Affine Determinant Programs: A Framework for Obfuscation and Witness Encryption

James Bartusek[*] Yuval Ishai[†] Aayush Jain[‡] Fermi Ma[§] Amit Sahai[‡]
Mark Zhandry[¶]

July 16, 2020

Abstract

An *affine determinant program* $\mathsf{ADP} : \{0,1\}^n \to \{0,1\}$ is specified by a tuple $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ of square matrices over $\mathbb{F}_q$ and a function $\mathsf{Eval} : \mathbb{F}_q \to \{0,1\}$, and evaluated on $\mathbf{x} \in \{0,1\}^n$ by computing $\mathsf{Eval}(\det(\mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i))$.

In this work, we suggest ADPs as a new framework for building general-purpose obfuscation and witness encryption. We provide evidence to suggest that constructions following our ADP-based framework may one day yield secure, practically feasible obfuscation.

As a proof-of-concept, we give a candidate ADP-based construction of indistinguishability obfuscation ($i\mathcal{O}$) for all circuits along with a simple witness encryption candidate. We provide cryptanalysis demonstrating that our schemes resist several potential attacks, and leave further cryptanalysis to future work. Lastly, we explore practically feasible applications of our witness encryption candidate, such as public-key encryption with near-optimal key generation.

[*] UC Berkeley. Email: `bartusek.james@gmail.com`.

[†] Technion. Email: `yuvali@cs.technion.ac.il`.

[‡] UCLA. Email: `{aayushjain,sahai}@cs.ucla.edu`.

[§] Princeton and NTT Research. Email: `fermima@alum.mit.edu`.

[¶] Princeton and NTT Research. Email: `mzhandry@princeton.edu`.

Contents

1 Introduction

Program obfuscation "scrambles" a program, preserving functionality while hiding implementation details. A theoretical study of program obfuscation was initiated by Barak et al. [BGI+01], who demonstrated that the natural notion of virtual black-box (VBB) obfuscation is impossible to achieve for all circuits. This initially led to skepticism that general cryptographically useful program obfuscation might not be realizable.

The situation changed several years ago when Garg et al. [GGH+13b] gave the first candidate construction of *indistinguishability obfuscation* — a weaker notion not subject to the impossibility — and, along with subsequent work [SW14], showed how to use this weaker notion in a multitude of cryptographically useful ways. In fact, today indistinguishability obfuscation (iO) is widely regarded as "crypto complete," capable of achieving essentially any cryptographic task.[1]

Initially all candidate obfuscators required cryptographic multilinear maps [GGH13a, CLT13, GGH15], a powerful new mathematical tool. Very recently, two new lines of work have emerged on constructing obfuscation without multilinear maps. The first line of work builds upon the connection between obfuscation and functional encryption (see [AJL+19] and the references therein); the second builds on a number of new mathematical methods [GJK18]. While these new directions expand the set of methodologies for achieving general-purpose obfuscation, the set of viable approaches for building general-purpose obfuscation remains extremely limited.

We view the situation as being reminiscent of the early days of public key cryptography, which also required new mathematical tools for the time. Some, such as Diffie-Hellman and RSA, have withstood the test of time, whereas others, such as the Merkle–Hellman knapsack cryptosystem, were ultimately found to be insecure. But even the failures yielded interesting insights, such as new cryptanalysis techniques. Therefore, given the importance of obfuscation, we believe it is crucial to investigate many potential lines of inquiry for realizing obfuscation. Furthermore, we believe it is particularly important to emphasize *simplicity* in our search for new approaches, since even successful cryptanalyses of simple proposals are likely to inform the scientific quest for secure and efficient obfuscation. (This perspective is part of a larger research agenda exploring new sources of hardness in cryptography [Sah19].)

This Work. In this work, we propose a new framework for building obfuscation through *affine determinant programs* (ADP). An ADP consists of a tuple of square matrices $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ over $\mathbb{F}_p$, where evaluation on an input $\mathbf{x} \in \{0,1\}^n$ involves computing the affine combination $\mathbf{A} + \sum_{i \in [n]} x_i \mathbf{B}_i$ and taking the determinant. While ADPs have appeared before in the literature (e.g. to build randomized encodings [IK02]), to the best of our knowledge no prior work has considered their application to general obfuscation. We believe ADPs are a promising route toward obfuscation for several reasons:

- ADPs are conceptually very simple.
- We can interpret the recent work of Bartusek et al. [BLMZ19] as an ADP for conjunction obfuscation, with provable security under the LPN assumption. Thus, ADPs certainly provide security for obfuscation in some situations.
- As we will show, ADPs may offer a route toward implementable applications.

[1] To be slightly more precise, building cryptography from iO also requires one-way functions.

Therefore, we initiate the study of ADPs for obfuscation, giving the following results:

- **A New Framework for Building Obfuscation.** We suggest ADPs as a route toward building obfuscation without multilinear maps. To facilitate this approach, we provide a number of techniques for protecting ADPs against attacks, and study the security of these techniques by developing new cryptanalysis techniques.
- **A Candidate Witness Encryption Scheme.** We examine a special case of obfuscation, called witness encryption, and give a very simple candidate construction. As an application, we use our scheme to build a public key encryption scheme with low-complexity key generation. Our approach yields for the first time *implementable* applications of witness encryption.
- **Towards Candidate iO Constructions.** We then turn to the much more difficult problem of constructing full indistinguishability obfuscation. We develop some initial ideas, including cryptanalytic methods, leading to an initial candidate construction.

1.1 Technical Overview

One of the main approaches to general-purpose program obfuscation, pioneered in [GGH+13b], can be seen as following a particular recipe:

1. Construct an algebraic randomized encoding for the input program, which can be seen as offering a sort of "one-time" security.
2. Place the algebraic encoding "in the exponent" of a multilinear map, using the multilinear map operations to evaluate the encoding.

In most of the literature, the randomized encoding used is a matrix branching program, which can be obtained using Barrington's theorem and Kilian re-randomization. Another common example due to [Zim15] converts a circuit into an algebraic circuit.

Affine Determinant Programs. In this work, we consider another variant of randomized encodings from the literature (though not yet considered in the *obfuscation* literature) which we call *Affine Determinant Programs* (ADPs). Here, the program consists of a matrix M whose entries are affine functions mapping inputs x to a field $\mathbb{F}_p$. We will denote by $M(x)$ the entry-wise evaluation of M on input x. There is also a fixed evaluation function $\mathsf{Eval} : \mathbb{F} \to \{0,1\}$. To evaluate the program on input x, we simply compute $\mathsf{Eval}(\det(M(x))$.

We note that, just like Barrington's theorem or the algebraic circuit approach to constructing algebraic randomized encodings, ADPs can also plausibly be used in conjunction with multilinear maps (and appropriate re-randomization) to yield secure obfuscation[2]. However, the central question we consider in this work is:

Can we obfuscate with ADP*s in the clear, without using multilinear maps?*

[2] There is a potential issue that the standard multilinear map interface does not allow for efficiently computing determinants over encoded values. However, the first two multilinear map candidates [GGH13a, CLT13] *do* allow for determinant computations.

Witness Encryption – An Initial Idea. As a starting point, we consider the easier task of constructing a secure *witness encryption* [GGSW13] scheme. In witness encryption, a ciphertext c is encrypted with respect to an instance x of some NP language L. The ciphertext may be decrypted using any witness π that attests that $x \in L$. Security stipulates that, for any false instance $x \notin L$, encryptions relative to x completely hide the message from computationally bounded adversaries. Note that the instance x is considered public.

We build a new framework for witness encryption for the NP-complete subset-sum problem, where instances x are vector/scalar pairs $(v, t) \in \mathbb{Z}^n \times \mathbb{Z}$, and witnesses are 0/1 vectors $w \in \{0, 1\}^n$ such that $v \cdot w = t$. Under standard NP reductions, our construction extends to any NP language.

Testing the validity of a subset-sum witness can be trivially expressed as an ADP of dimension one: $M_{v,t}(w) = -t + v \cdot w$ is an affine function that is 0 if and only if w is a witness for (v, t). We can introduce randomization and extend this to a matrix program by choosing a random $R \in \mathbb{F}^{k \times k}$ and setting $M_{v,t}(w) = (-t + v \cdot w)R$. Then, with overwhelming probability over the choice of R, for $w \in \{0, 1\}^n$, $\det(M_{v,t}(w)) = 0$ if and only if w is a valid witness.

While the above allows for testing if a witness is valid, we need a way to actually encode a message in the output. There are many ways to do this, but perhaps the simplest is to encrypt 0 by sampling the distribution above, and encrypting 1 by sampling a uniformly random ADP. Let $M_{v,t,m}$ be the resulting ADP. A random ADP over a large field will have $\det(M(w)) \neq 0$ with high probability for *any* w. Therefore for valid witnesses w, we have that $\det(M_{v,t,m}(w)) = 0$ if and only if $m = 0$.

Witness Encryption – Resisting Attacks Through Structured "Noise." It is not difficult to see that the idea so far is completely insecure. The attacker can learn m by, for example, choosing a w^* that satisfies $v \cdot w^* = t$, but where w^* now consists of arbitrary field elements rather than 0/1. Such a w^* can be found trivially using linear algebra. In this case, we will still have that $\det(M(w)) = 0$ if and only if $m = 0$, thus revealing m. We call attacks of this form *invalid input attacks*.

Alternatively, one could simply inspect a single entry of M and notice that in the case $m = 0$, the entry is a multiple of the (known) function $(-t + v \cdot w)$, whereas if $m = 1$ the entry is a random affine function and is most likely not a multiple of $(-t + v \cdot w)$.

To remedy these issues, we will *add* as "noise" a randomized "all-accept" ADP M_{AA}, which is an ADP that accepts every 0/1 input, but rejects (with high probability) any input that is not 0/1. The encryption of 0 will then be

$$M(w) = M_{AA}(w) + M_{v,t,m}(w).$$

Since the all-accept program rejects any input that is not 0/1, meaning $M_{AA}(w)$ is full rank, it is also the case that $M(w)$ is full rank with high probability. Therefore, our witness encryption scheme will reject any invalid inputs, just as a random ADP will. This shows that our witness encryption scheme is at least immune to invalid input attacks.

Of course, as suggested by our noise analogy above, the security of this proposal depends on how "noisy" the all-accept program M_{AA} is. In Section 5 we discuss two simple approaches, and how these approaches evade our attempts at cryptanalysis.

Applications. As suggested in [GGSW13], one application of witness encryption is to build a public key encryption scheme where public keys are extremely short and generated by a lightweight process. In particular, given a PRG G, the secret key will be a seed s and the public key will be the output $x = G(s)$. To encrypt a message m, simply witness encrypt m to the statement "x has a pre-image under G". Correctness is immediate, and security follows from a simple two-step hybrid: first replace x with a random string, which with overwhelming probability will not be in the image of G. Then invoke witness encryption security to show that m is hidden. Under a stronger assumption of *extractable* witness encryption due to [GKP+13], G can even be taken to be an arbitrary one-way function.

We show how to optimize our witness encryption candidate for use with Goldreich's one-way function, which is a very simple one-way function where every output bit depends on only 5 input bits. Thus, public-key generation is *linear-time* in the secret key length. For a plausible setting of parameters, we obtain 300-bit public keys, and 50 megabyte ciphertexts. We note that while our ciphertexts are too large for any practical use, they are well within the realm of implementability. In contrast, existing approaches to building witness encryption using multilinear maps are extremely impractical and their concrete efficiency has never even been estimated, to the best of our knowledge.

Indistinguishability Obfuscation. We now turn to the task of using ADPs to build general-purpose indistinguishability obfuscation ($i\mathcal{O}$). We note that this case is much more challenging than witness encryption. First, there is the obvious question of converting general computations into ADPs. Second, the security requirements for witness encryption are much weaker: it only needs security to hold in the evasive setting, where the adversary cannot find accepting inputs, and the adversary is allowed to know almost the entire program, save for a single message bit.

Fortunately, there are multiple ways to convert a circuit in NC^1 into an ADP, outlined in Section 4, and $i\mathcal{O}$ for NC^1 can be bootstrapped into $i\mathcal{O}$ for all circuits following known techniques [GGH+13b]. However, simply taking such a program and adding an all-accept program as in our witness encryption construction is insufficient, as doing so leads to various attacks, which we outline in Section 9.

As mentioned earlier, the common approach starting with [GGH+13b] to obfuscating circuits in NC^1 first applies Barrington's theorem [Bar86] to obtain a representation of the circuit as a polynomial-size branching program. Then, the branching program is represented as a *matrix branching program*, and the matrix entries are encoded in the exponent of a multilinear map.

In this work, we still make use of branching programs as the underlying computational model, but instead appeal to the works of [IK02, AIK04], which give a representation of any deterministic branching program as an ADP $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$. In particular, this representation has the property that for any binary input $\mathbf{x} \in \{0,1\}^n$, $\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) = 1$ if $\mathbf{x}$ induces an accepting path in the branching program, and $\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) = 0$ otherwise.

We argue that this ADP representation of branching programs has several qualitative advantages over the matrix branching program approach. Notably, the earlier matrix branching program approach gives rise to so-called "mixed input" attacks, since multiple matrices are associated with each input bit. In contrast, the ADP encoding associates a single matrix with each input bit.

