligible function α such that the following holds: for all $\lambda \in \mathbb{N}^+$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all input x and $|C_0| = |C_1|$ (where $|C|$ denotes the size of a circuit), then*

$$|\Pr[D(i\mathcal{O}(\lambda, C_0)) = 1] - \Pr[D(i\mathcal{O}(\lambda, C_1)) = 1]| \leq \alpha(\lambda).$$

3 Affine Determinant Programs

In this section we describe a way of representing $\mathsf{L/poly}$ computations as polynomial-size ADPs [IK02, AIK04]. We start with the definitions of $\mathsf{L/poly}$ computations, Branching Programs (BPs) and ADPs, followed by the connections among them.

Definition 2 (Non-uniform Logarithmic-space Turing Machines). *A logarithmic-space Turing machine with polynomial-sized advice is a logarithmic-space Turing machine M^* (i.e. a machine using a logarithmic amount of writable memory space) as well as an infinite collection of advice strings $\{a_n\}_{n \in \mathbb{N}}$ of polynomial size (i.e. $|a_n| = O(n^c)$ for some c). (M^*, a_n) decides a language $L^* \subset \{0,1\}^*$ if*

$$\forall x \in \{0,1\}^*, M^*(x, a_{|x|}) = \chi_{L^*}(x)$$

(where $\chi_{L^*}(x)$ is the indicator function for L^*, i.e. $\chi_{L^*}(x) = 1$ if and only if $x \in L^*$). The set of languages decided by of logarithmic-space Turing machines with polynomial-sized advice is denoted by $\mathsf{L/poly}$; we refer to (M^*, a_n) as an $\mathsf{L/poly}$ machine.

Definition 3 (Branching Programs). *A branching program is defined by a directed acyclic graph $G(V, E)$, two special vertices $s, t \in V$, and a labeling function ϕ assigning to each edge in E a literal (i.e., x_i or $\overline{x}_i$) or the constant 1. Its size is defined as $|V| - 1$. Each input assignment $\mathbf{x} = (x_1, \ldots, x_n)$ naturally induces an unlabeled subgraph $G_{\mathbf{x}}$, whose edges include every $e \in E$ such that $\phi(e)$ is satisfied by $\mathbf{x}$. An accepting path on input x is a directed $s-t$ path in the graph $G_{\mathbf{x}}$. BP is said to be deterministic if for every $\mathbf{x}$, the out-degree of every vertex in $G_{\mathbf{x}}$ is at most 1. Thus, an deterministic branching program computes the function $f : \{0,1\}^n \to \{0,1\}$, such that $f(\mathbf{x}) = 1$ if and only if the number of accepting paths on $\mathbf{x}$ is 1.*

Definition 4 (Affine Determinant Programs). *An affine determinant program is parameterized by an input length n, a width ℓ, and a finite field $\mathbb{F}_p$. It is comprised of an affine function $L : \{0,1\}^n \to \mathbb{F}_p^{\ell \times \ell}$ along with an evaluation function $\mathsf{Eval} : \mathbb{F}_p \to \{0,1\}$. The affine function L is specified by an $(n+1)-$tuple of $\ell \times \ell$ matrices $L = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ over $\mathbb{F}_p$ so that $L(\mathbf{x}) := \mathbf{A} + \sum_{i \in [n]} x_i \mathbf{B}_i$.*

On input $\mathbf{x} \in \{0,1\}^n$, $\mathsf{ADP}_{L,\mathsf{Eval}}(\mathbf{x})$ *is computed as* $\mathsf{Eval}(\det(L(\mathbf{x})))$. *Typically, we use one of the following* Eval *functions.*

- $\mathsf{Eval}_{=0}(y) \stackrel{\mathsf{def}}{=} \begin{cases} 1, & y = 0 \\ 0, & y \neq 0 \end{cases}$.
- $\mathsf{Eval}_{\neq 0}(y) \stackrel{\mathsf{def}}{=} \begin{cases} 1, & y \neq 0 \\ 0, & y = 0 \end{cases}$.
- $\mathsf{Eval}_{\mathsf{parity}}(y) \stackrel{\mathsf{def}}{=} y \mod 2$.

Transformation between L/poly *Computations and Deterministic* BP*s.* Suppose we have an $s(n)$-space bounded non-uniform deterministic Turing machine, its configuration graph on an input of length n is bounded by $2^{O(s^*(n))}$, where $s^*(n) = \max\{s(n), \lceil \log(n) \rceil, \lceil \log(a(n)) \rceil\}$ and $a(n)$ is the length of the advice. Then we can construct a deterministic branching program G^n to simulate the Turing machine. G^n has a vertex for each of the configurations that is reachable from the start configuration. The edge $e_{j,k}$ is labeled by x_i if configuration j can reach configuration k in one step when $x_i = 1$. The label $\overline{x}_i$ is defined analogously. The label 1 means that configuration j can always reach configuration k in one step. G^n is acyclic as we can require the Turing machine to count the steps taken and record it on the work tape. It is easy to see that ${G^n}_{\mathbf{x}}$ has a $s - t$ path if and only if the Turing machine accepts on input $\mathbf{x}$. On the other hand, after putting description of a deterministic branching program on the advice tape, finding a $s-t$ path in the BP can be computed in log-space since the out degree of every vertex is at most 1 for any $\mathbf{x}$. Due to these facts, we can conclude that polynomial-size deterministic BPs equal to L/poly computations.

Encoding BP*s as* ADP*s.* Suppose there is a branching program of size ℓ computing a Boolean function f, where each input induces at most one accepting path[4]. We can represent the branching program as an adjacency matrix of size $(\ell+1)\times(\ell+1)$. Each element in the matrix is 0, 1 or some variable (x_i or $\overline{x}_i$). We denote the adjacency matrix by $M(\mathbf{x})$. $M(\mathbf{x})$ is 0 below the main diagonal (including main diagonal) since a branching program can be view as a DAG. Then we modify the main diagonal elements of $M(\mathbf{x})$ to -1 and delete the leftmost column and lowermost row. We denote the resulting $\ell \times \ell$ matrix by $L(\mathbf{x})$. For all $\mathbf{x} \in \{0,1\}^n$, We have $\det(L(\mathbf{x})) = f(\mathbf{x})$. Then we set $\mathbf{A} = L(\mathbf{0})$, $\mathbf{B}_i = L(\mathbf{1}_i) - \mathbf{A}$, where $\mathbf{0}$ is the input whose bits are all 0 and $\mathbf{1}_i$ is the input whose i's bit is 1 and 0 everywhere else. For all $\mathbf{x} \in \{0,1\}^n$, We have $L(\mathbf{x}) = \mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i$. This immediately gives us an ADP for the branching program. The evaluation function is $\mathsf{Eval}_{\neq 0}$. We use the following theorem to show the correctness of the encoding. For more details we refer readers to [IK97].

Theorem 1 (Imported Theorem [IK97]). *Let* $\mathbf{A}_G$ *be the* $a \times a$ *adjacency matrix of a DAG* G *(over* $GF(p)$*). For any two vertices* s, t *in* G*, let* $n^p_{s,t}$ *denote*

[4] Here, we actually define a new class of branching programs that can be seen as a generalization of the deterministic BPs whose out degree of every vertex is not limited by 1 for all $\mathbf{x}$. This new notion can be helpful when obfuscating ADPs.

the number of distinct s-t paths in G modulo p, and for any $a \times a$ matrix $\mathbf{A}$, let $\mathbf{A}_{(i,j)}$ denote the $(a-1) \times (a-1)$ matrix obtained by removing the i^{th} row and the j^{th} column from $\mathbf{A}$. Then for any two vertices s, t the following assertion hold:

$$n^p_{s,t} = \det_p(\mathbf{I} - \mathbf{A}_G)^{-1} \det_p((\mathbf{I} - \mathbf{A}_G)_{(t,s)}).$$

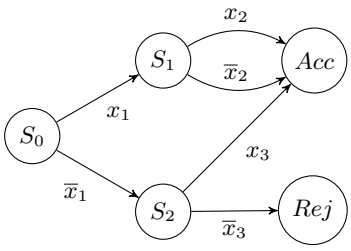
The entries in the main diagonal of $(\mathbf{I} - \mathbf{A}_G)$ are all 1s. Therefore, $\det_p(\mathbf{I} - \mathbf{A}_G)^{-1} = 1$.

Example. We give a small example for a BP/ADP for a 3-bit function that computes $x_1 \vee x_3 = 1$ (see Fig.1 (a)). First, we delete the rejection configuration and related edges. Then we apply topological sorting on the remaining 4 configurations. If there are two edges between any two configurations, we replace them by an edge labeled by "1". Now we can obtain a branching program corresponding to the Turing machine (see Fig.1 (b)). The $M(\mathbf{x}), L(\mathbf{x})$ of the branching program is

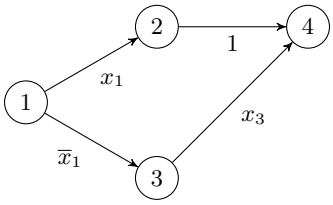
$$M(\mathbf{x}) = \begin{bmatrix} 0 & x_1 & 1-x_1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & x_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}, L(\mathbf{x}) = \begin{bmatrix} x_1 & 1-x_1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & x_3 \end{bmatrix}.$$

and the resulting ADP is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \mathbf{B}_1 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$



(a) configuration transition diagram of an L/poly Turing machine (input length is 3)



(b) a branching program

Fig. 1: A transformation between L/poly computations and BPs

Other examples can be found in, e.g., [BIJ+20, Section 4].

4 The BIJMSZ $i\mathcal{O}$ Scheme

In this section we recall the $i\mathcal{O}$ scheme proposed by Bartusek, Ishai, Jain, Ma, Sahai, and Zhandry [BIJ+20]. The scheme works by additionally applying the following four transformations in sequence to an ADP. These transformations are functionality-preserving. Readers who are familiar with the scheme can safely skip this section. Looking ahead, our attack will exploit the weakness of Transformation 1 and 2.

4.1 Transformation 1: Random Local Substitutions

The goal of Random Local Substitutions (RLS) is to inject entropy into the branching program by adding some vertices and modifying edges in a somewhat random way[5]. We denote the resulting BP by $M'(\mathbf{x})$. Specifically, we can add a vertex $v_{j,k}$ for each pair (v_j, v_k). For convenience, we only consider the 2×2 submatrices of $M'(\mathbf{x})$ with row indexed by v_j, $v_{j,k}$ and column indexed by $v_{j,k}$, v_k. Denote this matrix by $M^{'(j,k)}(\mathbf{x})$. If the edge between v_j, v_k is labeled by 1, then $M^{'(j,k)}(\mathbf{x})$ has following 4 choices (the last one is special as it is the only one which can change the label between v_j and v_k, we will analyze it separately in our attack):

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

If there is no edge between v_j, v_k, then $M^{'(j,k)}(\mathbf{x})$ has following 3 choices:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

If the edge between v_j, v_k is labeled by x_i, then $M^{'(j,k)}(\mathbf{x})$ has following 12 choices:

$$\begin{bmatrix} 0 & x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & x_i \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & x_i \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} 0 & x_i \\ 0 & \overline{x}_i \end{bmatrix}, \begin{bmatrix} 1 & x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & x_i \end{bmatrix},$$

$$\begin{bmatrix} x_i & x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} x_i & x_i \\ 0 & \overline{x}_i \end{bmatrix}, \begin{bmatrix} x_i & 0 \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} x_i & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} \overline{x}_i & x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} \overline{x}_i & x_i \\ 0 & x_i \end{bmatrix}.$$

If the edge between v_j, v_k is labeled by $\overline{x}_i$, then $M^{'(j,k)}(\mathbf{x})$ also has 12 choices, which is analogous to labeled by x_i. We can swap x_i and $\overline{x}_i$ in above matrices to obtain the 12 choices.

One can easily check that the above transformation does not change the amount of path from v_j to v_k. Namely, it is functionality-preserving[6].

[5] The transformation is actually applied to an ADP. We describe it by BP because BP is a DAG and thus can be better understood. You can understand the RLS here in this way: it decodes the input ADP back to a BP first, then it does the transformation and encodes the resulting BP as the final ADP.

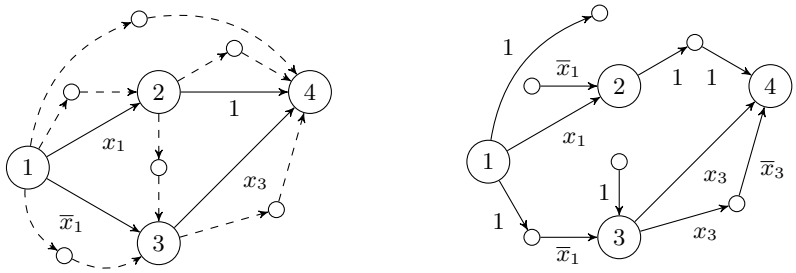
[6] There are many potential ways of applying RLS. The RLS transformation here is the candidate given in [BIJ+20].

Example. We start from the example branching program in section 3 and add a intermediate vertex for every two vertices (see Fig.2 (a)). Then we reassign the labels of the edges as described above. We show a possible result of RLS in Fig.2 (b). The ADP corresponding to the figure is

$$\mathbf{A}' = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}, \mathbf{B}'_1 = \begin{bmatrix} 0\,0\,0 & 1 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & -1 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & -1 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \\ 0\,0\,0 & 0 & 0\,0 & 0 & 0\,0 \end{bmatrix},$$

$$\mathbf{B}'_2 = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}, \mathbf{B}'_3 = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0\,0\,0\,1 & 1 \\ 0\,0\,0\,0\,0\,0\,0\,0 & -1 \end{bmatrix}.$$

(vertices are sorted in lexicographical order, i.e. $v_1, v_{1,2}, v_{1,3}, v_{1,4}, v_2, v_{2,3}, \cdots$).



(a) (b)

Fig. 2: Random Local Substitutions

4.2 Transformation 2: Small Even-Valued Noise

This transformation takes advantage of the fact that for any polynomial $g : \mathbb{Z}^n \to \mathbb{Z}$, and for any $\{e_i \in \mathbb{Z}\}_{i\in[n]}$, it holds that

$$g(x_1, x_2, \ldots, x_n) \equiv g(x_1 + 2e_1, x_2 + 2e_2, \ldots, x_n + 2e_n) \mod 2$$

Therefore, when taking an ADP as input, we can add independent random even numbers as the noise term to each entry of $\{\mathbf{A}, \{\mathbf{B}_i\}_{i\in[n]}\}$. We denote the resulting matrices by $\{\mathbf{A}+2\mathbf{E}_0, \{\mathbf{B}_i+2\mathbf{E}_i\}_{i\in[n]}\}$. The evaluation function also needs to change from $\mathsf{Eval}_{\neq 0}$ to $\mathsf{Eval}_{\mathsf{parity}}$.

The bound for the error terms and the modulus p must be set carefully to guarantee correctness and security. In particular, the noise term are relatively small compared to the modulus p (although both are super-polynomial) so that for any $y_1, ..., y_n \in \{0,1\}$:

$$\left(\det\left((\mathbf{A}+2\mathbf{E}_0)+\sum_{i\in[n]} y_i(\mathbf{B}_i+2\mathbf{E}_i)\right) \bmod p\right) \bmod 2 = \det(\mathbf{A}+\sum_{i\in[n]} y_i\mathbf{B}_i) \bmod 2$$

In other words, the noise term in an honest evaluation does not wrap around mod p.

4.3 Transformation 3: Block-Diagonal Matrices

Ideally, when obfuscating an ADP, we need to force the adversary to evaluate the program in the way we want. This goal is achieved by adding some randomness in the matrices. Only an honest evaluation can cancel out the randomness and reveal the output. Other combination of the matrices will leave the randomness intact, hiding all useful information of the origin ADP. This can be accomplished by sampling $2n$ random matrices $\{\mathbf{G}_i, \mathbf{H}_i\}_{i\in[n]}$ of determinant 1. We will append each $\mathbf{G}_i$ to $\mathbf{A}$ along the diagonal, and then append $\mathbf{H}_i - \mathbf{G}_i$ to $\mathbf{B}_i$ in the i^{th} slot along the diagonal. We denote the resulting matrices by $\left\{\mathsf{diag}(\mathbf{A}, \mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_n), \{\mathsf{diag}(\mathbf{B}_i, \mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{H}_i - \mathbf{G}_i, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{0})\}_{i\in[n]}\right\}$.

4.4 Transformation 4: AIK Re-randomization

The re-randomization step a la. [AIK04] is applied twice in the obfuscation, once after taking the second transformation and again after taking the third transformation. In both steps, we left- and right-multiply each matrices with uniformly random matrices $\mathbf{R}$, $\mathbf{S}$ respectively such that $\det(\mathbf{R}) \cdot \det(\mathbf{S}) = 1$.

To summarize, the final obfuscation is

$$\mathbf{R}'\Big(\mathsf{AddDiag}\Big(\mathbf{R}\Big(\mathsf{AddNoise}\big(\mathsf{RLS}(\mathsf{ADP})\big)\Big)\mathbf{S}\Big)\Big)\mathbf{S}'.$$

5 Our Attack

The BIJMSZ obfuscation scheme consists of three transformations along with the re-randomization step. Among the three transformations, the purpose of adding block-diagonal matrices is preventing adversary from evaluating program dishonestly (e.g. Computing $\mathbf{A} + 3\mathbf{B}_1$); adding even-valued noise is meant to

convert possibly low-rank matrices into full-rank ones. The re-randomization step is meant to hide information other than the determinant and rank of the matrices.