In this work, we initiate a systematic study of various randomization strategies that can be applied to ADPs that encode branching programs, in the hope that such safeguards will lead to a secure obfuscation scheme. As a first attempt, we left- and right-randomize with matrices $\mathbf{R}, \mathbf{S}$ over $\mathbb{F}_p$ such that $\det(\mathbf{R}) \cdot \det(\mathbf{S}) = 1$, preserving the correctness property of the ADP, while scrambling the individual matrix entries. However, this safeguard is not sufficient, due to the existence of "polynomial extension attacks," discussed in Section 9.1. Thus, we consider methods of scrambling the underlying ADP encoding first, before applying the left and right randomization as a final step.

In particular, we make use of the fact that on honest evaluation, the determinant will always be in $\{0,1\}$. This motivates the possibility of adding random *even-valued* noise to the matrices in the ADP, which shifts the determinant by an even value on every input. Correctness is then preserved by simply having the evaluator compute the parity of the determinant. Parameters must be set carefully so that the size of the field over which these determinant calculations are being carried out is large enough so that determinants on honest inputs do not wrap around the modulus. This is discussed in detail in Section 8.1.

Unfortunately, as we show in Section 9.3, this even-valued error does not yet give a secure obfuscation scheme. In particular, there exist attacks on $i\mathcal{O}$ that use the fact that the adversary may know the underlying structure of the branching program it is attacking to learn non-trivial information about the random even-valued error. This in turn motivates the need to inject entropy into the structure of the branching program itself, before encoding it as an ADP and applying the safeguards described above. We term this process "random local substitution," where each individual "step" of the branching program is randomized, and discuss one concrete method of doing this in Section 8.1.1. However, we stress that there is a vast space of transformations to explore here, and our proposal is only the first of many simple possibilities to explore.

In summary, we present the following generic recipe for obfuscating a deterministic branching program BP:

- Apply random local *functionality-preserving* transformations to BP, producing BP'.
- Represent BP' as an ADP, following [IK02, AIK04].
- Add random even-valued error to each matrix in ADP, fixing the evaluation function to compute the parity of the determinant.
- Left- and right-randomize with matrices $\mathbf{R}, \mathbf{S}$ such that $\det(\mathbf{R}) \cdot \det(\mathbf{S}) = 1$.

In the body, we give an instantiation of the above recipe, resulting in a candidate $i\mathcal{O}$ scheme for all circuits (noting that it suffices to obfuscate the class NC^1, which can be simulated by polynomial-size branching programs). However, we view the recipe itself as the main contribution of this section, and hope that it will motivate other candidates and more cryptanalysis, with the eventual goal of obtaining efficient and secure obfuscation without the use of multilinear maps.

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 $p \in \mathbb{N}$, denote $\mathbb{Z}/p\mathbb{Z}$ by $\mathbb{Z}_p$, and denote the finite field of 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 ith 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 $f(n)$ is said to be negligible if it is $n^{-\omega(1)}$ and we denote it by $f(n) := \mathsf{negl}(n)$. A probability $p(n)$ is said to be overwhelming if it is $1 - n^{-\omega(1)}$. We write $D_1 \approx_S D_2$ if there exists a negligible function $\mathsf{negl}(n)$ such that the statistical distance between

D_1 and D_2, denoted $SD(D_1, D_2)$, is bounded above by $\mathsf{negl}(n)$. Recall for any two distribution D_1 and D_2 taking values in some set $\mathcal{S}$ the statistical distance or $SD(D_1, D_2)$ is defined as:

$$SD(D_1, D_2) = 1/2 \sum_{x \in \mathcal{S}} |\Pr[D_1 = x] - \Pr[D_2 = x]|$$

We write $D_1 \approx_C D_2$ if no computationally-bounded adversary can distinguish between D_1 and D_2 except with advantage $\mathsf{negl}(n)$.

2.1 Randomized Encodings

A randomized encoding of the function $f(x)$ is a function $\hat{f}(x, r)$ such that a sample from the output distribution of $\hat{f}(x, r)$ for a uniformly chosen r reveals $f(x)$ and no additional information about x. We formalize and generalize this below.

Definition 1 (Randomized Encodings [AIK07])**.** Let $f : \{0,1\}^n \to \{0,1\}^\ell$ be a function. We say that $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a δ-correct, ϵ-private randomized encoding of f, if it satisfies the following:

- **δ-correctness**. There exists an algorithm B, called a decoder, such that for any input $\mathbf{x} \in \{0,1\}^n$,

$$\Pr[B(\hat{f}(x, U_m)) \neq f(x)] \leq \delta.$$

- **ϵ-privacy**. There exists a randomized simulator algorithm Sim such that for any $\mathbf{x} \in \{0,1\}^n$,

$$SD(\mathsf{Sim}(f(x)), \hat{f}(x, U_m)) \leq \epsilon.$$

2.2 Witness Encryption

The following definition is taken almost verbatim from [GGSW13].

Definition 2. A witness encryption scheme for an NP language L (with corresponding witness relation R) consists of the following two polynomial-time algorithms:

- $\mathsf{Encrypt}(1^\lambda, x, m)$ takes as input a security parameter 1^λ, an unbounded-length string x, and a message $m \in \{0,1\}$, and outputs a ciphertext ct.
- $\mathsf{Decrypt}(ct, w)$ takes as input a ciphertext ct and an unbounded-length string w, and outputs a message m or the symbol $\perp$.

These algorithms satisfy the following two conditions:

- **Correctness.** For any security parameter λ, for any $m \in \{0,1\}$, and for any $x \in L$ such that $R(x, w)$ holds, we have that

$$\Pr[\mathsf{Decrypt}(\mathsf{Encrypt}(1^\lambda, x, m), w) = m] = 1.$$

- **Soundness Security.** For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $x \notin L$, we have:

$$|\Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, x, 0)) = 1] - \Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, x, 1)) = 1]| < \mathsf{negl}(\lambda).$$

We also consider an "extractable" version of witness encryption, introduced by [GKP$^+$13]. In extractable witness encryption, if an adversary that can break semantic security for an instance x, an extractor can extract the witness for x. Formally, in an extractable witness encryption, the following definition (taken verbatim from [GKP$^+$13]) replaces the standard soundness security notion.

- **Extractable Security.** A witness encryption scheme for a language $L \in \mathsf{NP}$ is secure if for all PPT adversary $\mathcal{A}$ and all polynomials q, there exists a PPT extractor E and a polynomial $p(\cdot)$ such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$, the following holds:

$$\Pr[m \leftarrow \{0,1\}; ct \leftarrow \mathsf{WE.Encrypt}(1^\lambda, x, m) : \mathcal{A}(x, ct, z) = b] \geq \frac{1}{2} + \frac{1}{q(|x|)}$$
$$\implies \Pr[\mathsf{E}(x, z) = w : (x, w) \in R_L] \geq \frac{1}{p(|x|)}.$$

2.3 Indistinguishability Obfuscation

Definition 3 (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 $C \in \mathcal{C}_\lambda$,

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

- 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 inputs 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: Syntax and Definitions

An *affine determinant program* (ADP) : $\{0,1\}^n \to \{0,1\}$ is parameterized by an input length n, a width k, and a finite field $\mathbb{F}_p$. An ADP is comprised of an affine function $\mathsf{M} : \{0,1\}^n \to \mathbb{F}_p^{k \times k}$ along with an evaluation function $\mathsf{Eval} : \mathbb{F}_p \to \{0,1\}$. The affine function M is specified by an $(n+1)$-tuple of $k \times k$ matrices $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ over $\mathbb{F}_p$ so that

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

On input $\mathbf{x} \in \{0,1\}^n$, $\mathsf{ADP}_{\mathsf{M},\mathsf{Eval}}(\mathbf{x})$ is computed as

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

We will typically use one of the following Eval functions.

- $\mathsf{Eval}_0(y) = \begin{cases} 1 & \text{if } y = 0 \\ 0 & \text{if } y \neq 0 \end{cases}.$

- $\mathsf{Eval}_{\neq 0}(y) = \begin{cases} 1 & \text{if } y \neq 0 \\ 0 & \text{if } y = 0 \end{cases}$.
- $\mathsf{Eval}_{\text{parity}}(y) = y \bmod 2$.

3.1 Encoding Functions as ADPs

We will consider a randomized procedure ADP-Encode which takes as input a description $\langle f \rangle$ of a function $f : \{0,1\}^n \to \{0,1\}$ and produces $\mathsf{ADP}_{\mathsf{M},\mathsf{Eval}}$ such that $\mathsf{ADP}_{\mathsf{M},\mathsf{Eval}} \equiv f$. Formally, the algorithm works as follows.

- $\mathsf{ADP\text{-}Encode}(1^\lambda, \langle f \rangle)$: This algorithm takes in a security parameter 1^λ and a canonical description $\langle f \rangle$ of a function $f \in \mathcal{F}_n$. It outputs width-k $\mathsf{ADP}_{\mathsf{M},\mathsf{Eval}}$ over $\mathbb{F}_p$ where $k = \mathsf{poly}(\lambda, |\langle f \rangle|)$ and $p = 2^{\mathsf{poly}(\lambda)}$.

In this work, Eval will be fixed as part of the description of ADP-Encode and will be independent of the particular input $\langle f \rangle$. We will therefore view ADP-Encode as an algorithm outputting $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ where $\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n \in \mathbb{F}_p^{k \times k}$.

3.2 One-time Security.

We say that ADP-Encode satisfies *one-time security* if the matrix $\mathsf{M}(\mathbf{x})$ can be simulated given only $f(x)$.

Definition 4 (One-time Security)**.** ADP-Encode is *one-time secure* for $\mathcal{F}_n$ if there exists a PPT simulator Sim such that for any $f \in \mathcal{F}_n$ and any $\mathbf{x} \in \{0,1\}^n$,

$$\mathsf{M}(\mathbf{x}) \approx_S \mathsf{Sim}(f(x))$$

where $\mathsf{M} \leftarrow \mathsf{ADP\text{-}Encode}(1^\lambda, \langle f \rangle)$.

We stress that while $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$, the simulator is only required to simulate a single evaluation of $\mathsf{M}(\mathbf{x}) = \mathbf{A} + \sum_{i \in [n]} x_i \mathbf{B}_i$.

Theorem 1 ([IK02])**.** *There exists an algorithm* ADP-Encode *which satisfies correctness and one-time security for all branching programs.*

Remark 1. One-time security implies that $\mathsf{M}(\mathbf{x})$ is a randomized encoding [IK00] of $f(x)$.

Achieving One-time Security Generically. When the evaluation function is Eval_0 or $\mathsf{Eval}_{\neq 0}$, there is a simple generic transformation that turns any functionality-preserving ADP-Encode procedure into one that additionally satisfies one-time security.

We first import the following lemma from [IK97].

Lemma 1 ([IK97])**.** *For any fixed* $\mathbf{A} \in \mathbb{F}_q^{k \times k}$ *and uniformly random invertible* $\mathbf{R}, \mathbf{S} \leftarrow \mathbb{F}_q^{k \times k}$*, the distribution of* $\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S}$ *depends only on* $\mathrm{rank}(\mathbf{A})$.

Let ADP-Encode be an encoding procedure which uses Eval_0 or $\mathsf{Eval}_{\neq 0}$, and outputs

$$\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n) \in (\mathbb{F}_p^{k \times k})^{n+1}.$$

Claim 1 (One-Time Security for $\mathsf{Eval}_0, \mathsf{Eval}_{\neq 0}$)**.** *Suppose* ADP-Encode *has the property that when* $\det(\mathsf{M}(\mathbf{x})) = 0$*, then* $\operatorname{rank}(\mathsf{M}(\mathbf{x})) = k'$ *for some fixed* $k' < k$*. Furthermore,* ADP-Encode *uses* Eval_0 *or* $\mathsf{Eval}_{\neq 0}$*. Let* ADP-Encode$'$ *be an algorithm which runs* ADP-Encode*, samples uniformly random invertible* $\mathbf{R}, \mathbf{S} \leftarrow \mathbb{F}_p^{k \times k}$*, and outputs*

$$\mathbf{R} \cdot \mathsf{M} \cdot \mathbf{S} := (\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S}, \mathbf{R} \cdot \mathbf{B}_1 \cdot \mathbf{S}, \ldots, \mathbf{R} \cdot \mathbf{B}_n \cdot \mathbf{S}).$$

Then ADP-Encode$'$ *has identical functionality to* ADP-Encode*, and is additionally one-time secure.*

Proof. Identical functionality holds since $\det(\mathsf{M}(\mathbf{x})) = 0$ implies $\det(\mathbf{R} \cdot \mathsf{M}(\mathbf{x}) \cdot \mathbf{S}) = 0$, and $\det(\mathsf{M}(\mathbf{x})) \neq 0$ implies $\det(\mathbf{R} \cdot \mathsf{M}(\mathbf{x}) \cdot \mathbf{S}) \neq 0$ since $\mathbf{R}, \mathbf{S}$ are invertible.

One-time security follows from the fact that given $f(x)$, $\mathbf{R} \cdot \mathsf{M}(\mathbf{x}) \cdot \mathbf{S}$ can be simulated by either drawing a random matrix of rank k' or of rank k (by Lemma 1) □

3.3 $i\mathcal{O}$ Security.

In contrast to one-time security, $i\mathcal{O}$ security holds even if the adversary is free to evaluate M on as many inputs $\mathbf{x}$ of its choosing. In particular, we ask that $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ be an $i\mathcal{O}$ of the input program f.

Definition 5. ADP-Encode is *$i\mathcal{O}$-secure* for $\mathcal{F}_n$ if for any two functions $f_1, f_2 \in \mathcal{F}_n$ such that $f_1 \equiv f_2$ and $|\langle f_1 \rangle| = |\langle f_2 \rangle|$, then

$$\mathsf{M}_1 \approx_C \mathsf{M}_2$$

where $\mathsf{M}_1 \leftarrow \mathsf{ADP\text{-}Encode}(1^\lambda, \langle f_1 \rangle)$ and $\mathsf{M}_2 \leftarrow \mathsf{ADP\text{-}Encode}(1^\lambda, \langle f_2 \rangle)$.