Therefore, after applying these two transformations and the re-randomization step, we expect that the leakage only comes from the determinant of $\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i$. Indeed, our attack is based on the following observations about the determinant.

Key Observations. The adversary can get extra information by computing $\det(\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i) \bmod 4$, namely, first computing the determinant over $\mathbb{Z}_p$, then computing the result mod 4. Note that we can ignore modulo p when analyzing our attack since we always add matrices of the ADPs together honestly, i.e. compute $\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i$ where $x_i \in \{0,1\}$. In this case, the determinant will never wrap around mod p. See Section 4.2.

The idea of computing $\det(\mathbf{A} + \sum_{i\in[n]} x_i\mathbf{B}_i) \bmod 4$ was also observed by Bartusek et al. [BIJ+20, Section 9.3], where they suggest that computing such a value is useful for extracting the parities of the noise terms. The reason that Bartusek et al. introduce the RLS transformation is precisely to prevent this attack.

However, if the RLS transformation does not inject randomness into some matrices, then we can still extract information by computing the determinant modulo 4. Indeed, we observe that the RLS candidate given in [BIJ+20, Section 8.1.1] is not guaranteed to inject randomness into every matrix. Specifically, if we have a program $\{\mathbf{A}, \mathbf{B}_1 = \mathbf{0}\}$, the program after RLS will be $\{\mathbf{A}', \mathbf{B}'_1 = \mathbf{0}\}$. Namely, when applying RLS on a zero matrix $\mathbf{B}$, it only increases the dimension of the $\mathbf{B}$ matrix, and the resulting matrix remains a zero matrix.

5.1 Base case

Running example. Consider an ADP $(\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2)$ computing $f : \{0,1\}^2 \to 1$ that is in one of the following forms:

1. $\mathbf{B}_1 = \mathbf{B}_2 = \mathbf{0}$;
2. $\mathbf{B}_1 \neq \mathbf{0}$ or $\mathbf{B}_2 \neq \mathbf{0}$.

First, apply RLS to the ADP and denote the resulting ADP by $(\mathbf{A}', \mathbf{B}'_1, \mathbf{B}'_2)$. Let $L'(\mathbf{x}) = \mathbf{A}' + \sum_i x_i\mathbf{B}'_i$. In case 1, we have $\mathbf{B}'_1 = \mathbf{B}'_2 = \mathbf{0}$, whereas $\mathbf{B}'_1 \neq \mathbf{0}$ or $\mathbf{B}'_2 \neq \mathbf{0}$ in case 2. Then applying the AddNoise operation to $(\mathbf{A}', \mathbf{B}'_1, \mathbf{B}'_2)$, hoping that the choice of ADP is masked by the operation. We let $(\mathbf{A}'' = \mathbf{A}' + 2\mathbf{E}_0, \mathbf{B}''_1 = \mathbf{B}'_1 + 2\mathbf{E}_1, \mathbf{B}''_2 = \mathbf{B}'_2 + 2\mathbf{E}_2)$ denote the resulting ADP and evaluate the ADP by computing $\det(L''(\mathbf{x}))$, where $L''(\mathbf{x}) = \mathbf{A}'' + \sum_i x_i\mathbf{B}''_i$. We omit the AddDiag operation as well as the re-randomization step since they will not change $\det(\mathbf{A}'' + \sum_i x_i\mathbf{B}''_i)$. We have

Theorem 2.

$$\begin{aligned}&\det(\mathbf{A}'' + \sum_{i\in[n]} x_i \mathbf{B}''_i)\\ \equiv &\det(L'(\mathbf{x})) + \sum_{j\in[\ell'],k\in[\ell']} (2e^{(0)}_{j,k} + \sum_i x_i 2e^{(i)}_{j,k}) \det(L'(\mathbf{x})_{(j,k)}) \mod 4\end{aligned} \tag{1}$$

where $L'(\mathbf{x})_{(j,k)}$ is a matrix obtained by deleting the j^{th} row and k^{th} column of $L'(\mathbf{x})$, $e^{(i)}_{j,k}$ is the (j,k) element of $\mathbf{E}_i$ and ℓ' is the dimension of $L'(\mathbf{x})$.

To see the correctness of the equation, first we only need to consider constant terms and linear terms of the noises. Quadratic terms or terms with higher degree will be cancelled out by modulo 4 since noises are all even numbers. Then we notice that when computing $\det(\mathbf{M} + 2\mathbf{E})$ for a matrix $\mathbf{M}$ and a noise matrix $\mathbf{E}$, the constant terms of noises are equal to $\det(\mathbf{M})$; the linear terms of noises can be divided into non-intersecting parts, each part only relevant to one entity of $\mathbf{E}$. For the (j,k) element of $\mathbf{E}$ which is denoted by $e_{j,k}$, the linear term of $e_{j,k}$ is $e_{j,k} \cdot \det(\mathbf{M}_{(j,k)})$. We can obtain Eqn. (1) by replacing $\mathbf{M}$ with $L'(\mathbf{x})$ and $\mathbf{E}$ with $2\mathbf{E}_0 + 2\sum_i x_i \mathbf{E}_i$.

To formally prove Theorem 2, we prove the following lemma.

Lemma 1. *For any $\ell \geq 2$, $\mathbf{A} \in \mathbb{Z}^{\ell\times\ell}$, and any $\mathbf{E} \in \mathbb{Z}^{\ell\times\ell}$, we have*

$$\det(\mathbf{A} + 2\mathbf{E}) = \det(\mathbf{A}) + \sum_{j\in[\ell],k\in[\ell]} 2e_{j,k} \det(\mathbf{A}_{(j,k)}) \pmod 4 \tag{2}$$

where $e_{j,k}$ is the $(j,k)^{th}$ entry of $\mathbf{E}$, $\mathbf{A}_{(j,k)} \in \mathbb{Z}^{(\ell-1)\times(\ell-1)}$ is a matrix obtained by deleting the j^{th} row and the k^{th} column of $\mathbf{A}$.

Proof. Recall the Laplace expansion for determinant: for any matrix $\mathbf{V} \in \mathbb{R}^{\ell\times\ell}$, for any $k \in [\ell]$,

$$\det(\mathbf{V}) = \sum_{j\in[\ell]} (-1)^{j+k} v_{j,k} \det(\mathbf{V}_{(j,k)}) \tag{3}$$

We prove Lemma 1 by induction. For the base case of $\ell = 2$,

$$\begin{aligned}&\det(\mathbf{A} + 2\mathbf{E})\\ &= (a_{1,1} + 2e_{1,1}) \cdot (a_{2,2} + 2e_{2,2}) - (a_{1,2} + 2e_{1,2}) \cdot (a_{2,1} + 2e_{2,1})\\ &\equiv_{(1)} (a_{1,1}a_{2,2} - a_{1,2}a_{2,1}) + 2(e_{1,1}a_{2,2} + e_{2,2}a_{1,1} - e_{1,2}a_{2,1} - e_{2,1}a_{1,2}) \pmod 4\\ &\equiv \det(\mathbf{A}) + \sum_{j\in[\ell],k\in[\ell]} (-1)^{j+k} 2e_{j,k} \det(\mathbf{A}_{(j,k)}) \pmod 4\\ &\equiv_{(2)} \det(\mathbf{A}) + \sum_{j\in[\ell],k\in[\ell]} 2e_{j,k} \det(\mathbf{A}_{(j,k)}) \pmod 4,\end{aligned} \tag{4}$$

where (1) is obtained by dropping the multiples of 4, (2) is obtained by dropping the -1 sign since $-2e = 2e \pmod 4$ for every $e \in \mathbb{Z}$ [7].

[7] For the same reason, we will ignore the sign of the minors in the rest of this paper.

For $\ell \geq 3$,

$$\begin{aligned}
&\det(\mathbf{A}+2\mathbf{E}) \\
=&\sum_{j\in[\ell]}(-1)^{j+1}(a_{j,1}+2e_{j,1})\cdot\det((\mathbf{A}+2\mathbf{E})_{(j,1)}) \\
\equiv_{(1)}&\ \det(\mathbf{A})+\sum_{j\in[\ell]}(2e_{j,1})\cdot\det((\mathbf{A})_{(j,1)})+ \\
&\sum_{j\in[\ell]}a_{j,1}\cdot\left(\sum_{i\in[\ell],i\neq j,k\in[\ell],k\neq 1}2e_{i,k}\det((\mathbf{A}_{(i,k)})_{(j,1)})\right) \pmod 4 \\
\equiv_{(2)}&\ \det(\mathbf{A})+\sum_{j\in[\ell],k\in[\ell]}2e_{j,k}\det(\mathbf{A}_{(j,k)}) \pmod 4
\end{aligned} \tag{5}$$

where (1) uses the induction hypothesis, (2) is obtained by fixing each $e_{j,k}$ and regrouping the terms of $\mathbf{A}_{(j,k)}$.

Therefore Theorem 2 holds:

$$\begin{aligned}
&\det(\mathbf{A}''+\sum_{i\in[n]}x_i\mathbf{B}_i'') \\
\equiv&\det(L'(\mathbf{x}))+\sum_{j\in[\ell'],k\in[\ell']}(2e_{j,k}^{(0)}+\sum_i x_i 2e_{j,k}^{(i)})\det(L'(\mathbf{x})_{(j,k)}) \mod 4
\end{aligned} \tag{6}$$

Let us now show how to use Theorem 2 to distinguish two programs.

Case 1 We have $L'(00)=L'(10)=L'(01)=L'(11)$ since $\mathbf{B}_1'=\mathbf{B}_2'=\mathbf{0}$. Thus, we can write $\det(L''(\mathbf{x})) \mod 4$ as:
$\det(L''(00)) \equiv \det(L'(00))+\sum_{j,k}(2e_{j,k}^{(0)})\det(L'(00)_{(j,k)}) \mod 4$
$\det(L''(10)) \equiv \det(L'(00))+\sum_{j,k}(2e_{j,k}^{(0)}+2e_{j,k}^{(1)})\det(L'(00)_{(j,k)}) \mod 4$
$\det(L''(01)) \equiv \det(L'(00))+\sum_{j,k}(2e_{j,k}^{(0)}+2e_{j,k}^{(2)})\det(L'(00)_{(j,k)}) \mod 4$
$\det(L''(11)) \equiv \det(L'(00))+\sum_{j,k}(2e_{j,k}^{(0)}+2e_{j,k}^{(1)}+2e_{j,k}^{(2)})\det(L'(00)_{(j,k)}) \mod 4$
Then we sum them all:
$\det(L''(00))+\det(L''(01))+\det(L''(10))+\det(L''(11))$
$\equiv 4\det(L'(00))+\sum_{j,k}(8e_{j,k}^{(0)}+4e_{j,k}^{(1)}+4e_{j,k}^{(2)})\det(L'(00)_{(j,k)}) \mod 4$
$\equiv 0 \mod 4$

Case 2 We do computations analogous to case 1. However, in this case, we do not have $L'(00)=L'(10)=L'(01)=L'(11)$ any more. As the result, we cannot combine the $2e_{j,k}^{(i)}\det(L'(\mathbf{x})_{(j,k)})$ terms. Therefore, when computing $\sum_{\mathbf{x}\in\{0,1\}^2}\det(L''(\mathbf{x})) \mod 4$, the result may be either 0 or 2, both with probability 1/2. As we will show in section 5.2, we can achieve $\sum_{\mathbf{x}\in\{0,1\}^2}\det(L''(\mathbf{x}))\equiv 0 \mod 4$ by setting $\mathbf{A},\mathbf{B_1},\mathbf{B_2}$ carefully even when $\mathbf{B}_1\neq\mathbf{0}\wedge\mathbf{B}_2\neq\mathbf{0}$. However, for most of ADPs, the result of the equation will be either 0 or 2, both with probability 1/2. So it is easy to find such ADPs which can be distinguished from case 1.

In conclusion, we can guess the random choice of ADP with probability at least 3/4 by computing $\sum_{\mathbf{x}\in\{0,1\}^2} \det(L''(\mathbf{x})) \mod 4$. We guess case 1 when the result is 0. Otherwise, we guess case 2.

5.2 Advanced case

In the base case we have shown that an ADP with two matrices being 0s can be distinguished from a functionally equivalent ADP with non-zero matrices at the same input bits. Such a condition is quite restricted, as it can be easily prevented by, for example, adding a dummy non-zero entry at the diagonal of each matrix. So it is natural to raise the following question:

Can we apply the attack without forcing $\mathbf{B}_1 = \mathbf{B}_2 = \mathbf{0}$*?*

The answer is yes. To see why, we observe that the attack in the base case crucially uses the fact that we can combine the $2e_{j,k}^{(i)} \det(L'(\mathbf{x})_{(j,k)})$ terms when they are equal across different inputs. Namely, for any $\mathbf{x_1}, \mathbf{x_2} \in \{0,1\}^2$, $\hat{L}'(\mathbf{x_1}) = \hat{L}'(\mathbf{x_2})$, where $\hat{\mathbf{M}}$ is the minor matrix of $\mathbf{M}^{\ell\times\ell}$, i.e.

$$\hat{\mathbf{M}} = \begin{bmatrix} \det(\mathbf{M}_{(1,1)}) & \det(\mathbf{M}_{(1,2)}) & \cdots & \det(\mathbf{M}_{(1,\ell)}) \\ \det(\mathbf{M}_{(2,1)}) & \det(\mathbf{M}_{(2,2)}) & \cdots & \det(\mathbf{M}_{(2,\ell)}) \\ \vdots & \vdots & \ddots & \vdots \\ \det(\mathbf{M}_{(\ell,1)}) & \det(\mathbf{M}_{(\ell,2)}) & \cdots & \det(\mathbf{M}_{(\ell,\ell)}) \end{bmatrix}.$$

Let us remark that instead of defining $\hat{\mathbf{M}}$ as a matrix, we can define it as any ordered set $\{\det(\mathbf{M}_{(i,j)})\}$. However, writing it as a matrix is a convenient notation.

In the base case, we assume the entire matrices of $L'(\mathbf{x}_i)$, for $i = 1, 2, 3, 4$, are equal to each other. However, for the attack to work **we only require $\hat{L}'(\mathbf{x_i})$, for $i = 1, 2, 3, 4$, to be equal to each other**. The rest of the section is devoted to analyzing the relationship between $\hat{L}'(\mathbf{x})$ and $\hat{L}(\mathbf{x})$ and figure out that to what extent the entries of $\hat{L}'(\mathbf{x})$ are unpredictable after applying RLS on $L(\mathbf{x})$.

Let us first classify the vertices of the graphs we are dealing with.

Theorem 3. *Vertices in $L'(\mathbf{x})$ can be classified into two categories: original vertices and intermediate vertices. The entries of $\hat{L}'(\mathbf{x})$ have the following cases:*

1. $\forall s, j \in [\ell+1]$ *satisfying* $s \le \ell$ *and* $j > 1$, $\hat{L}'(\mathbf{x})[v_s, v_j] = \hat{L}(\mathbf{x})[v_s, v_j]$.[8]
2. $\forall s, i, j \in [\ell+1]$ *satisfying* $s \le \ell$ *and* $i < j$, $\hat{L}'(\mathbf{x})[v_s, v_{i,j}] = \hat{L}(\mathbf{x})[v_s, v_j] \cdot L'(\mathbf{x})[v_{i,j}, v_j]$.
3. $\forall s, t, j \in [\ell+1]$ *satisfying* $s < t$ *and* $j > 1$, $\hat{L}'(\mathbf{x})[v_{s,t}, v_j] = \hat{L}(\mathbf{x})[v_s, v_j] \cdot L'(\mathbf{x})[v_s, v_{s,t}]$.

[8] Recall that when encoding a BP into an ADP, the lowermost row and the leftmost column are deleted. Thus, if the dimension of $L(\mathbf{x})$ is ℓ, the number of nodes should be $\ell + 1$.

4. $\forall s,t,i,j \in [\ell+1]$ *satisfying* $s<t$ *and* $i<j$,
   $\hat{L}'(\mathbf{x})[v_{s,t},v_{i,j}]_{(v_{s,t}\neq v_{i,j})}=\hat{L}(\mathbf{x})[v_s,v_j]\cdot L'(\mathbf{x})[v_{i,j},v_j]\cdot L'(\mathbf{x})[v_s,v_{s,t}]$.
5. $\forall i,j \in [\ell+1]$ *satisfying* $i<j$,
$$\hat{L}'(\mathbf{x})[v_{i,j},v_{i,j}]=\begin{cases}\det(L(\mathbf{x})), & L(\mathbf{x})[v_i,v_j]=0 \text{ or } L'(\mathbf{x})[v_i,v_j]=1\\ \det(L(\mathbf{x})_{(v_i,v_j)=0}), & L(\mathbf{x})[v_i,v_j]=1 \text{ and } L'(\mathbf{x})[v_i,v_j]=0\end{cases}$$

where $\mathbf{M}[v_i,v_j]$ *is the entry in the row corresponding to* v_i *(row* v_i *for short) and the column corresponding to* v_j *(column* v_j *for short) of* $\mathbf{M}$, $\mathbf{M}_{(v_i,v_j)=0}$ *is a matrix obtained by modifying the* (v_i,v_j) *entry of* $\mathbf{M}$ *to 0 and* $v_{i,j}$ *is the intermediate vertex between* v_i *and* v_j*, as we defined in section 4.1.*

Proof. We prove the theorem by showing following 4 lemmas.