4 Constructing ADPs

4.1 Obfuscating Simple Functionalities with ADPs

In this section we describe simple ADP-based obfuscations of point functions and conjunctions. These constructions have appeared before in the literature, but we recast them in the language of our ADP framework.

Point Functions. Consider the class of point functions $\{\mathcal{I}_{\mathbf{y}}\}_{\mathbf{y}}$ over $\{0,1\}^n$, where for any $\mathbf{x} \in \{0,1\}^n$, $\mathcal{I}_{\mathbf{y}}(\mathbf{x})$ outputs 1 if and only if $\mathbf{x} = \mathbf{y}$, and outputs 0 otherwise. There exists a simple ADP encoding scheme for point functions due to [BKM+18] where each ADP matrix is 1×1 (i.e. a scalar).

On input point $\mathbf{y}$, the ADP encoding of $\mathcal{I}_{\mathbf{y}}$ is generated as follows. Fix a superpolynomial-size modulus p, draw uniformly random $b_1, \ldots, b_n \leftarrow \mathbb{Z}_p$, and set

$$a = -\sum_{i \in [n]: y_i = 1} b_i.$$

The resulting scalar ADP is $\mathsf{M} = (a, b_1, \ldots, b_n)$. To evaluate M on input $\mathbf{x} \in \{0,1\}^n$, compute the scalar sum $\mathsf{M}(\mathbf{x}) = a + \sum_{i \in [n]} x_i b_i$ and output 1 if and only if the resulting scalar is 0. Correctness holds with high probability on any input since the modulus p is superpolynomial. Observe that this ADP construction fits into the more general framework of taking a determinant of an affine combination of matrices, since the determinant is simply the identity for 1×1 matrices.

If the accepting point $\mathbf{y}$ is chosen from a distribution with sufficient entropy, this ADP information-theoretically hides $\mathbf{y}$ (this follows immediately from the Leftover Hash Lemma).

Conjunctions. Point functions can be naturally extended to *conjunctions*, a more expressive class of functionalities that allow for "wildcard" positions (which we denote by $*$) indicating locations where the input bit can be either 0 or 1. Formally, a conjunction $\mathcal{C}_{\mathsf{pat}}$ is parameterized by a pattern string $\mathsf{pat} \in \{0,1,*\}^n$. On input $\mathbf{x} \in \{0,1\}^n$, $\mathcal{C}_{\mathsf{pat}} = 1$ if and only if $\mathbf{x}$ matches pat at all indices i where $\mathsf{pat}_i \in \{0,1\}$. Bartusek et al. [BLMZ19] give a simple ADP-based obfuscation for conjunctions, which we recall here.

For $\mathsf{pat} \in \{0,1,*\}^n$, the corresponding ADP will consist of square matrices of width n over $\mathbb{F}_p$ where p is a superpolynomial-size prime. We use the notation $\mathsf{colspan}(\mathbf{M})$ to denote the linear subspace spanned by the columns of $\mathbf{M}$.

1. Sample a uniformly random rank-$(n-1)$ matrix $\mathbf{L} \in \mathbb{F}_p^{n \times n}$.
2. For each index $i \in [n]$ where $\mathsf{pat}_i \in \{0,1\}$, sample a uniformly random rank-one matrix $\mathbf{B}_i \in \mathbb{F}_p^{n \times n}$.
3. For each index $i \in [n]$ where $\mathsf{pat}_i = *$, sample a uniformly random rank-one matrix $\mathbf{B}_i$ conditioned on $\mathsf{colspan}(\mathbf{B}_i) \subseteq \mathsf{colspan}(\mathbf{L})$. Since a rank-one $\mathbf{B}_i$ can be written as $\mathbf{u}_i \mathbf{v}_i^\top$, this is equivalent to sampling $\mathbf{v}_i$ uniformly at random from $\mathbb{F}_p^n$, sampling $\mathbf{w}_i$ uniformly at random from $\mathbb{F}_p^n$, and setting $\mathbf{u}_i \coloneqq \mathbf{L}\mathbf{w}_i$.
4. Set $\mathbf{A} = \mathbf{L} - \sum_{i \in [n], \mathsf{pat}_i = 1} \mathbf{B}_i$, and output

$$\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$$

To evaluate the ADP M on input $\mathbf{x} \in \{0,1\}^n$, compute

$$\mathsf{M}(\mathbf{x}) = \mathbf{A} + \sum_{i \in [n] : x_i = 1} \mathbf{B}_i$$

and accept (output 1) if and only if $\det(\mathsf{M}(\mathbf{x})) = 0$.

Correctness follows since for any input $\mathbf{x}$ that agrees with pat on all 0/1 positions, $\mathsf{colspan}(\mathsf{M}(\mathbf{x})) \subseteq \mathsf{colspan}(\mathbf{L})$. Since $\mathbf{L}$ is rank-$(n-1)$, $\det(\mathsf{M}(\mathbf{x}))$ will be 0. On any input $\mathbf{x}$ that disagrees with pat on a 0/1 position, $\mathsf{M}(\mathbf{x})$ is the sum of a random rank-$(n-1)$ matrix and at least one independent random rank-one matrix, which yields a full-rank with overwhelming probability.

Bartusek et al. [BLMZ19] prove that this construction statistically hides pat assuming that pat has sufficient entropy on its 0/1 entries. The entropy of pat is used to argue that $\mathbf{A} = \mathbf{L} - \sum_{i \in [n], \mathsf{pat}_i = 1} \mathbf{B}_i$ is statistically close to a uniformly random matrix. This means $\mathbf{L}$ is statistically hidden, and so for all $i \in [n]$, $\mathbf{B}_i$ is statistically close to uniformly random rank-one matrix. At this point, the distribution of $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ is independent of pat.

4.2 ADPs for Log-depth Boolean Formulas

We now give a simple construction that directly converts any log-depth formula into a functionally equivalent ADP. In contrast to the point function and conjunction setting, we have been unable to prove that this ADP encoding achieves any meaningful notion of obfuscation security. Nevertheless, this encoding procedure will be useful for one of our candidate witness encryption constructions (see Section 5), as well as to develop general intuition for the ADP framework.

We describe the construction recursively, associating an ADP with each wire in the formula and demonstrating how to construct an output ADP given two input ADPs to either an OR gate or an AND gate.

Fix a superpolynomial-size prime p. For this ADP construction, evaluation is done by computing the determinant of $\mathsf{M}(\mathbf{x}) := \mathbf{A} + \sum_{i \in [n]: x_i = 1} \mathbf{B}_i$ over $\mathbb{F}_p$ and accepting if and only if the result is 0.

Throughout this construction, we maintain the invariant that (with overwhelming probability) on an accepting input $\mathsf{M}(\mathbf{x})$ will be rank-deficient by 1, and on a rejecting input $\mathsf{M}(\mathbf{x})$ will be full-rank.

- Positive Input Wire: An ADP for the (positive) input wire $f(x_1, \ldots, x_n) = x_j$ consists of $n+1$ scalars (i.e. a width-1 ADP) $(a, b_1, \ldots, b_n)$ set as follows. Sample a uniformly random $r \leftarrow \mathbb{F}_p$ and set $a = -r$ and $b_j = r$. For all $i \neq j$, set $b_i = 0$.
- Negated Input Wire: An ADP for the (negated) input wire $f(x_1, \ldots, x_n) = \neg x_j$ consists of $n+1$ scalars $(a, b_1, \ldots, b_n)$ set as follows. Sample a uniformly random $r \leftarrow \mathbb{F}_p$ and set $b_j = r$. Set $a = 0$ and $b_i = 0$ for all $i \neq j$.

For any input $\mathbf{x} \in \{0,1\}^n$, correctness of the input wire ADP constructions holds with high probability over the randomness of r.

The ADP M that computes the AND of two n-bit input ADPs $\mathsf{M}_1, \mathsf{M}_2$ with widths w_1, w_2 respectively is constructed as follows. Define $k := k_1 + k_2$. Sample a uniformly random matrix $\mathbf{R} \leftarrow \mathbb{F}_q^{(k-1) \times k}$ and a uniformly random matrix $\mathbf{S} \leftarrow \mathbb{F}_q^{k \times (k-1)}$. Output

$$\mathsf{M}(\mathbf{x}) = \mathbf{R} \cdot \begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & 0 \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix} \cdot \mathbf{S}.$$

We briefly sketch correctness of the AND construction.

- (Both input wires accept) If $\det(\mathsf{M}_1(\mathbf{x})) = 0$ and $\det(\mathsf{M}_2(\mathbf{x})) = 0$, then

  $$\begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & 0 \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix}$$

  is a $k \times k$ matrix which is rank-deficient by 2. Left- and right-multiplying by random matrices $\mathbf{R} \leftarrow \mathbb{F}_q^{(k-1) \times k}, \mathbf{S} \leftarrow \mathbb{F}_q^{k \times (k-1)}$ yields $\mathsf{M}(\mathbf{x})$ which is rank deficient by 1 with overwhelming probability.
- (Some input wire rejects) If either of $\det(\mathsf{M}_1(\mathbf{x})) \neq 0$ or $\det(\mathsf{M}_2(\mathbf{x})) \neq 0$ holds, then

  $$\begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & 0 \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix}$$

  is a $k \times k$ matrix which is rank-deficient by at most 1. Left- and right-multiplying by random matrices $\mathbf{R} \leftarrow \mathbb{F}_q^{(k-1) \times k}, \mathbf{S} \leftarrow \mathbb{F}_q^{k \times (k-1)}$ yields $\mathsf{M}(\mathbf{x})$ which is full-rank with overwhelming probability.

An ADP M which computes the OR of two n-bit input ADPs $\mathsf{M}_1, \mathsf{M}_2$ with widths w_1, w_2 respectively can be constructed as follows. Define $k \coloneqq k_1 + k_2$. Sample a uniformly random matrix $\mathbf{R} \leftarrow \mathbb{F}_q^{k \times k}$ and a uniformly random matrix $\mathbf{S} \leftarrow \mathbb{F}_q^{k \times k}$. Sample a uniformly random affine function $\mathbf{T} : \{0,1\}^n \to \mathbb{F}_q^{k_1 \times k_2}$. Output

$$\mathsf{M}(\mathbf{x}) = \mathbf{R} \cdot \begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & \mathbf{T}(\mathbf{x}) \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix} \cdot \mathbf{S}.$$

We briefly sketch correctness of the OR construction.

- (Either input wire accepts) If $\det(\mathsf{M}_1(\mathbf{x})) = 0$ or $\det(\mathsf{M}_2(\mathbf{x})) = 0$, then

$$\begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & \mathbf{T}(\mathbf{x}) \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix}$$

  is a $k \times k$ matrix which is rank-deficient by 1; even if both $\det(\mathsf{M}_1(x)) = 0$ and $\det(\mathsf{M}_2(x)) = 0$ hold, the random $\mathbf{T}(\mathbf{x})$ will ensure the rank is only deficient by 1 with overwhelming probability. Left- and right-multiplying by random matrices $\mathbf{R} \leftarrow \mathbb{F}_q^{k \times k}, \mathbf{S} \leftarrow \mathbb{F}_q^{k \times k}$ yields $\mathsf{M}(\mathbf{x})$ which is still rank deficient by 1 with overwhelming probability.

- (Both input wires reject) If both $\det(\mathsf{M}_1(\mathbf{x})) \neq 0$ and $\det(\mathsf{M}_2(\mathbf{x})) \neq 0$ hold, then

$$\begin{pmatrix} \mathsf{M}_1(\mathbf{x}) & \mathbf{T}(\mathbf{x}) \\ 0 & \mathsf{M}_2(\mathbf{x}) \end{pmatrix}$$

  is a full-rank $k \times k$ matrix. Left- and right-multiplying by random matrices $\mathbf{R} \leftarrow \mathbb{F}_q^{k \times k}, \mathbf{S} \leftarrow \mathbb{F}_q^{k \times k}$ still yields a full-rank $\mathsf{M}(\mathbf{x})$ with overwhelming probability.

4.3 Encoding Branching Programs as ADPs

In this section we describe a way to represent branching programs as ADPs. This representation is implicit in [IK02] (see also [AIK04]). Different notions of branching programs (e.g., deterministic vs. non-deterministic vs. counting) correspond to different choices of the field and evaluation Eval.

We start by defining the most expressive notion and then specialize it to the weaker notions on which we will rely.

Definition 6 (Counting Branching Programs [IK02])**.** A $\mathbb{Z}$-arithmetic branching program computing $f : \{0,1\}^n \to \mathbb{Z}$ is specified by a directed acyclic graph $G = (V, E)$ and a labeling $\phi(\cdot,\cdot)$ where each edge (u, v) is labeled with a literal $\phi(u, v) = x_i$ or $\phi(u, v) = \neg x_i$, or the constant 1. Two of the vertices are labeled s, t. Its size is $|V| - 1$. Any input x induces a subgraph G_x limited to edges consistent with x (i.e. edges that evaluate to 1 on x). An accepting path on input x is a directed $s - t$ path in G_x. A $\mathbb{Z}$-arithmetic branching program (ABP) computes the function $f : \{0,1\}^n \to \mathbb{Z}$ such that $f(x)$ is the number of accepting paths. It computes a boolean function $f : \{0,1\}^n \to \{0,1\}$ if the number of accepting paths is equal to $f(x) \in \{0,1\}$.