Lemma 2. $\forall s,j \in [\ell+1]$ *satisfying* $s \leq \ell$ *and* $j>1, \hat{L}'(\mathbf{x})[v_s,v_j]=\hat{L}(\mathbf{x})[v_s,v_j]$.

Proof. Comparing $L'(\mathbf{x})_{(v_s,v_j)}$ with $L(\mathbf{x})_{(v_s,v_j)}$, there are mainly two kinds of differences: 1) $L'(\mathbf{x})_{(v_s,v_j)}$ have rows and columns corresponding to intermediate vertices. 2) $L'(\mathbf{x})_{(v_s,v_j)}[v_i,v_t]$ may not equal to $L(\mathbf{x})_{(v_s,v_j)}[v_i,v_t]$. To be specific, recall that if $L(\mathbf{x})[v_i,v_t]=1$, the RLS will set $L'(\mathbf{x})[v_i,v_t]=0$ with probability 1/4. Thus, if we delete the intermediate vertices as well as related edges one by one and recover the values between original vertices at the same time, we can convert $L'(\mathbf{x})_{(v_s,v_j)}$ to $L(\mathbf{x})_{(v_s,v_j)}$. To prove the lemma, we only need to prove that the determinant remains unchanged during the conversion. We use $v_{i,t}$ to denote the intermediate vertex to be deleted. There are broadly 2 cases in the conversion:

Label between original vertices deleted or unchanged. In this case, we do not need to recover the label between original vertices (namely, the label between v_i and v_t). Also, we can find row $v_{i,t}$ or column $v_{i,t}$ with only nonzero entry -1 at $(v_{i,t},v_{i,t})$. Therefore, Computing the expansion of $\det(L'(\mathbf{x})_{(v_s,v_j)})$ by row $v_{i,t}$ or column $v_{i,t}$ is equal to computing the determinant after deleting row $v_{i,t}$ and column $v_{i,t}$. See Fig.3 (a)(b).

Label between original vertices changed. This case can be transformed to the first one by adding row $v_{i,k}$ to row v_i or adding column $v_{i,k}$ to column v_k, which keeps the determinant unchanged as well as recovers the label between v_i and v_t. See Fig.3 (c).

Lemma 3. $\forall i,j \in [\ell+1]$ *satisfying* $i<j$, $\hat{L}'(\mathbf{x})[v_*,v_{i,j}]=\hat{L}(\mathbf{x})[v_*,v_j]\cdot L'(\mathbf{x})[v_{i,j},v_j]$, *where* v_* *is either an original vertex or an intermediate vertex,* $v_* \neq v_{i,j}$ *and* $v_* \neq v_1$.

Proof. We notice that row $v_{i,j}$ of $L'(\mathbf{x})_{(v_*,v_{i,j})}$ has the only possible nonzero entry at $(v_{i,j},v_j)$. We can expand $\det(L'(\mathbf{x})_{(v_*,v_{i,j})})$ by row $v_{i,j}$. The result is $L'(\mathbf{x})[v_{i,j},v_j]*\det((L'(\mathbf{x})_{(v_*,v_{i,j})})_{(v_{i,j},v_j)})$. We can rewrite $(L'(\mathbf{x})_{(v_*,v_{i,j})})_{(v_{i,j},v_j)}$ as $(L'(\mathbf{x})_{(v_*,v_j)})_{(v_{i,j},v_{i,j})}$, namely, the matrix obtained by deleting row $v_{i,j}$ and column $v_{i,j}$ of $L'(\mathbf{x})_{(v_*,v_j)}$. As we showed in the proof of Lemma 2, $\det(L'(\mathbf{x})_{(v_*,v_j)})$ equals to $\det((L'(\mathbf{x})_{(v_*,v_j)})_{(v_{i,j},v_{i,j})})$. See Fig.4.

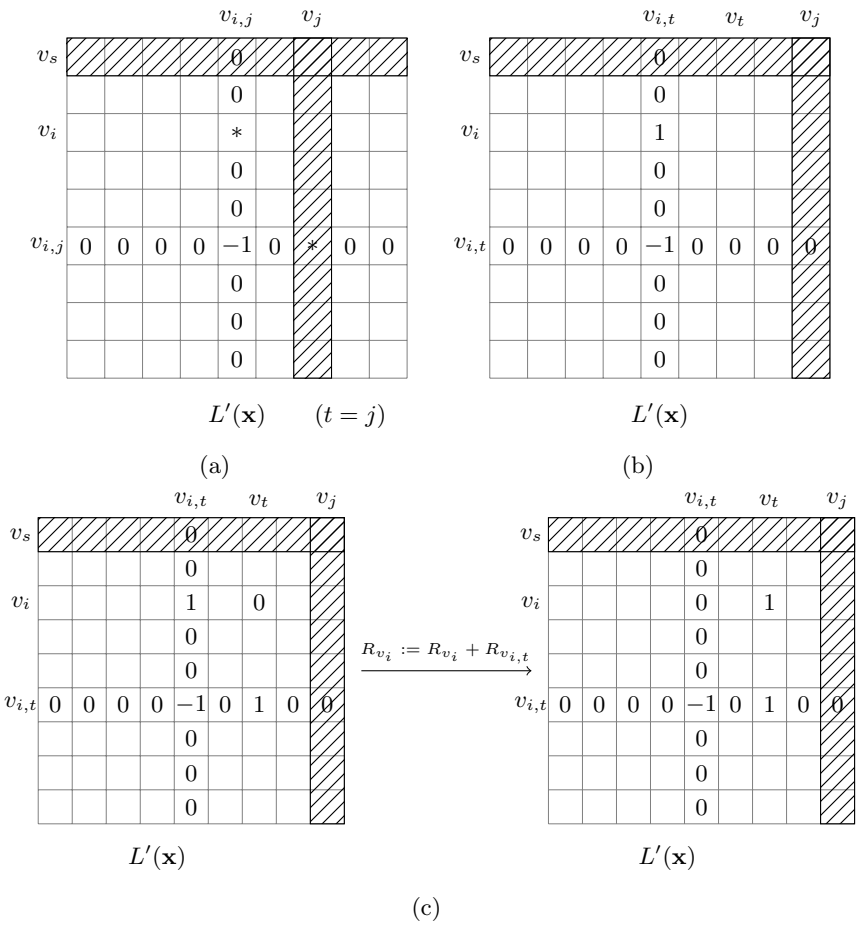


Fig. 3: Minors unrelated to any intermediate vertex

Lemma 4. $\forall s,t \in [\ell{+}1]$ *satisfying* $s<t$, $\hat{L}'(\mathbf{x})[v_{s,t},v_*]{=}\hat{L}(\mathbf{x})[v_s,v_*]{\cdot}L'(\mathbf{x})[v_s,v_{s,t}]$. *where* v_* *is either an original vertex or an intermediate vertex,* $v_* \neq v_{s,t}$ *and* $v_* \neq v_{\ell+1}$.

We omit the proof as it is analogous to Lemma 3.

Lemma 5. $\forall i,j \in [\ell{+}1]$ *satisfying* $i<j$, $\hat{L}'(\mathbf{x})[v_{i,j},v_{i,j}] = \det(L(\mathbf{x})_{(v_i,v_j)=L'(\mathbf{x})[v_i,v_j]})$.

Proof. As we showed in the proof of Lemma 2, we have a conversion that deletes all intermediate vertices and recovers labels between original vertices. However, the label between v_i and v_j is an exception. To recover the label, we need to add row $v_{i,j}$ to row v_i or add column $v_{i,j}$ to column v_j. Unfortunately, both row $v_{i,j}$ and column $v_{i,j}$ are deleted in $L'(\mathbf{x})_{(v_{i,j},v_{i,j})}$. As the result, there's no way that we can recover the label. Therefore, after the conversion, we will obtain a

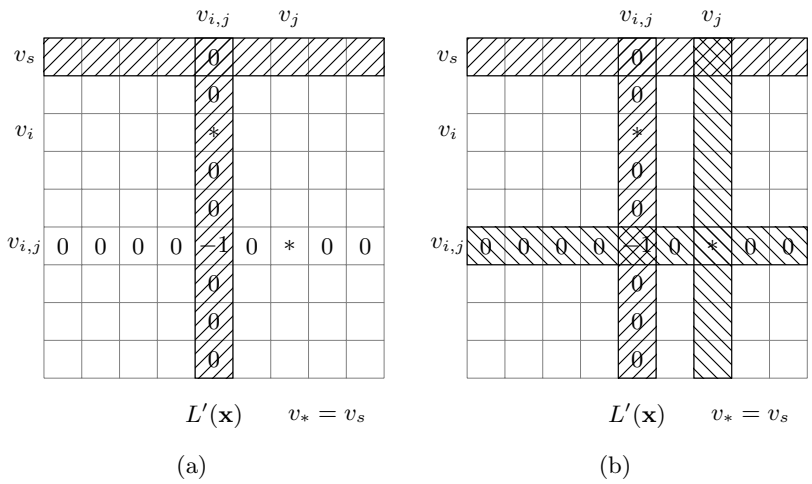


Fig. 4: Minor related to some intermediate vertex

matrix whose (v_i, v_j) entry may be the only different entry compared with $L(\mathbf{x})$. To be specific, if $L(\mathbf{x})[v_i, v_j] = 1$ and $L'(\mathbf{x})[v_i, v_j] = 0$, the (v_i, v_j) entry of the resulting matrix is 0.

This completes the proof of Theorem 3.

With Theorem 3 we can find the necessary and sufficient condition for $\hat{L}'(\mathbf{x_1}) = \hat{L}'(\mathbf{x_2})$, where $\mathbf{x_1}, \mathbf{x_2}$ are two different inputs. In fact, we have

Theorem 4. *For $L(\mathbf{x_1})$ and $L(\mathbf{x_2})$ satisfying following conditions, we can conclude that $\hat{L}'(\mathbf{x_1}) = \hat{L}'(\mathbf{x_2})$ regardless of the randomness injected by the* RLS*:*

1. $\hat{L}(\mathbf{x_1}) = \hat{L}(\mathbf{x_2})$.
2. $\forall i, j \in [\ell+1]$ *satisfying* $i \leq \ell$ *and* $j > 1$ *and* $L(\mathbf{x_1})[v_i, v_j] \neq L(\mathbf{x_2})[v_i, v_j]$*, the entries in the i^{th} row and j^{th} column of $\hat{L}(\mathbf{x_1})$ are all 0s.*
3. $\forall i, j \in [\ell+1]$ *satisfying* $i < j$*,*

$$\det(L(\mathbf{x_1})_{(v_i,v_j)=0}) = \det(L(\mathbf{x_1})) = \det(L(\mathbf{x_2})) = \det(L(\mathbf{x_2})_{(v_i,v_j)=0}).$$

Proof. We will analyse these three conditions one by one.

First, we require that $\hat{L}(\mathbf{x_1}) = \hat{L}(\mathbf{x_2})$. The reason is that for any pair of original vertices v_i, v_j, $\hat{L}'(\mathbf{x})[v_i, v_j] = \hat{L}(\mathbf{x})[v_i, v_j]$. (See Theorem 3, the first case.)

Then, we compute $\Delta L(\mathbf{x_1}, \mathbf{x_2}) = L(\mathbf{x_1}) - L(\mathbf{x_2})$. The nonzero entries in $\Delta L(\mathbf{x_1}, \mathbf{x_2})$ represent the differences between $L(\mathbf{x_1})$ and $L(\mathbf{x_2})$. If $\Delta L(\mathbf{x_1}, \mathbf{x_2})[v_i, v_j] \neq 0$, the difference may be propagated into $(v_i, v_{i,j})$, $(v_{i,j}, v_j)$ and (v_i, v_j) of $\Delta L'(\mathbf{x_1}, \mathbf{x_2})$ after applying RLS (See Fig.5). We notice that row $v_{i,j}$ entries of $\hat{L}'(x)$ depend on $L'(\mathbf{x})[v_i, v_{i,j}]$ (marked in north east lines) and column $v_{i,j}$ entries of $\hat{L}'(x)$ depend on $L'(\mathbf{x})[v_{i,j}, v_j]$ (marked in north west lines). So the difference may cause entries in row $v_{i,j}$ or column $v_{i,j}$ (except $(v_{i,j}, v_{i,j})$, which we will discuss later)

of $\Delta\hat{L}'(\mathbf{x_1}, \mathbf{x_2})$ to be nonzero. Fortunately, these entries of $\hat{L}'(\mathbf{x})$ also depend on row v_i entries and column v_j entries of $\hat{L}(\mathbf{x})$. To be specific, if row v_i entries and column v_j entries of $\hat{L}(\mathbf{x})$ are all zero, row $v_{i,j}$ entries and column $v_{i,j}$ entries of $\hat{L}'(x)$ are all zero (except $(v_{i,j}, v_{i,j})$), whatever the entries of $L'(\mathbf{x})$ are. (See Theorem 3, the second to the forth case.) Therefore, we further require that for any nonzero entry of $\Delta L(\mathbf{x_1}, \mathbf{x_2})[v_i, v_j]$, row v_i entries and column v_j entries of $\hat{L}(\mathbf{x_1})$ are all zero.

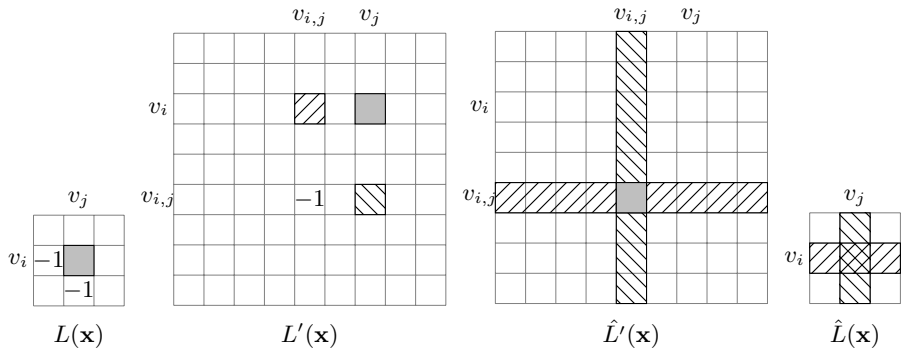


Fig. 5: The relationship among (minor) matrices before and after the RLS

Finally, we analyze the condition for $\Delta\hat{L}'(\mathbf{x_1}, \mathbf{x_2})[v_{i,j}, v_{i,j}] = 0$. If $L(\mathbf{x_1})[v_i, v_j] = L(\mathbf{x_2})[v_i, v_j] = 0$, we have $\hat{L}'(\mathbf{x_1})[v_{i,j}, v_{i,j}] = \det(L(\mathbf{x_1}))$, $\hat{L}'(\mathbf{x_2})[v_{i,j}, v_{i,j}] = \det(L(\mathbf{x_2}))$. Therefore, we require $\det(L(\mathbf{x_1})) = \det(L(\mathbf{x_2}))$. If $L(\mathbf{x_1})[v_i, v_j] = L(\mathbf{x_2})[v_i, v_j] = 1$, with probability 1/4, we have $\hat{L}'(\mathbf{x_1})[v_{i,j}, v_{i,j}] = \det(L(\mathbf{x_1})_{(v_i,v_j)=0})$, $\hat{L}'(\mathbf{x_2})[v_{i,j}, v_{i,j}] = \det(L(\mathbf{x_2})_{(v_i,v_j)=0})$. We further require $\det(L(\mathbf{x_1})_{(v_i,v_j)=0}) = \det(L(\mathbf{x_2})_{(v_i,v_j)=0})$. (See Theorem 3, the fifth case.) If $L(\mathbf{x_1})[v_i, v_j] = 0, L(\mathbf{x_2})[v_i, v_j] = 1$, we require $\det(L(\mathbf{x_1})) = \det(L(\mathbf{x_2})_{(v_i,v_j)=0})$. If $L(\mathbf{x_1})[v_i, v_j] = 1, L(\mathbf{x_2})[v_i, v_j] = 0$, we require $\det(L(\mathbf{x_1})_{(v_i,v_j)=0}) = \det(L(\mathbf{x_2}))$.

This completes the proof of Theorem 4.

Running Example. Let us start with defining a family of ADPs to which our attack could apply.[9] Since we only need 4 inputs to carry out our attack, we can fix the other $n-2$ input bits. w.l.o.g. we assume that x_1 and x_2 are unfixed. $\forall$ ADP P in the family, there exists an assignment $\mathbf{a} \in \{0,1\}^{n-2}$ to the values of

[9] The family of ADPs here is only a subset of all ADPs our attack could apply.

$x_3x_4\ldots x_n$ respect to P, s.t. the program matches the following pattern:

$$L(x_1x_2\mathbf{a}) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 & 0 \\ -1 & * & \cdots & * & * & 0 \\ 0 & -1 & \cdots & * & * & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & * & 0 \\ 0 & 0 & \cdots & 0 & -1 & 0 \end{bmatrix}.$$

where $*$ is a wildcard and represents one element in $\{0, 1, x_1, \overline{x_1}, x_2, \overline{x_2}\}$.