Lemma 2 ([IK02, AIK04])**.** *Suppose there is a $\mathbb{Z}$-arithmetic branching program (ABP) of size ℓ computing a boolean function f. Suppose $L(x)$ satisfies the following*

- *$L(x)$ has -1 along the second diagonal (right below the main diagonal).*

- *$L(x)$ is 0 below the second diagonal.*
- *Each entry of $L(x)$ is a degree (at most) 1 polynomial in a single input variable x_i.*

  More precisely, $L(x)$ is defined as follows. Fix a topological ordering of the vertices in V, and label the columns / rows (from left to right / top to bottom) according to this ordering of vertices. In particular we want s labeled 1 and t labeled ℓ. We first define a matrix $A(x)$ of dimension $\ell \times \ell$. For entry (i, j) of $A(x)$, write $\phi(i, j)$ if (i, j) is an edge in G and 0 otherwise. Note that $A(x)$ will be 0 on and below the main diagonal. Now consider $A(x) - I$, and delete its first column and last row to obtain the $(\ell - 1) \times (\ell - 1)$ dimensional matrix $L(x)$.

Then for all $x \in \{0, 1\}^n$, we have $\det(L(x)) = f(x)$. (If f is boolean, we say $f(x)$ accepts if $\det(L(x))$ is non-zero.)

This immediately gives us an ADP for $\mathbb{Z}$-arithmetic branching programs. Namely, $\mathsf{M} = L$ and Eval is simply the identity function (if f is boolean). This ADP can also be implemented over a finite field $\mathbb{F}_p$ if the number of accepting paths is at most $p-1$. Simply compute $f(x) = \det(L(x)) \bmod p$.

Example. Suppose we have vertices in the order s, v_1, v_2, t. We can write a point function that accepts on $x = 010$ with an edge from s to v_1 labeled with $1 - x_1$, an edge from v_1 to v_2 labeled with x_2, and an edge from v_2 to t labeled with $1 - x_3$. So the resulting $A(x) - I$ and $L(x)$ matrices are

$$A(x) = \begin{pmatrix} -1 & 1-x_1 & 0 & 0 \\ 0 & -1 & x_2 & 0 \\ 0 & 0 & -1 & 1-x_3 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

$$L(x) = \begin{pmatrix} 1-x_1 & 0 & 0 \\ -1 & x_2 & 0 \\ 0 & -1 & 1-x_3 \end{pmatrix}.$$

Thus the resulting ADP is of the following form:

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

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

Then, as done in [AIK04] and described above, the resulting program can be "re-randomized" by left- and right-multiplying by uniformly random matrices $\mathbf{R}, \mathbf{S}$ such that $\det(\mathbf{R}) = \det(\mathbf{S}) = 1$.

5 Witness Encryption

5.1 Definitions

In this section we will use ADPs to construct *witness encryption* [GGH+13c]. Since our focus will not be on encoding *functions* as ADPs, we will ignore the Eval part of our ADP formalism. For the remainder of this section, a width-k ADP M over $\mathbb{F}_p$ will refer to a tuple of $n+1$ matrices $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$, where each matrix is in $\mathbb{F}_p^{k\times k}$. Since we will not have a dedicated Eval function, the evaluation of $\mathsf{M}(\cdot)$ on an input $\mathbf{x} \in \{0,1\}^n$ will refer to the matrix

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

Consider the NP-complete subset sum problem (or more generally, *vector* subset sum).

Definition 7. The $\mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$ 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 $\mathbf{H} \cdot \mathbf{w} = \boldsymbol{\ell}$ (over the integers).

We refer to d as the *dimension* of the instance. When $d = 1$, we simply refer to this as $\mathsf{SUBSET\text{-}SUM}$.

Vector Subset Sum Encoding. A dimension $d = 1$ instance $(\mathbf{h}, \ell)$ can be represented as an ADP of arbitrary width k as follows. Sample a uniformly random matrix $\mathbf{R} \leftarrow \mathbb{F}_p^{k\times k}$ and set $\mathbf{A} = -\ell\mathbf{R}$ and $\mathbf{B}_i = h_i\mathbf{R}$ for all $i \in [n]$. This ADP accepts on input $\mathbf{w} \in \{0,1\}^n$ if and only if

$$\det(\mathbf{A} + \sum_{i\in[n]} w_i\mathbf{B}_i) = \det((-\ell + \sum_{i\in[n]} w_i h_i)\mathbf{R}) = 0.$$

We can generalize this approach to any instance $(\mathbf{H}, \boldsymbol{\ell})$ of dimension $d \geq 1$ as follows. We define the procedure $\mathsf{VSS.Encode}$, which takes an instance $(\mathbf{H}, \boldsymbol{\ell})$ of $\mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$ instance and outputs a width-k ADP over $\mathbb{F}_p$.

- $\mathsf{VSS.Encode}((\mathbf{H} \in \mathbb{Z}^{d\times n}, \boldsymbol{\ell} \in \mathbb{Z}^d), p, k)$: Interpret each entry of $\mathbf{H}$ and $\boldsymbol{\ell}$ as the corresponding element over $\mathbb{F}_p$; for correctness we will require $p > \max_{\mathbf{x}\in\{0,1\}^n} ||\mathbf{H}\cdot\mathbf{x}||_\infty$. For each $j \in [d]$, draw uniformly random $\mathbf{R}_j \leftarrow \mathbb{F}_p^{k\times k}$. Set $\mathbf{A} \coloneqq -\sum_{j\in[d]} \ell_j \mathbf{R}_j$ and $\mathbf{B}_i \coloneqq \sum_{j\in[d]} H_{j,i}\mathbf{R}_j$ for each $i \in [n]$. Output

$$\mathsf{M}_{\mathbf{H},\boldsymbol{\ell}} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n).$$

An Insecure Witness Encryption. Consider the following *insecure* construction of witness encryption for $\mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$. To "encrypt" a bit b with respect to an instance $(\mathbf{H}, \boldsymbol{\ell})$:

1. Fix a width k and a prime $p > \max_{\mathbf{x}\in\{0,1\}^n} ||\mathbf{H}\cdot\mathbf{x}||_\infty$ such that p is at least super-polynomial in the security parameter.

2. Sample

$$\mathsf{M}_{\mathbf{H},\boldsymbol{\ell}} \leftarrow \mathsf{VSS.Encode}((\mathbf{H}, \boldsymbol{\ell}), p, k).$$

3. Sample uniformly random $\mathbf{S} \leftarrow \mathbb{F}_p^{k\times k}$ and output the ciphertext

$$\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b} \coloneqq \mathsf{M}_{\mathbf{H},\boldsymbol{\ell}} + (b\mathbf{S}, \mathbf{0}, \ldots, \mathbf{0}).$$

To decrypt a ciphertext of the form $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ using a witness $\mathbf{w} \in \{0,1\}^n$, we compute

$$\det(\mathbf{A} + \sum_{i \in [n]} w_i \mathbf{B}_i).$$

If this determinant is a non-zero element in $\mathbb{F}_p$, then $b = 1$ and if the result is zero then $b = 0$.

Correctness holds since if $\mathbf{w} \in \{0,1\}^n$ is a witness for $(\mathbf{H}, \boldsymbol{\ell}) \in \mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$, then $\mathbf{H} \cdot \mathbf{w} = \boldsymbol{\ell}$, so

$$\mathbf{A} + \sum_{i \in [n]} w_i \mathbf{B}_i = b\mathbf{S}.$$

Since $\mathbf{S}$ is a random matrix in $\mathbb{F}_p$, where p is super-polynomial in the security parameter, its determinant is nonzero with overwhelming probability.

However, this construction is insecure; given an instance $(\mathbf{H}, \boldsymbol{\ell}) \notin \mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$, an adversary can potentially find $\mathbf{x} \in \mathbb{F}_p^n$ (where $\mathbf{x} \notin \{0,1\}^n$) such that $\mathbf{H} \cdot \mathbf{x} = \boldsymbol{\ell}$. The same correctness argument used for *valid* witnesses $\mathbf{w} \in \{0,1\}^n$ will also imply that for such an $\mathbf{x} \in \mathbb{F}_p^n$, the "encrypted" bit b can be recovered as

$$b = \mathsf{Eval}_{\neq 0}(\det(\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b}(\mathbf{x}))).$$

Preventing Invalid Evaluations. For this approach to have any hope for security, we will need to modify the ciphertext $\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b}$ so that $\det(\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b}(\mathbf{x}))$ does not leak the value of b on inputs $\mathbf{x} \notin \{0,1\}^n$.

Our approach is to add "noise" to $\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b}$ which will prevent the adversary from gaining information on non-binary inputs. This noise will take the form of another ADP $\mathsf{M}_{\mathsf{noise}}$ which will satisfy the following properties:

- (Zero on binary inputs) For all $\mathbf{x} \in \{0,1\}^n$, $\det(\mathsf{M}_{\mathsf{noise}}(\mathbf{x})) = 0$,
- (Non-zero on non-binary inputs): For all $\mathbf{x} \in \mathbb{F}_p^n \setminus \{0,1\}^n$, $\Pr[\det(\mathsf{M}_{\mathsf{noise}}(\mathbf{x})) = 0] = \mathsf{negl}(n)$,

where the probability is taken over the randomness used to generate $\mathsf{M}_{\mathsf{noise}}$. In the construction, we will simply add $\mathsf{M}_{\mathsf{noise}}$ to $\mathsf{M}_{\mathbf{H},\boldsymbol{\ell},b}$, so the "non-zero on non-binary inputs" property will intuitively block the attack described above. We set $\mathsf{M}_{\mathsf{noise}}$ to be the result of sampling from an "All-Accept" encoding scheme, defined as follows.

An All-Accept ADP encoding scheme $\mathsf{AllAcc.Gen}$ is a randomized procedure that takes an input length n and field $\mathbb{F}_p$, and outputs M of width k with the following properties:

- (Correctness) For all $\mathbf{x} \in \{0,1\}^n$,

$$\Pr[\det(\mathsf{M}(\mathbf{x})) = 0 : \mathsf{AllAcc.Gen}(n, \mathbb{F}_p) \to (\mathsf{M}, k)] = 1.$$

- (Rejection of Invalid Inputs) For all non-binary $\mathbf{x}$, i.e. $\mathbf{x} \in \mathbb{F}_p^n \setminus \{0,1\}^n$,

$$\Pr[\det(\mathsf{M}(\mathbf{x})) = 0 : \mathsf{AllAcc.Gen}(n, \mathbb{F}_p) \to (\mathsf{M}, k)] = \mathsf{negl}(n).$$

An All-Accept ADP from a Boolean Formula Encoding. A natural attempt to construct such an encoding scheme would be to use the formula encoding scheme described in Section 4.2 to encode any formula that accepts all boolean inputs $\mathbf{x} \in \{0,1\}^n$. Unfortunately, this does not in general guarantee rejection of invalid inputs. However, it turns out that encoding the simple boolean formula

$$f(x_1, \dots, x_n) = (x_1 \vee \neg x_1) \wedge \dots \wedge (x_n \vee \neg x_n)$$

via the procedure described above does guarantee rejection of invalid inputs. This sampling procedure is equivalent to the concrete procedure given here.

$\mathsf{AllAcc.Gen}^{\mathsf{FORM}}(n, \mathbb{F}_p)$:

- Draw uniformly at random 4 sets of n vectors of dimension $n+1$:

$$\{\mathbf{u}_i\}_{i\in[n]}, \{\mathbf{v}_i\}_{i\in[n]}, \{\mathbf{s}_i\}_{i\in[n]}, \{\mathbf{t}_i\}_{i\in[n]},$$

  along with n^2 scalars $\{c_j^{(i)}\}_{i,j\in[n]}$ over $\mathbb{F}_p$.

- Let

$$\mathbf{A} = \sum_{i\in[n]} \mathbf{u}_i \mathbf{v}_i^\top,$$
$$\mathbf{B}_i = -\mathbf{u}_i \mathbf{v}_i^\top + \mathbf{s}_i \mathbf{t}_i^\top + \sum_{j\in[n]} c_i^{(j)} \mathbf{u}_j \mathbf{t}_j^\top,$$

  and output $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n)$.

Correctness and security follow by fixing the field size $p = \omega(\mathsf{poly}(n))$ and rewriting an evaluation on input $\mathbf{x} \in \mathbb{F}_p^n$ as follows, setting $k_i \coloneqq \sum_{j\in[n]} x_j c_j^{(i)}$:

$$\begin{aligned}
\mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i &= \sum_{i\in[n]} \left((1 - x_i)\mathbf{u}_i \mathbf{v}_i^\top + x_i \mathbf{s}_i \mathbf{t}_i^\top + k_i \mathbf{u}_i \mathbf{t}_i^\top \right) \\
&= \sum_{i\in[n]:x_i=0} \mathbf{u}_i \left(\mathbf{v}_i^\top + k_i \mathbf{t}_i^\top \right) + \sum_{i\in[n]:x_i=1} \left(\mathbf{s}_i + k_i \mathbf{u}_i \right) \mathbf{t}_i^\top \\
&\quad + \sum_{i\in[n]:x_i\notin\{0,1\}} \mathbf{u}_i \left((1 - x_i)\mathbf{v}_i^\top + k_i \mathbf{t}_i^\top \right) + x_i \mathbf{s}_i \mathbf{t}_i^\top .
\end{aligned}$$

Thus when $\mathbf{x}$ is binary, $\mathbf{A} + \sum_{i\in[n]} x_i \mathbf{B}_i$ can be written as a sum of n rank one matrices (recall the dimension is $n+1$). Otherwise, it can be written as a sum of at least $n+1$ rank one matrices which do not share column or row spans (with overwhelming probability over the sampling randomness). Since the field size is $p = \omega(\mathsf{poly}(n))$, the resulting matrix will be full rank except with probability negligible in the input length n.