Next, we will show that

$$\sum_{\mathbf{x}\in\{00\mathbf{a},10\mathbf{a},01\mathbf{a},11\mathbf{a}\}} \det(L''(\mathbf{x})) \equiv 0 \mod 4.$$

First, we have

$$\hat{L}(x_1x_2\mathbf{a}) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

Namely, $\det(L(x_1x_2\mathbf{a})_{(i,j)}) = \begin{cases} 1, & (i = 1 \vee i = \ell) \wedge j = \ell \\ 0, & otherwise \end{cases}$. To see why, notice that the rightmost column of $\hat{L}(x_1x_2\mathbf{a})$ is an all-zero column, thus $\det(L(x_1x_2\mathbf{a})_{(i,j)}) = 0$ for $j < \ell$. Moreover, we can add the topmost and lowermost rows of $\hat{L}(x_1x_2\mathbf{a})$ together to obtain an all-zero row, thus $\det(L(x_1x_2\mathbf{a})_{(i,j)}) = 0$ for $1 < i < \ell$. It is easy to check that $\det(L(x_1x_2\mathbf{a})_{(1,\ell)}) = \det(L(x_1x_2\mathbf{a})_{(\ell,\ell)}) = 1$.

Then, nonzero entries in $\Delta L(\mathbf{x_1}\mathbf{a}, \mathbf{x_2}\mathbf{a})$ depend on $x_1, \overline{x_1}, x_2$ and $\overline{x_2}$ entries in $L(x_1x_2\mathbf{a})$, where $\mathbf{x_1}, \mathbf{x_2} \in \{0,1\}^2 \wedge \mathbf{x_1} \neq \mathbf{x_2}$. These entries are marked by $*$ in the matrix above. Therefore, we hope that entries in 2^{nd}-$(\ell-1)^{th}$ rows, 2^{nd}-$(\ell-1)^{th}$ columns of $\hat{L}(\mathbf{x} \in \{00\mathbf{a}, 10\mathbf{a}, 01\mathbf{a}, 11\mathbf{a}\})$ are all zero, which is exactly the case.

Finally, only entries marked by $*$ may be modified from 1 to 0 after RLS. Since entries in the rightmost column of $L(x_1x_2\mathbf{a})$ are always all zero, we can conclude that $\det(L(x_1x_2\mathbf{a})) = \det(L(x_1x_2\mathbf{a})_{(i,j)=0}) = 0$, where $i, j \in [\ell]$ and $L(x_1x_2\mathbf{a})[i,j] = *$ according to the above matrix.

We also give a concrete example:

$$L(\mathbf{x}) = \begin{bmatrix} 0 & \overline{x_3} & 0 & x_3 & 0 \\ -1 & 0 & x_4 & 0 & \overline{x_4} \\ 0 & -1 & \overline{x_1} & 0 & 0 \\ 0 & 0 & -1 & x_2 & 0 \\ 0 & 0 & 0 & -1 & \overline{x_3} \end{bmatrix}, L(x_1x_2\mathbf{a})_{\mathbf{a}=11} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & \overline{x_1} & 0 & 0 \\ 0 & 0 & -1 & x_2 & 0 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}.$$

$L(\mathbf{x})$ computes $\overline{x_1} \wedge x_2 \wedge \overline{x_3}$, i.e. the output depends on x_1 and x_2. Therefore, in advanced case attack, we don't require the output bit to ignore some input bits,

unlike in the base case attack.
Let us remark that the successful condition of our attack can be further relaxed. For example, if $\det(L(01)_{(v_2,v_3)=0})$ ((v_2, v_3) corresponding to the $(2, 2)$ entry of $L(\mathbf{x})$) did not equal to $\det(L(00))$, then $\hat{L}'(01)[v_{2,3}, v_{2,3}] = \hat{L}'(00)[v_{2,3}, v_{2,3}]$ still holds with probability $3/4$. As a result, the advantage that we can distinguish the ADP in the example from another functionally equivalent ADP after obfuscation will decrease by a factor of $3/4$, which is still noticeable.

5.3 The scope of the attack

In the end let us discuss

What kind of programs does the attack apply to?

We are afraid that we cannot give an exact and succinct answer to this question. The reason is when analysing ADPs, we focus on constraints on determinants and minors. However, the connection between these constraints and functionality is unclear. Moreover, with constraints on 4 inputs (we only need 4 inputs to apply the attack), it is difficult to figure out what the whole function looks like.

Therefore, we choose to describe the necessary condition and the sufficient condition separately. The necessary condition of the attack is to find 4 inputs $\mathbf{x}_1 = \mathbf{a}0\mathbf{b}0\mathbf{c}, \mathbf{x}_2 = \mathbf{a}1\mathbf{b}0\mathbf{c}, \mathbf{x}_3 = \mathbf{a}0\mathbf{b}1\mathbf{c}, \mathbf{x}_4 = \mathbf{a}1\mathbf{b}1\mathbf{c}$ s.t. the program outputs the same value on these inputs where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are some fixed strings of arbitrary length. The sufficient condition of the attack is that for the 4 inputs mentioned above, we always have $\hat{L}'(\mathbf{x}_1) = \hat{L}'(\mathbf{x}_2) = \hat{L}'(\mathbf{x}_3) = \hat{L}'(\mathbf{x}_4)$ regardless of the randomness injected by the RLS. This condition is satisfiable when the plaintext ADP satisfies the conditions in Theorem 4.

We also notice that the attack can be further generalized. Recall that in the above attack, we require $\hat{L}'(00) = \hat{L}'(10) = \hat{L}'(01) = \hat{L}'(11)$. But why we need the equality of these four minor matrices? When looking back to section 5.1, on the high level, we can write the idea of our attack as

$$\begin{bmatrix} e_{j,k}^{(0)} & e_{j,k}^{(1)} & e_{j,k}^{(2)} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \det(L'(00)_{(j,k)}) \\ \det(L'(10)_{(j,k)}) \\ \det(L'(01)_{(j,k)}) \\ \det(L'(11)_{(j,k)}) \end{bmatrix} = 0 \mod 2.$$

To make the equation hold regardless of the choice of $e_{j,k}^{(0)}, e_{j,k}^{(1)}$ and $e_{j,k}^{(2)}$, the equality of these four minor matrices is necessary. However, if we have $2^b (b > 2)$ inputs, we will not require the equality of minor matrices. Take $b = 3$ as an

example, the idea of our attack can be written as:

$$\begin{bmatrix} e_{j,k}^{(0)} & e_{j,k}^{(1)} & e_{j,k}^{(2)} & e_{j,k}^{(3)} \end{bmatrix} \begin{bmatrix} 1&1&1&1&1&1&1&1 \\ 0&1&0&0&1&1&0&1 \\ 0&0&1&0&1&0&1&1 \\ 0&0&0&1&0&1&1&1 \end{bmatrix} \begin{bmatrix} \det(L'(000)_{(j,k)}) \\ \det(L'(100)_{(j,k)}) \\ \det(L'(010)_{(j,k)}) \\ \det(L'(001)_{(j,k)}) \\ \det(L'(110)_{(j,k)}) \\ \det(L'(101)_{(j,k)}) \\ \det(L'(011)_{(j,k)}) \\ \det(L'(111)_{(j,k)}) \end{bmatrix} = 0 \mod 2.$$

Therefore, for $\hat{L}'(\mathbf{x} \in \{0,1\}^3)$ satisfying following four conditions:

1. $\hat{L}'(000)+\hat{L}'(100)+\hat{L}'(010)+\hat{L}'(001)+\hat{L}'(110)+\hat{L}'(101)+\hat{L}'(011)+\hat{L}'(111) = \mathbf{0} \mod 2$
2. $\hat{L}'(100) + \hat{L}'(110) + \hat{L}'(101) + \hat{L}'(111) = \mathbf{0} \mod 2$
3. $\hat{L}'(010) + \hat{L}'(110) + \hat{L}'(011) + \hat{L}'(111) = \mathbf{0} \mod 2$
4. $\hat{L}'(001) + \hat{L}'(101) + \hat{L}'(011) + \hat{L}'(111) = \mathbf{0} \mod 2$

we have

$$\sum_{\mathbf{x}\in\{0,1\}^3} \det(L''(\mathbf{x})) \equiv 0 \mod 4.$$

To conclude, if we cannot find four inputs satisfying the conditions in Theorem 4, it is still possible to find eight or more inputs that are capable of applying the "mod 4" attack.

6 A Plausible Fix and Further Discussions

In this section, we describe a possible approach of preventing our attack. Intuitively, the reason for the attack to work is that the RLS transformation does not inject enough randomness into the original ADP. To be specific, if the edge between v_j and v_k is labeled 0 or 1 before RLS, the edges among v_j, v_k and $v_{j,k}$ are never labelled x_i or $\overline{x}_i$ after RLS.

Therefore, a natural way of fixing the attack is to get around this limitation. If the edge between v_j, v_k is labeled by 1, then $M'^{(j,k)}(\mathbf{x})$ has the following extra choices:

$$\begin{bmatrix} x_i & 1 \\ 0 & \overline{x}_i \end{bmatrix}, \begin{bmatrix} \overline{x}_i & 1 \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} x_i & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} \overline{x}_i & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & \overline{x}_i \end{bmatrix}.$$

If there is no edge between v_j, v_k, then $M'^{(j,k)}(\mathbf{x})$ has following extra choices:

$$\begin{bmatrix} x_i & 0 \\ 0 & \overline{x}_i \end{bmatrix}, \begin{bmatrix} \overline{x}_i & 0 \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} x_i & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & x_i \end{bmatrix}, \begin{bmatrix} \overline{x}_i & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & \overline{x}_i \end{bmatrix}.$$

We use the example in section 4.1 to show the revision of the RLS (see Fig.6, changes compared with Fig.2 (b) are marked in red color).

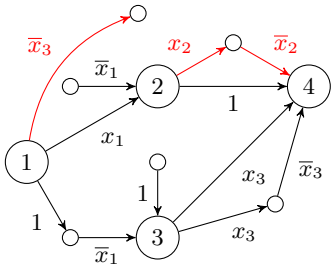


Fig. 6: The revision of the RLS

With the revision we can defend against the base case attack since it effectively makes the B matrices non-zero. But how about the advanced case? Suppose that the label between $v_{j,\ell}$ and v_ℓ is x_i or $\overline{x}_i$ after the RLS (this is always possible in the revision of the RLS). As the result, $\forall \mathbf{x}, \mathbf{y}$ satisfying $x_i \neq y_i$, $L'(\mathbf{x})[v_{j,\ell}, v_\ell] \neq L'(\mathbf{y})[v_{j,\ell}, v_\ell]$. Recall that $\hat{L}'(\mathbf{x})[v_1, v_{j,\ell}]{=}\hat{L}(\mathbf{x})[v_1, v_\ell] \cdot L'(\mathbf{x})[v_{j,\ell}, v_\ell]$. In addition, we always have $\hat{L}(\mathbf{x})[v_1, v_\ell] = 1$. We can conclude that $\hat{L}'(\mathbf{x})[v_1, v_{j,\ell}] \neq \hat{L}'(\mathbf{y})[v_1, v_{j,\ell}]$. Namely, the revision can prevent the equality of the minors and thus defend against the advanced case attack.

We also notice the necessity of setting up connection between security parameter λ and the RLS transformation. The amount of randomness of the RLS introduced in [BIJ+20] only depends on the matrix size of ADP. Even for the revision of the RLS we mentioned above (as it is), the amount of randomness only depends on the input length and the matrix dimension. Therefore, for programs with very small input lengths and matrix dimensions, the adversary can guess the output of RLS correctly with some probability that is independent of λ, in which situation the adversary could break the $i\mathcal{O}$ scheme with non-negligible probability. A simple way of preventing this attack is applying RLS iteratively for λ times. Adding more intermediate vertices is another possible solution.

Let us remark that the revision of RLS we provide merely prevents the attack we describe in this paper, it should not be viewed as a candidate with enough confidence. We cannot even ensure that with the above revision the $i\mathcal{O}$ scheme can be secure against all "modulo 4 attacks". We leave it as future work to give an RLS candidate with concrete parameters in some restricted adversarial model that is provably secure against known attacks. For example, it will be interesting to provide a candidate with provable security against all "modulo 4 attacks".

References

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, EUROCRYPT 2019, Part I, volume 11476 of LNCS, pages 191–225. Springer, Heidelberg, May 2019.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In 45th FOCS, pages 166–175. IEEE Computer Society Press, October 2004.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, CRYPTO 2015, Part I, volume 9215 of LNCS, pages 308–326. Springer, Heidelberg, August 2015.

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, CRYPTO 2019, Part III, volume 11694 of LNCS, pages 284–332. Springer, Heidelberg, August 2019.

[AP20] Shweta Agrawal and Alice Pellet-Mary. Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In Anne Canteaut and Yuval Ishai, editors, EUROCRYPT 2020, Part I, volume 12105 of LNCS, pages 110–140. Springer, Heidelberg, May 2020.

[BDGM20a] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Guilio Malavolta. Candidate io from homomorphic encryption schemes. In EUROCRYPT, 2020.

[BDGM20b] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Guilio Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. Cryptology ePrint Archive, Report 2020/1024, 2020. https://eprint.iacr.org/2020/1024.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, CRYPTO 2001, volume 2139 of LNCS, pages 1–18. Springer, Heidelberg, August 2001.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, Advances in Cryptology – EUROCRYPT 2014, volume 8441 of Lecture Notes in Computer Science, pages 221–238, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[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, ITCS 2020, volume 151, pages 82:1–82:39. LIPIcs, January 2020.

[BLMZ19] James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. New techniques for obfuscating conjunctions. In Vincent Rijmen and Yuval Ishai, editors, EUROCRYPT 2019, Part III, LNCS, pages 636–666. Springer, Heidelberg, May 2019.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, 56th FOCS, pages 1480–1498. IEEE Computer Society Press, October 2015.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, TCC 2014: 11th Theory of Cryptography Conference, volume 8349 of Lecture Notes in Computer Science, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, 56th FOCS, pages 171–190. IEEE Computer Society Press, October 2015.

[CGH17] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, EUROCRYPT 2017, Part III, volume 10212 of LNCS, pages 278–307. Springer, Heidelberg, April / May 2017.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, Advances in Cryptology – EUROCRYPT 2015, Part I, volume 9056 of Lecture Notes in Computer Science, pages 3–12, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In STOC, 2016.

[CLT13] Jean-Sébastien Coron, Tancrè de Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, CRYPTO 2013, Part I, volume 8042 of LNCS, pages 476–493. Springer, Heidelberg, August 2013.

[DQV+21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee and Daniel Wichs. Succinct LWE Sampling, Random Polynomials, and Obfuscation. In Kobbi Nissim and Brent Waters, editors, TCC 2021, volume 13043 of LNCS, pages 256–287. Springer, Raleigh, NC, USA, November 8-11, 2021.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 1–17. Springer, Heidelberg, May 2013.

[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 54th FOCS, pages 40–49. IEEE Computer Society Press, October 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, TCC 2015, Part II, volume 9015 of LNCS, pages 498–527. Springer, Heidelberg, March 2015.

[GJLS20] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. Cryptology ePrint Archive, Report 2020/764, 2020. https://eprint.iacr.org/2020/764.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2021. ACM, 2021.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In Marc Fischlin and Jean-Sébatisen Coron, editors, Advances in Cryptology – EUROCRYPT 2016, Part I, volume 9665 of Lecture Notes in Computer Science, pages 537–565, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[HJL21] Sam Hopkins, Aayush Jain and Huijia Lin. Counterexamples to New Circular Security Assumptions Underlying iO. In CRYPTO, 2021.

[IK97] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In Proc. of ISTCS '97, pp. 174-183, 1997.

[IK02] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Proc. 29th ICALP, pp. 244–256, 2002.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa R to build iO. In Yuval Ishai and Vincent Rijmen, editors, EUROCRYPT 2019, Part I, volume 11476 of LNCS, pages 251–281. Springer, Heidelberg, May 2019.

[JLS21a] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2021. ACM, 2021.

[JLS21b] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability Obfuscation from LPN over F_p, DLIN, and PRGs in NCˆ0 Cryptology ePrint Archive, Report 2021/1334, 2021. https://eprint.iacr.org/2021/1334.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In STOC, 2015.

[Lin16] Huijia Lin. Indistinguishability Obfuscation from Constant-Degree Graded Encoding Schemes. In EUROCRYPT 2016.

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, CRYPTO 2017, Part I, volume 10401 of LNCS, pages 599–629. Springer, Heidelberg, August 2017.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, CRYPTO 2017, Part I, volume 10401 of LNCS, pages 630–660. Springer, Heidelberg, August 2017.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, 57th FOCS, pages 11–20. IEEE Computer Society Press, October 2016.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In Matthew Robshaw and Jonathan Katz, editors, Advances in Cryptology – CRYPTO 2016, Part II, volume 9815 of Lecture Notes in Computer Science, pages 629–658, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, STOC, pages 475–484. ACM, 2014.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious lwe sampling. In EUROCRYPT, 2021.