We now describe a slight generalization of the above sampling procedure, which will help with security of the eventual witness encryption candidate. We replace the vectors $\mathbf{u}_i, \mathbf{v}_i, \mathbf{s}_i, \mathbf{t}_i$ with matrices $\mathbf{U}_i, \mathbf{V}_i, \mathbf{S}_i, \mathbf{T}_i$ of width $q(n)$ (where $q(\cdot)$ is now a parameter for the encoding procedure). In the sampling procedure given above, $q(n)$ is implicitly set to always output 1. In general, the resulting dimension of the output program will be set to $k = nq(n)+1$. This more general sampling procedure is used in the concrete candidate we give in Section 5.2.

An All-Accept ADP from "Nearly Skew Symmetric" Matrices. To state our next construction of an all-accept ADP, it will be helpful to define a "nearly-skew-symmetric" matrix.

Definition 8 (Nearly Skew Symmetric Matrices)**.** A matrix $\mathbf{N} \in \mathbb{F}_p^{k \times k}$ is nearly-skew-symmetric (NSS) if $\mathbf{N}_{i,j} = -\mathbf{N}_{i,j}$ for all $i, j > 1$ such that $i \neq j$, and $\mathbf{N}_{i,1} + \mathbf{N}_{1,i} + \mathbf{N}_{i,i} = 0$ for all i.

Observe that any nearly-skew-symmetric matrix will have $\mathbf{N}_{1,1} = 0$. Furthermore, the bottom-right $(k-1)\times(k-1)$ supmatrix of any NSS matrix is skew-symmetric. Setting $n = k-1$, note that it is easy to sample a random NSS by choosing uniformly random field elements $a_1, \ldots, a_n, d_1, \ldots, d_n$, as well as a uniformly random width-n skew-symmetric matrix B, and outputting

$$\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & B & \\ 0 & & & \end{pmatrix} + \begin{pmatrix} 0 & a_1 & \cdots & a_n \\ -a_1 - d_1 & d_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -a_n - d_n & 0 & \cdots & d_n \end{pmatrix}.$$

The above formulation makes it clear that for any finite field $\mathbb{F}_p$ and matrix width k, the set of all NSS matrices lies in a known linear subspace. Let $\mathsf{NSS.Sample}(p, k)$ be the procedure that produces a uniformly random sample from this subspace, i.e. outputs a random nearly-skew-symmetric matrix $\mathbf{N}$ of width k over $\mathbb{F}_p$.

We define the following all-accept encoding procedure.

$\mathsf{AllAcc.Gen}^{\mathsf{NSS}}(n, \mathbb{F}_p)$:

- Let $k = n + 1$, and draw a uniformly random full rank matrix $\mathbf{T} \leftarrow \mathbb{F}_p^{k \times k}$.
- For each $i \in [k]$, draw $\mathbf{N}_i \leftarrow \mathsf{NSS.Sample}(k)$ and set $\mathbf{C}_i = \mathbf{T}^{-1} \cdot \mathbf{N}_i$.
- Let $\mathbf{A}$ be the $k \times k$ matrix obtained by concatenating the first column of each of the $\mathbf{C}_i$ matrices, and let $\mathbf{B}_i$ be the $k \times k$ matrix obtained by concatenating the $i+1$st columns of each of the $\mathbf{C}_i$ matrices. Output $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$.

To show correctness, fix any $\mathbf{x} \in \{0,1\}^n$, and let $\mathbf{x}' = [1 \mid \mathbf{x}^\top]^\top$. Denote the jth column of a matrix $\mathbf{M}$ by $(\mathbf{M})_j$. Then for all $j \in [n+1]$,

$$\mathbf{T} \cdot (\mathbf{A} + \sum_{i \in [n]} x_i \mathbf{B}_i)_j = \mathbf{T} \cdot \mathbf{C}_j \cdot \mathbf{x}' = \mathbf{N}_j \cdot \mathbf{x}'.$$

Now we claim that for all $j, \mathbf{x}'^\top \cdot \mathbf{N}_j \cdot \mathbf{x}' = 0$, which implies that $\mathbf{x}'^\top \cdot \mathbf{T}$ is in the kernel of $\mathbf{A} + \sum_{i \in [n]} x_i \mathbf{B}_i$, and thus that $\det(\mathsf{M}(\mathbf{x})) = 0$. In fact, for any NSS matrix $\mathbf{N}$, and any vector $\mathbf{x}' = [1 \mid \mathbf{x}^\top]^\top$, where $\mathbf{x} \in \{0,1\}^n$, it holds that $\mathbf{x}'^\top \cdot \mathbf{N} \cdot \mathbf{x}' = 0$. To see this, treat each entry of $\mathbf{x}$ as a distinct formal variable, and expand out the expression as a quadratic polynomial over $x_1, \ldots, x_n$. Since $\mathbf{x}$ is binary, we know that $x_i^2 = x_i$. Thus, the coefficient on x_i is equal to $\mathbf{N}_{1,i} + \mathbf{N}_{i,1} + \mathbf{N}_{i,i} = 0$, and the coefficient on $x_i x_j$ for $i \neq j$ is equal to $\mathbf{N}_{i,j} + \mathbf{N}_{j,i} = 0$, which establishes the claim.

To argue security, we take n to be the security parameter, and fix the field size to be $p = \omega(\mathsf{poly}(n))$. Fix any $\mathbf{x} \notin \{0,1\}^n$ and again let $\mathbf{x}' = [1 \mid \mathbf{x}^\top]^\top$. We want to bound the probability that there exists some vector $\mathbf{v} \neq \mathbf{0}$ such that $\mathbf{v}^\top \cdot (\mathbf{A} + \sum_i x_i \mathbf{B}_i) = 0$. Setting $\mathbf{w'}^\top = \mathbf{v}^\top \cdot \mathbf{T}^{-1}$, we have that

$$\mathbf{v}^\top \cdot (\mathbf{A} + \sum_i x_i \mathbf{B}_i) = 0 \Leftrightarrow \mathbf{w'}^\top \cdot \mathbf{N}_i \cdot \mathbf{x'} = 0 \ \forall i \in [n+1].$$

We will union bound over all vectors $\mathbf{w'}^\top$ to show that there does not exist such a vector with high probability. Note that it suffices to union bound over all $\mathbf{w'}$ with first entry equal to 0 or 1, for a total of $2p^n$ vectors.

First consider the case where $\mathbf{w'}^\top = [0 \mid \mathbf{w}^\top]$ for some $\mathbf{w} \in \mathbb{F}_p^n$. Recall that each $\mathbf{N}_i$ is generated by choosing random values $a_1, \ldots, a_n, d_1, \ldots, d_n$ and a skew-symmetric $\mathbf{B}$ and arranging in a matrix as described above. Now treat these values as formal variables, and consider the linear polynomial over these variables induced by the expression $[0 \mid \mathbf{w}^\top] \cdot \mathbf{N}_i \cdot [1 \mid \mathbf{x}^\top]^\top$. We argue that this polynomial must not be identically zero. The coefficient on the variable a_i is exactly $-w_i$. Since $\mathbf{w'} \neq \mathbf{0}$, it must be the case that $\mathbf{w} \neq \mathbf{0}$, so there must be some i such that $w_i \neq 0$.

Now consider the case where $\mathbf{w'}^\top = [1 \mid \mathbf{w}^\top]$ for some $\mathbf{w} \in \mathbb{F}_p^n$. The coefficient on a_i is exactly $w_i(x_i - 1)$ and the coefficient on d_i is exactly $x_i(w_i - 1)$. But by assumption, there must be some i such that $x_i \notin \{0, 1\}$, implying that the coefficient on either a_i or d_i is non-zero. Thus, for each of the $2p^n$ vectors $\mathbf{w'}$ that we consider, the probability that $\mathbf{w'}^\top \cdot \mathbf{N}_i \cdot \mathbf{x'} = 0$ is at most $1/p$ over the randomness of sampling $\mathbf{N}_i$. This holds simultaneously for each $i \in [n+1]$ with probability at most $1/p^{n+1}$. So by a union bound, there exists a $\mathbf{w'}$ orthogonal to each $\mathbf{N}_i \cdot \mathbf{x'}$ with probability at most $2/p = \mathsf{negl}(n)$.

5.2 Candidate Witness Encryption Constructions

Generic Candidate. Given the above definitions, we can define a general paradigm for constructing witness encryption (encrypting a single bit b) for VECTOR-SUBSET-SUM. Instantiating the framework with any AllAcc.Gen and VSS.Encode procedures will result in a correct witness encryption.

- WE.Enc$(b, (\mathbf{H}, \boldsymbol{\ell}))$: Let $d \times n$ be the dimension of $\mathbf{H}$.
    1. Choose a prime $p > \max_{\mathbf{x} \in \{0,1\}^n} ||\mathbf{H} \cdot \mathbf{x}||_\infty$. Additionally we require $p = \omega(\mathsf{poly}(n))$.
    2. Sample $(\mathsf{M}_{\mathsf{AA}}, k) \leftarrow \mathsf{AllAcc.Gen}(n, \mathbb{F}_p)$.
    3. Sample $\mathsf{M}_{\mathbf{H},\boldsymbol{\ell}} \leftarrow \mathsf{VSS.Encode}((\mathbf{H}, \boldsymbol{\ell}), \mathbb{F}_p, k)$.
    4. Sample $\mathbf{S} \leftarrow \mathbb{F}_p^{k \times k}$.
    5. Output $\mathsf{M}_{\mathsf{AA}} + \mathsf{M}_{\mathbf{H},\boldsymbol{\ell}} + (b\mathbf{S}, \mathbf{0}, \ldots, \mathbf{0})$.
- WE.Dec(M, w) : output $\mathsf{Eval}_{\neq 0}(\det(\mathsf{M}(w)))$.

Concrete Candidate. We now give a simple, self-contained instantiation of the general witness encryption framework described above.

This candidate will be a witness encryption for SUBSET-SUM, with instances $(\mathbf{h} \in \mathbb{Z}^n, \ell \in \mathbb{Z})$. Given an instance $(\mathbf{h} \in \mathbb{Z}^n, \ell \in \mathbb{Z})$, define the instance $(\mathbf{H} \in \mathbb{Z}^{(n+1) \times 2n}, \boldsymbol{\ell} \in \mathbb{Z}^{n+1})$ of VECTOR-SUBSET-SUM where

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

As discussed in Section 7, this transformation will help with security. In particular, an instance $((h_1, \dots, h_n), \ell) \notin$ SUBSET-SUM could potentially have many $h_i = 0$, so encoding without the above transformation will leak many $\mathbf{B}_i$ of the all-accept encoding in the clear.

The following instantiation combines the VSS.Encode scheme described above with the all-accept encoding $\mathsf{AllAcc.Gen}^{\mathsf{FORM}}$, with parameter $q(n) = n^{\epsilon}$ for some constant $\epsilon > 0$. As discussed in Section 7.1, setting $q(\cdot)$ to be a super-constant function of n will help defend against MQ-style attack strategies. Note that the encryption scheme below takes as input a one-dimensional SUBSET-SUM instance and first follows the above conversion to a VECTOR-SUBSET-SUM instance. Then, it encrypts under this VECTOR-SUBSET-SUM instance following the generic paradigm above.

- $\mathsf{WE.Enc}(b, ((h_1, \dots, h_n), \ell))$. Choose $p > \max_{\mathbf{x} \in \{0,1\}^n} |\sum_i x_i h_i|$ such that $p = \omega(\mathsf{poly}(n))$, and fix the field $\mathbb{F}_p$.

  1. For each $i \in [2n]$, draw 4 uniformly random matrices
  $$\mathbf{U}_i, \mathbf{V}_i, \mathbf{S}_i, \mathbf{T}_i \leftarrow \mathbb{F}_p^{(2n^{1+\epsilon}+1) \times n^{\epsilon}}.$$
  2. For each $i, j \in [2n]$, draw a uniformly random scalar
  $$c_j^{(i)} \leftarrow \mathbb{F}_p.$$
  3. Draw $2n$ uniformly random matrices
  $$\mathbf{R}_0, \mathbf{R}_1, \dots, \mathbf{R}_n, \mathbf{S} \leftarrow \mathbb{F}_p^{(2n^{1+\epsilon}+1) \times (2n^{1+\epsilon}+1)}.$$
  4. Define
  $$\mathbf{A} \coloneqq \sum_{i \in [2n]} \mathbf{U}_i \mathbf{V}_i^{\top} - \ell \mathbf{R}_0 - \sum_{i \in [n]} \mathbf{R}_i + b\mathbf{S}.$$
  For $\ell \in [n]$, define
  $$\mathbf{B}_{\ell} \coloneqq -\mathbf{U}_{\ell} \mathbf{V}_{\ell}^{\top} + \mathbf{V}_{\ell} \mathbf{T}_{\ell}^{\top} + \sum_{j \in [2n]} c_{\ell}^{(j)} \mathbf{U}_j \mathbf{T}_j^{\top} + h_{\ell} \mathbf{R}_0 + \mathbf{R}_{\ell}$$
  For $\ell \in \{n+1, \dots, 2n\}$, define
  $$\mathbf{B}_{\ell} \coloneqq -\mathbf{U}_{\ell} \mathbf{V}_{\ell}^{\top} + \mathbf{S}_{\ell} \mathbf{T}_{\ell}^{\top} + \sum_{j \in [2n]} c_{\ell}^{(j)} \mathbf{U}_j \mathbf{T}_j^{\top} + \mathbf{R}_{\ell}.$$
  5. Finally, output
  $$\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_{2n}).$$

- $\mathsf{WE.Dec}(\mathsf{M}, w)$. Let $\widehat{w}$ be such that each $\widehat{w}_i = 1 - w_i$. Output $\mathsf{Eval}_{\neq 0}(\det(\mathsf{M}([w \mid \widehat{w}])))$.

6 Applications of ADP-Based Witness Encryption

6.1 Public-Key Encryption with Short Public Keys

In this section, we show how to use our witness encryption candidate to construct a public key encryption scheme with extremely short public keys and with fast key generation.

6.1.1 A Generic Framework.

We recall from [GGSW13] the generic construction of public-key encryption from a witness encryption scheme WE and any pseudo-random generator $g : \{0,1\}^\lambda \to \{0,1\}^{n(\lambda)}$.

- $\mathsf{PKE.Gen}(1^\lambda)$. Sample a uniformly random $\mathsf{sk} \leftarrow \{0,1\}^\lambda$, and set $\mathsf{pk} = g(\mathsf{sk})$. Output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{PKE.Encrypt}(\mathsf{pk}, m \in \{0,1\})$. Define the boolean circuit $C[g, \mathsf{pk}](\cdot)$, which on input $\mathbf{x} \in \{0,1\}^\lambda$, outputs 1 if and only if $g(\mathbf{x}) = \mathsf{pk}$. Interpreting $C[g, \mathsf{pk}](\cdot)$ as an instance of $\mathsf{CIRCUIT\text{-}SAT}$, apply a deterministic reduction from $\mathsf{CIRCUIT\text{-}SAT}$ to $\mathsf{SUBSET\text{-}SUM}$ and obtain an instance $(\mathbf{h}, \ell)$. Output $c \leftarrow \mathsf{WE.Enc}(1^\lambda, (\mathbf{h}, \ell), m)$.
- $\mathsf{PKE.Decrypt}(\mathsf{sk}, (\mathbf{h}, \ell), c)$. Run the reduction from $\mathsf{CIRCUIT\text{-}SAT}$ to $\mathsf{SUBSET\text{-}SUM}$ to transform the witness sk for $C[g, \mathsf{pk}](\cdot)$ to a subset-sum witness $\mathbf{w}$ for $(\mathbf{h}, \ell)$. Return $m \leftarrow \mathsf{WE.Dec}(c, \mathbf{w})$.

Remark 2. The choice of g is fixed in the specification of the public-key encryption scheme and is not included in pk. Notice that 1^λ is an input to $\mathsf{WE.Enc}$ since there is no set-up/gen algorithm in witness encryption, but it is not an explicit input to $\mathsf{PKE.Encrypt}$; however, it can be easily obtained by inspecting the length of pk.

The following claim captures the fact that security of this construction can be proved assuming either the standard security notion for witness encryption [GGSW13], or the stronger extractable security notion given by [GKP+13]; in the latter case the requirement on g can be weakened to one-wayness.

Claim 2 (Security of Witness-Encryption-based PKE)**.** *The above public-key encryption scheme satisfies CPA-security if*

- WE *is secure under the standard [GGSW13] security notion for witness encryption, and* $g : \{0,1\}^\lambda \to \{0,1\}^{n(\lambda)}$ *is a pseudo-random generator with* $n(\lambda) \geq 2\lambda$*, OR*
- WE *is secure under the stronger [GKP+13] security notion for* extractable *witness encryption, and* $g : \{0,1\}^\lambda \to \{0,1\}^{n(\lambda)}$ *is a one-way function.*

Proof. In the case where we use the standard [GGSW13] security notion for witness encryption, an encryption with respect to pk which is not generated as the output of g computationally hides the encrypted message. Therefore, an adversary that can distinguish encryptions of 0 and 1 also distinguishes between outputs of g and uniformly random strings, breaking the pseudorandomness of g.

If we rely on the [GKP+13] extractable security notion, then an adversary which can distinguish encryptions of 0 and 1 implies the existence of an extractor which outputs a witness that pk is in the image of g; this breaks the one-wayness of g. □

6.1.2 A Concrete Instantiation from Goldreich's PRG

For an arbitrary one-way function, the CIRCUIT-SAT to SUBSET-SUM reduction may be costly. Thus, we give a concrete instantiation of the above approach using Goldreich's PRG [Gol00, App16] in place of g, along with a custom reduction to VECTOR-SUBSET-SUM. We first recall the construction of Goldreich's PRG.

Definition 9 (Goldreich's PRG with locality k)**.** Fix a boolean predicate $P : \{0,1\}^k \to \{0,1\}$, and a list $S_1, \ldots, S_m \in [n]^k$ of k-tuples of indices. Write S_i as $(i_1, \ldots, i_k)$. On input $\mathbf{x} \in \{0,1\}^n$, the jth bit of the output is set to $P(x_{i_1}, \ldots, x_{i_k})$.

For this work, we will exclusively focus on the locality $k = 5$ setting with the TSA ("Tri-Sum-And") predicate, defined as

$$\mathrm{TSA}(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 x_5 \mod 2.$$

Hereinafter, "Goldreich's PRG" will refer to the instantiation with the TSA predicate. For a list $S = (S_1, \ldots, S_m)$ where each $S_i \in [n]^5$, let $g_S : \{0,1\}^n \to \{0,1\}^m$ be Goldreich's PRG instantiated with the TSA predicate, parameterized by index sets $S_1, \ldots, S_m$.

Definition 10. The GOLDREICH language consists of instances $(n, \mathbf{y}, (S_1, \ldots, S_m))$ such that each $S_i \in [n]^5$, and $\mathbf{y}$ is in the image of g_S where $S = (S_1, \ldots, S_m)$.

Lemma 3 (GOLDREICH to VECTOR-SUBSET-SUM)**.** *There is a Karp reduction which takes $(n, \mathbf{y}, (S_1, \ldots, S_m))$ in GOLDREICH and outputs $(\mathbf{H}, \boldsymbol{\ell})$ in VECTOR-SUBSET-SUM, where $\mathbf{H} \in \mathbb{Z}^{m \times (n+2m)}$ and $\boldsymbol{\ell} \in \mathbb{Z}^m$.*

Proof. Given a list $S = (S_1, \ldots, S_m)$ with corresponding Goldreich PRG $g_S : \{0,1\}^n \to \{0,1\}^m$, along with a string $\mathbf{y} \in \{0,1\}^m$, we construct an instance $(\mathbf{H}, \boldsymbol{\ell})$ where $\mathbf{H} \in \mathbb{Z}^{m \times (n+2m)}$ and $\boldsymbol{\ell} \in \mathbb{Z}^m$ such that there exists $\mathbf{w} \in \{0,1\}^{n+2m}$ satisfying $\mathbf{H} \cdot \mathbf{w} = \boldsymbol{\ell}$ if and only there exists $\mathbf{x} \in \{0,1\}^n$ satisfying $G_S(\mathbf{x}) = \mathbf{y}$.

We define $n + 2m$ variables by allocating 1 variable for each input position $j \in [n]$, and 2 variables a_i, b_i for each output position $i \in [m]$. We generate m equations as follows. For each $i \in [m]$, write $S_i = (i_1, i_2, i_3, i_4, i_5)$ and consider the pair (S_i, y_i).

- If $y_i = 0$, the associated equation is

$$2x_{i_1} + 2x_{i_2} + 2x_{i_3} - x_{i_4} - x_{i_5} - 4a_i + b_i = 0.$$

- If $y_i = 1$, the associated equation is

$$2x_{i_1} + 2x_{i_2} + 2x_{i_3} + x_{i_4} + x_{i_5} - 4a_i - b_i = 2.$$

Since each equation is a linear equation over $n + 2m$ variables, writing the coefficients of each of the m equations as a matrix yields the matrix $\mathbf{H} \in \mathbb{Z}^{m \times (n+2m)}$. Correspondingly, $\boldsymbol{\ell} \in \mathbb{Z}^m$ is specified by the right-hand-side values of the m equations.

We can extend a pre-image $\mathbf{x} \in \{0,1\}^n$ into a vector $\mathbf{w} \in \{0,1\}^{n+2m}$ satisfying $\mathbf{H} \cdot \mathbf{w} = \boldsymbol{\ell}$ as follows. For each $i \in [m]$:

- If $y_i = 0$, set $a_i = 1$ if and only if $x_{i_1} + x_{i_2} + x_{i_3} \geq 2$.

- If $y_i = 1$, set $a_i = 1$ if and only if $x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4}x_{i_5} \geq 3$.
- Regardless of y_i, set $b_i = 1$ if and only if $x_{i_4} + x_{i_5} = 1$.

It can be verified that if any constraint i is not satisfied, then there is no $\{0, 1\}$ setting of the a_i, b_i variables that can result in a satisfying VECTOR-SUBSET-SUM witness. □

6.1.3 Optimizations and Parameter Size Estimates

In this section, we describe an extension of the above witness encryption candidate for one-bit messages that allows for encoding an h-bit message in a single ADP. We will make use of the NSS based all-accept, and will first generalize the $\mathsf{AllAcc.Gen}^{\mathsf{NSS}}$ algorithm to output wide rectangular matrices of dimension $k \times (k + r)$. We include the parameter r in the input to the sampling procedure, noting that the procedure given before implicitly set $r = 0$.

$\mathsf{AllAcc.Gen}^{\mathsf{NSS}}(n, r, \mathbb{F}_p)$:

- Let $k = n + 1$, and draw a uniformly random full rank matrix $\mathbf{T} \leftarrow \mathbb{F}_p^{k \times k}$.
- For each $i \in [k + r]$, draw $\mathbf{N}_i \leftarrow \mathsf{NSS.Sample}(k)$ and set $\mathbf{C}_i = \mathbf{T}^{-1} \cdot \mathbf{N}_i$.
- Let $\mathbf{A}$ be the $k \times (k + r)$ matrix obtained by concatenating the first column of each of the $\mathbf{C}_i$ matrices, and let $\mathbf{B}_i$ be the $k \times (k + r)$ matrix obtained by concatenating the $i + 1$st column of each of the $\mathbf{C}_i$ matrices. Output $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n)$.

It is also straightforward to generalize the VSS.Encode procedure to take in an additional input r and produce an ADP with matrices of width $k \times (k + r)$. Simply draw the random $\mathbf{R}_i$ matrices over $\mathbb{F}_p^{k \times (k+r)}$.

The optimized witness encryption algorithm will additionally take as input a parameter λ which determines the correctness of the encryption, and the input message will now be $\mathbf{m} \in \{0, 1\}^h$. It operates as follows.

- $\mathsf{WE.Enc}(1^\lambda, \mathbf{m}, (\mathbf{H}, \boldsymbol{\ell}))$: Let $d \times n$ be the dimension of $\mathbf{H}$, and h be the bit length of $\mathbf{m}$.
  1. Choose a prime $p > \max_{\mathbf{x} \in \{0,1\}^n} ||\mathbf{H} \cdot \mathbf{x}||_\infty$.
  2. Set $t = \lceil h / \log(p) \rceil$ and $r = \lambda + t$.
  3. Sample $(\mathsf{M}_{\mathsf{AA}}, k) \leftarrow \mathsf{AllAcc.Gen}^{\mathsf{NSS}}(n, r, \mathbb{F}_p)$.
  4. Sample $\mathsf{M}_{\mathbf{H}, \boldsymbol{\ell}} \leftarrow \mathsf{VSS.Encode}((\mathbf{H}, \boldsymbol{\ell}), \mathbb{F}_p, k, r)$.
  5. Encode $\mathbf{m} \in \{0, 1\}^h$ as a vector $\hat{\mathbf{m}} \in \mathbb{F}_p^t$. Let $\mathbf{X}_{\hat{\mathbf{m}}}$ be the matrix of dimension $k \times (k + r)$ whose only non-zero entries consist of the vector $\hat{\mathbf{m}}$, placed in the last t positions of the first row.
  6. Output $\mathsf{M}_{\mathsf{AA}} + \mathsf{M}_{\mathbf{H}, \boldsymbol{\ell}} + \mathbf{X}_{\hat{\mathbf{m}}}$.
- $\mathsf{WE.Dec}(\mathsf{M}, \mathbf{w})$: Compute $\mathbf{A}_\mathbf{w} = \mathbf{A} + \sum_i w_i \mathbf{B}_i$. If the first $k + \lambda$ columns of $\mathbf{A}_\mathbf{w}$ do not form a matrix of rank exactly $k - 1$, abort. Otherwise, find a rank $k - 1$ submatrix $\mathbf{A}'_\mathbf{w}$ of dimension $k \times (k - 1)$ among the first $k + \lambda$ columns of $\mathbf{A}_\mathbf{w}$. Let $\mathbf{X}$ be the matrix of all zeros except

for a formal variable x in the top right corner. Now, for each $i \in [t]$, let $\hat{m}_i$ be the solution to the linear equation $\det([\mathbf{A}'_{\mathbf{w}} \mid (\mathbf{A}_{\mathbf{w}})_{k+\lambda+i}] - \mathbf{X}) = 0$ over the variable x. Assemble the $\hat{m}_i$ into a vector $\hat{\mathbf{m}} \in \mathbb{F}_p^t$ and recover the message $\mathbf{m} \in \{0,1\}^h$.

To argue correctness of decryption on input a valid witness $\mathbf{w}$, first note that since $\mathbf{A}'_{\mathbf{w}}$ has full rank $k-1$, each equation $\det([\mathbf{A}'_{\mathbf{w}} \mid (\mathbf{A}_{\mathbf{w}})_{k+\lambda+i}] - \mathbf{X}) = 0$ will be linear with a non-zero coefficient on variable x. Hence, each element of $\hat{\mathbf{m}}$ can easily be recovered. It is left to argue that the decryption function aborts with negligible probability. Let $\overline{\mathbf{A}}_{\mathbf{w}}$ be the first $k+\lambda$ columns of $\mathbf{A}_{\mathbf{w}}$. We claim that the rank of $\overline{\mathbf{A}}_{\mathbf{w}}$ is $k-1$, except with negligible probability. We know it is not full rank, since by the properties of the NSS all-accept scheme, the vector $[1 \mid \mathbf{w}] \cdot \mathbf{T}$ must be in its left kernel. Similar to the proof of security of the original NSS all accept, we can union bound over all other possible vectors, concluding that $\overline{\mathbf{A}}_{\mathbf{w}}$ is rank deficient by at least 2 with probability at most $1/p^{\lambda}$, which is negligible in λ, even for constant size fields.

Now we estimate the ciphertext size of the public key encryption scheme that results from combining the optimized witness encryption scheme with our reduction from $\mathsf{GOLDREICH}$ to $\mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$. We will set $\lambda = 80$ to be the security and correctness parameter. Note that for any public key encryption scheme, it suffices to encrypt a security parameter number of bits and then appeal to key encapsulation to encrypt arbitrarily long messages. We set $p = 11$ to be the smallest prime satisfying $p > \max_{\mathbf{x} \in \{0,1\}^n} ||\mathbf{H} \cdot \mathbf{x}||_\infty$ for $\mathbf{H}$ output by the reduction from $\mathsf{GOLDREICH}$.

We will make use of Goldreich's PRG[3] with input length $n = 300$ and output length $m = 300$ [CDM+18]. Plugging in these parameters, the optimized witness encryption will output $n + 2m = 900$ matrices of dimension 900×1007 over the field of size 11. Assuming 4 bits per field element, this comes to a ciphertext of size about 400 MB. Again, the *public key* size will be extremely small: 300 bits. Perhaps even more impressively, the key generation can be done by a Boolean circuit with 300 AND gates and 900 XOR gates. This can be useful for fast distributed generation of keys via secure multiparty computation. We are not aware of any other (practically feasible) public-key encryption schemes whose key generation is nearly as efficient.

7 Cryptanalysis of the Witness Encryption Candidate

Recall that in our generic witness encryption framework of Section 5, we encrypt a bit b with respect to an instance $(\mathbf{h}, \ell)$ by sampling the following 3 ADPs.

- An instance-encoding ADP

$$\mathsf{M}_{\mathbf{h},\ell} = (\mathbf{A}^{(\mathbf{h},\ell)}, \mathbf{B}_1^{(\mathbf{h},\ell)}, \ldots, \mathbf{B}_n^{(\mathbf{h},\ell)}).$$

- A bit-encoding ADP

$$\mathsf{M}_b = (b\mathbf{S}, 0, \ldots, 0).$$

[3] This function is conjectured to be *one-way* when the output length is equal to the input length. Here we use it as a one-way function, which by Claim 2 suffices if our WE candidate is extractable. The latter assumption is only sufficient but not necessary; in fact, the security of this PKE candidate is a relatively clean and falsifiable assumption that can serve as a good target for cryptanalysis.

- An all-accept ADP

$$\mathsf{M}_{\mathsf{AA}} = (\mathbf{A}^{(\mathsf{AA})}, \mathbf{B}_1^{(\mathsf{AA})}, \ldots, \mathbf{B}_n^{(\mathsf{AA})}) \leftarrow \mathsf{AllAcc.Gen}(n, \mathbb{F}_p).$$

The ciphertext is the ADP

$$\mathsf{M}_{\mathbf{h},\ell} + \mathsf{M}_b + \mathsf{M}_{\mathsf{AA}} = (\mathbf{A}^{(\mathbf{h},\ell)} + b\mathbf{S} + \mathbf{A}^{(\mathsf{AA})}, \mathbf{B}_1^{(\mathbf{h},\ell)} + \mathbf{B}_1^{(\mathsf{AA})}, \ldots, \mathbf{B}_n^{(\mathbf{h},\ell)} + \mathbf{B}_n^{(\mathsf{AA})}).$$

A described in Section 5, $\mathsf{M}_{\mathbf{h},\ell} + \mathsf{M}_b$ on its own would be insufficient for security, since given any vector $\mathbf{w} \in \mathbb{F}_p^n$ such that $\mathbf{h} \cdot \mathbf{w} = \ell$, it is possible to recover b. For security, recovering b should only be possible when the vector $\mathbf{w}$ additionally satisfies $\mathbf{w} \in \{0,1\}^n$.

We therefore include the "all-accept" ADP M_{AA} such that for any $\mathbf{x} \notin \{0,1\}^n$, $\det(\mathsf{M}_{\mathsf{AA}}(\mathbf{x})) \neq 0$ with overwhelming probability, and for any $\mathbf{x} \in \{0,1\}^n$, $\det(\mathsf{M}_{\mathsf{AA}}(\mathbf{x})) = 0$. We note that both $\mathsf{M}_{\mathbf{h},\ell} + \mathsf{M}_b$ and M_{AA} on their own are "insecure" in the sense that we can show *randomness recovery* attacks on the procedures used to sample them. However, security relies on the idea that adding these ADPs to each other may block such attacks.

First, we show how to mount a randomness recovery attack on M_{AA}. On its own, this will not constitute an attack on the witness encryption ciphertext. However, we will show that this attack strategy can be extended to a full attack on a simplified version of our concrete witness encryption candidate, without the additional safeguards.

Indeed, to obtain secure witness encryption for VECTOR-SUBSET-SUM, we need semantic security of ciphertexts to hold for *any* instance $(\mathbf{h}, \ell)$ that is not in the language. A toy NO instance to consider is

$$\mathbf{h} = (0, \ldots, 0, 1), \ell = 2.$$

Intuitively, such an instances is particularly vulnerable since the resulting $\mathsf{M}_{\mathbf{h},\ell} + \mathsf{M}_b + \mathsf{M}_{\mathsf{AA}}$ will give out all but 2 of the matrices of M_{AA} in the clear.

Attacks on this instance motivate the concrete candidate given in Section 5, in which we first encode the instance $(\mathbf{h}, \ell)$ into a VECTOR-SUBSET-SUM instance $(\mathbf{H}, \boldsymbol{\ell})$ where $\mathbf{H}$ consists of many linearly independent rows.

7.1 Formula-Based All-Accept

We will consider two attacks on the $\mathsf{AllAcc.Gen}^{\mathsf{FORM}}$ sampling procedure, where the width of each $\mathbf{u}_i, \mathbf{v}_i, \mathbf{s}_i, \mathbf{t}_i$ is set to 1. The second attack in particular will motivate the need to consider a more general width parameter $q(\cdot)$.

7.1.1 Correlated Span Attack

We first show a polynomial-time algorithm that given $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n) \leftarrow \mathsf{AllAcc.Gen}(n, \mathbb{F}_p)$, recovers all $4n$ vectors $\{\mathbf{u}_i, \mathbf{v}_i, \mathbf{s}_i, \mathbf{t}_i\}_{i \in [n]}$ up to scalings. The high-level intuition for the attack is that for any $\mathbf{x} \in \{0,1\}^n$, the matrix $\mathbf{A} + \sum_i x_i \mathbf{B}_i$ is rank-deficient. Moreover, if we look at the kernel of this matrix for many different choices of $\mathbf{x} \in \{0,1\}^n$, correlations between these kernels will allow us to recover the original randomness used to sample $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$. We refer to this as a *correlated span attack*.

More precisely, the algorithm proceeds in the following steps.

1. Observe that $\mathbf{A}$ is rank deficient by 1. Thus, we can find vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ such that $\mathbf{A} \cdot \boldsymbol{\alpha} = \mathbf{0}$ and $\boldsymbol{\beta}^\top \cdot \mathbf{A} = \mathbf{0}$.

2. Observe that with high probability $\boldsymbol{\alpha} \perp \mathbf{v}_i$ for all $i \in [n]$. Similarly, $\boldsymbol{\beta} \perp \mathbf{u}_i$ for all $i \in [n]$.

3. Now to recover $\mathbf{t}_i$ up to a scalar, compute $\boldsymbol{\beta}^\top \cdot \mathbf{B}_i$. Since $\boldsymbol{\beta}$ is perpendicular to $\mathbf{u}_i$ for all $i \in [n]$, $\boldsymbol{\beta}^\top \cdot \mathbf{B}_i = (\boldsymbol{\beta}^\top \cdot \mathbf{s}_i) \cdot \mathbf{t}_i$.

4. Now find a vector $\boldsymbol{\gamma}$ that is perpendicular to $\mathbf{t}_i$ for $i \in [n]$. Then, one can compute $\mathbf{B}_i \cdot \boldsymbol{\gamma}$ to find a vector that is a scalar multiple of $\mathbf{u}_i$. Continuing this way, one can recover all vectors $\mathbf{u}_i, \mathbf{v}_i, \mathbf{s}_i, \mathbf{t}_i$ for $i \in [n]$ up to scalar multiples.

Note that this attack crucially relies on access to the rank deficient matrix $\mathbf{A} + \sum_i x_i \mathbf{B}_i$ for various choices of $x \in \{0,1\}^n$. In the witness encryption candidate, it appears difficult to recover such a matrix when the instance is not in the language SUBSET-SUM, which is crucial for security.

7.1.2 Linearization Attack

Next we discuss linearization-style attacks [KS99, CP02]. At a high level, a linearization attack first models the problem to be solved as a system of non-linear equations. Then, it attempts to find certain structures within the equations that enable solving them with just linear algebra. Generally, one derives new variables and re-expresses the system as a larger system of linear equations. Note that in general, systems of quadratic equations are difficult to solve. However, certain systems may be vulnerable to this style of attack.

For concreteness, we give a particular witness encryption instance that we are interested in attacking. We describe the output of encrypting a bit b under the SUBSET-SUM NO instance $(\mathbf{h}, \ell) = ((0, \ldots, 0, 1), 2)$, without first applying the transformation to VECTOR-SUBSET-SUM instance $(\mathbf{H}, \boldsymbol{\ell})$.

- $\mathsf{WE.Enc}(b, ((0, \ldots, 0, 1), 2))$. Fix a large enough field $\mathbb{F}_p$ and draw at random 4 sets of n vectors of dimension $n+1$: $\{\mathbf{u}_i\}_{i \in [n]}, \{\mathbf{v}_i\}_{i \in [n]}, \{\mathbf{s}_i\}_{i \in [n]}, \{\mathbf{t}_i\}_{i \in [n]}$, along with n^2 scalars $\{c_j^{(i)}\}_{i,j \in [n]}$ over $\mathbb{F}_p$. Also draw uniformly random matrices $\mathbf{R}, \mathbf{S} \in \mathbb{F}_p^{(n+1)\times(n+1)}$. Let

$$\begin{aligned}
\mathbf{A} &= \sum_{i \in [n]} \mathbf{u}_i \mathbf{v}_i^\top - 2\mathbf{R} + b\mathbf{S}, \\
\mathbf{B}_1 &= -\mathbf{u}_1 \mathbf{v}_1^\top + \mathbf{s}_1 \mathbf{t}_1^\top + \sum_{j \in [n]} c_1^{(j)} \mathbf{u}_j \mathbf{t}_j^\top, \\
&\vdots \\
\mathbf{B}_{n-1} &= -\mathbf{u}_{n-1} \mathbf{v}_{n-1}^\top + \mathbf{s}_{n-1} \mathbf{t}_{n-1}^\top + \sum_{j \in [n]} c_{n-1}^{(j)} \mathbf{u}_j \mathbf{t}_j^\top, \\
\mathbf{B}_n &= -\mathbf{u}_n \mathbf{v}_n^\top + \mathbf{s}_n \mathbf{t}_n^\top + \sum_{j \in [n]} c_n^{(j)} \mathbf{u}_j \mathbf{t}_j^\top + \mathbf{R}.
\end{aligned}$$

  Output

$$\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n).$$

First, observe that $\mathbf{A}$ and $\mathbf{B}_n$ are masked by $\mathbf{R}$, so we will attempt to extract information only from $\mathbf{B}_1, ..., \mathbf{B}_{n-1}$. Our goal is to recover vectors $(\mathbf{u}_1, ..., \mathbf{u}_n, \mathbf{s}_1)$. In the end, we will be left with $\Omega(n^6)$ equations of degree 3 over the the $O(n^2)$ variables that constitute these vectors. As discussed at the end of the section, these asymptotics fall in the range where successful linearization attacks may be mounted. Thus, we view this as evidence that the above simplified scheme requires the safeguards described in Section 5.2.

The idea can now be described as follows.

1. Compute $\mathbf{M}_i = \mathbf{B}_1 \cdot \mathbf{B}_i^{-1}$ for $i \in \{2, \ldots, n-1\}$. Denote matrix $\mathbf{P} = [\mathbf{u}_1, ..., \mathbf{u}_n, \mathbf{s}_1]$ and $\mathbf{Q} = [\mathbf{t}_1, ..., \mathbf{t}_n, -\mathbf{v}_1]^\top$. In this notation, $\mathbf{B}_1$ can be written as $\mathbf{P} \cdot \mathbf{D}_1 \cdot \mathbf{Q}$ where $\mathbf{D}_1 \in \mathbb{F}_p^{(n+1)\times(n+1)}$ is:

$$\mathbf{D}_1 = \begin{bmatrix} c_1^{(1)} & 0 & \ldots & 0 & 1 \\ 0 & c_1^{(2)} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & c_1^{(n)} & 0 \\ 1 & 0 & \ldots & 0 & 0 \end{bmatrix}$$

2. Now for $i \in \{2, \ldots, n-1\}$, $\mathbf{B}_i$ can be expressed as $\mathbf{P} \cdot \mathbf{D}_i \cdot \mathbf{Q} + \mathsf{LowRank}_i$. Here, $\mathbf{D}_i$ is described below, and $\mathsf{LowRank}_i$ is a matrix with rank at most 2.

$$\mathbf{D}_i = \begin{bmatrix} c_i^{(1)} & 0 & \ldots & 0 & 0 \\ 0 & c_i^{(2)} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & c_i^{(n)} & 0 \\ 0 & 0 & \ldots & 0 & 0 \end{bmatrix}$$

3. Calculate $\mathbf{D}'_i \coloneqq \mathbf{D}_1^{-1} \cdot \mathbf{D}_i$ as

$$\mathbf{D}'_i = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 \\ 0 & (c_1^{(2)})^{-1} c_i^{(2)} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & (c_1^{(n)})^{-1} c_i^{(n)} & 0 \\ 0 & 0 & \ldots & 0 & 0 \end{bmatrix}$$

4. Now, define $\mathbf{M}_i \coloneqq \mathbf{B}_1^{-1} \cdot \mathbf{B}_i$. Note that the attacker can compute all $\mathbf{M}_i$ for $i \in \{2, \ldots, n-1\}$ in the clear. Observe that

$$\mathbf{Q} \cdot \mathbf{M}_i = \mathbf{Q} \cdot \mathbf{B}_1^{-1} \cdot \mathbf{B}_i = \mathbf{Q} \cdot \mathbf{Q}^{-1} \cdot \mathbf{D}'_i \cdot \mathbf{Q} + \mathbf{Q} \cdot \mathsf{LowRank}_i = \mathbf{D}'_i \cdot \mathbf{Q} + \mathsf{LowRank}'_i,$$

where $\mathsf{LowRank}'_i$ is a matrix of rank at most 2.

5. Defining $\mathbf{Q}'_i := \mathbf{D}'_i \cdot \mathbf{Q}$, we have that $\mathbf{Q} \cdot \mathbf{M}_i - \mathbf{Q}'_i$ has rank at most 2. Then, for any i, letting $\mathbf{Q}$ and $\mathbf{Q}'_i$ be matrices of formal variables, the attacker can set up $O(n^6)$ degree 2 equations over $2n^2$ variables. These arise by setting the determinant of each of the $O(n^6)$ submatrices of size 3×3 in $\mathbf{Q} \cdot \mathbf{M}_i - \mathbf{Q}'_i$ equal to zero.

   We also have more equations that arise from the fact that $\mathbf{Q}'_i$ has the following structure:

$$\mathbf{Q}'_i = \begin{bmatrix} 0 & 0 & \dots & 0 \\ (c_1^{(2)})^{-1} c_i^{(2)} \mathbf{Q}_{2,1} & (c_1^{(2)})^{-1} c_i^{(2)} \mathbf{Q}_{2,2} & \dots & (c_1^{(2)})^{-1} c_i^{(2)} \mathbf{Q}_{2,n+1} \\ \vdots & \vdots & & \vdots \\ (c_1^{(n)})^{-1} c_i^{(n)} \mathbf{Q}_{n,1} & (c_1^{(n)})^{-1} c_i^{(n)} \mathbf{Q}_{n,2} & \dots & (c_1^{(n)})^{-1} c_i^{(n)} \mathbf{Q}_{n,n+1} \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

   Thus, for every $i_1, i_2 \in [2, \dots, n-1]$ and $j_1, j_2 \in [n+1]$, $\mathbf{Q}'_{i,j_1} \cdot \mathbf{Q}_{i,j_2} = \mathbf{Q}'_{i,j_2} \cdot \mathbf{Q}_{i,j_1}$. Similarly, one can set up new equations for every $i \in [n]$. We can further impose that $\sum_i \mathbf{Q}_{i,i} = 1$ to ensure non-triviality.

6. It is not immediately clear that there are enough linearly independent equations to mount an attack. As long as we have a unique solution, Kipnis and Shamir [KS99] suggest that (heuristically) it should be possible to solve for $\mathbf{Q}$ and $\mathbf{D}'_i$ (up to scaling factors).[4] This argument can be generalized to any constant degree d. For a degree 3 system with n^2 variables, we need $0.1 \cdot n^6$ equations. A rudimentary counting argument suggests that if $\mathbf{B}_i$ are random, such a solution should not exist. We leave a more refined analysis of this attack to future work.

This attack suggests that we should set parameters so that LowRank matrices have a super-constant rank, or alternatively, that we do not allow the attacker to see many $\mathbf{B}_i$ matrices in the clear. The construction in Section 5.2 incorporates safeguards that address both, generalizing the width of the $\mathbf{u}_i, \mathbf{v}_i, \mathbf{s}_i, \mathbf{t}_i$ vectors via the parameter $q(\cdot)$, and encoding SUBSET-SUM instances into VECTOR-SUBSET-SUM instances with many linearly independent vectors.

7.2 NSS-Based All-Accept

One potential advantage that the NSS-based construction has over the formula-based construction is in the amount of randomness used to generate a sample. For dimension n, the formula-based sampling procedure uses $4(n+1)n + n^2 = O(n^2)$ uniformly random field elements to produce $n+1$ matrices of dimension $(n+1) \times (n+1)$. On the other hand, the NSS-based sampling procedure generates a fresh NSS matrix for each $i \in [n]$, each of which requires about $n^2/2$ fresh random variables. Thus the sampling procedure overall uses $O(n^3)$ random field elements to produce $n+1$ matrices of dimension $(n+1) \times (n+1)$.

However, the NSS matrices are very structured, which leads to randomness recovery attacks in certain settings, even when the attacker is not given the entire all-accept sample. Consider a sample $(\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n) \leftarrow \mathsf{AllAcc.Gen}^{\mathsf{NSS}}(n, \mathbb{F}_p)$, recalling that the width k of each matrix is $n+1$. The goal will be to recover the random matrix $\mathbf{T}$ drawn during the sampling procedure. First assume

[4] More specifically, for a degree-2 system of equation over n variables, roughly $0.1 \cdot n^2$ linearly independent quadratic equations suffice to recover an over-determined solution in polynomial time [KS99, TW12].

that just $\mathbf{B}_1$ and $\mathbf{B}_2$ are given in the clear. This means we have the 2nd and 3rd column of each $\mathbf{C}_i$ matrix for $i \in [k]$, where $\mathbf{N}_i = \mathbf{T} \cdot \mathbf{C}_i$ and each $\mathbf{N}_i$ is NSS. The symmetric properties of each $\mathbf{N}_i$ immediately give us k linear equations over the $2k$ variables comprising the 2nd and 3rd row of $\mathbf{T}$. Generalizing, if we started with s matrices $\mathbf{B}_i$ in the clear, we could form $\binom{s}{2}$ linear equations over $2s$ variables of $\mathbf{T}$ and eventually the linear system will be over-determined.

We can use this structure to attack the $(\mathbf{h}, \ell) = ((1, 0, \ldots, 0), 2)$ instance of $\mathsf{SUBSET\text{-}SUM}$ as follows (note this is the same instance considered above but relabeled for convenience). Encryption of the bit 0 results in the set of matrices $\mathbf{A}' = \mathbf{A} + 2\mathbf{R}, \mathbf{B}'_1 = \mathbf{B}_1 + \mathbf{R}, \mathbf{B}_2, \ldots, \mathbf{B}_n$, where $(\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n) \leftarrow \mathsf{AllAcc.Gen}^{\mathsf{NSS}}(n, \mathbb{F}_p)$ and $\mathbf{R}$ is a uniformly random matrix. Using $\mathbf{B}_2, \ldots, \mathbf{B}_n$, recover all but the first two rows of $\mathbf{T}$ as described above, and call this matrix $\widetilde{\mathbf{T}} \in \mathbb{F}_p^{(k-2)\times k}$. We will be interested in solving for the first and second rows of $\mathbf{T}$, denoted $\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$, using $\widetilde{\mathbf{T}}$ and the matrices $\mathbf{A}'$ and $\mathbf{B}'_1$.

Note that $\widetilde{\mathbf{T}}$ and $\mathbf{B}_2, \ldots, \mathbf{B}_n$ reveal the bottom right $(k-2) \times (k-2)$ submatrix of each $\mathbf{N}_i$, and in particular the diagonal entries of each of these submatrices. Now let $\mathbf{d}_i$ be the column vector that consists of the final $k-2$ elements on the diagonal of $\mathbf{N}_i$, and arrange the k columns $\mathbf{d}_i$ into a matrix $\widetilde{\mathbf{N}}$ of dimension $(k-2)$ by k. Using the symmetries present in the $\mathbf{N}_i$, we see that

$$\widetilde{\mathbf{T}} \cdot \mathbf{A} = \begin{bmatrix} -\mathbf{t}^{(1)} \cdot \mathbf{B}_2 \\ \vdots \\ -\mathbf{t}^{(1)} \cdot \mathbf{B}_n \end{bmatrix} - \widetilde{\mathbf{N}} \coloneqq \widetilde{\mathbf{T}}^{(1)} - \widetilde{\mathbf{N}} \quad \text{and} \quad \widetilde{\mathbf{T}} \cdot \mathbf{B}_1 = \begin{bmatrix} -\mathbf{t}^{(2)} \cdot \mathbf{B}_2 \\ \vdots \\ -\mathbf{t}^{(2)} \cdot \mathbf{B}_n \end{bmatrix} \coloneqq \widetilde{\mathbf{T}}^{(2)},$$

and we can then compute

$$\widetilde{\mathbf{T}}(\mathbf{A}' + 2\mathbf{B}'_1) = \widetilde{\mathbf{T}}(\mathbf{A} - 2\mathbf{R} + 2\mathbf{B}_1 + 2\mathbf{R}) = \widetilde{\mathbf{T}}(\mathbf{A} + 2\mathbf{B}_1) = \widetilde{\mathbf{T}}^{(1)} - 2\widetilde{\mathbf{T}}^{(2)} - \widetilde{\mathbf{N}}.$$

This gives k^2 linear equations over the $2k$ variables comprising $\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$.

This attack again highlights the danger of giving the adversary many $\mathbf{B}_i$ matrices in the clear. This motivates the need for encoding the $\mathsf{SUBSET\text{-}SUM}$ instance $(\mathbf{h}, \ell)$ into a $\mathsf{VECTOR\text{-}SUBSET\text{-}SUM}$ instance $(\mathbf{H}, \boldsymbol{\ell})$ where $\mathbf{H}$ contains many linearly independent rows, as described in the concrete candidate above. Now an adversary attacking the $(\mathbf{h}, \ell) = ((1, 0, \ldots, 0), 2)$ instance can recover matrices $\mathbf{B}_1 - \mathbf{B}_{n/2+1}, \ldots, \mathbf{B}_{n/2} - \mathbf{B}_n$ in the clear, and about half the information in $\mathbf{T}$, but it is unclear how to use this limited information to mount a full randomness recovery or message distinguishing attack.

8 Candidate Obfuscation for Branching Programs

As discussed in Section 4.3, the ADP model is powerful enough to capture different types of branching programs. In this section, we describe a general paradigm for taking an arbitrary deterministic branching program BP and producing an ADP that is plausibly an *indistinguishability obfuscation* ($i\mathcal{O}$) of BP. Since polynomial-size deterministic branching programs are powerful enough to simulate the class NC^1 (by Barrington's theorem) or even log-space computations, our construction directly yields a candidate $i\mathcal{O}$ for NC^1 and log-space. As shown in [GGH+13b], this suffices to obtain a candidate $i\mathcal{O}$ for all polynomial-size circuits.

8.1 Functionality-Preserving Transformations

In this section, we describe three generic transformations that can be applied to affine determinant programs satisfying specific conditions. Our candidate NC^1 obfuscation will be the result of applying these three transformations in sequence to an ADP encoding of [IK02, AIK04].

8.1.1 Transformation 1: Random Local Substitutions

Recall the ADP encoding of counting branching programs described in Section 4.3. Consider any such branching program, specified by a directed acyclic graph $G = (V, E)$, and fix a topological ordering on the vertices $v_1, \ldots, v_{|V|}$. Each pair of ordered vertices (v_j, v_k), for $j < k$, is labeled by a function x_i, $\neg x_i$, 1, or 0 (no edge). Given such a branching program, our first transformation will perform what we term a *random local substitution* for each vertex pair (v_j, v_k). We describe in Section 9.3 an attack strategy that motivates the need for random local substitutions.

Viewing the branching program as $\mathsf{ADP} = (\mathsf{M}, \mathsf{Eval})$, where $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$, we see that each vertex pair (v_j, v_k), for $j < k$, defines a width-1 ADP. For concreteness, these matrices can be thought to be over some finite field $\mathbb{F}_p$ and Eval is just the identity function. In particular, each pair gives rise to the $(j, k-1)$th entry of each matrix $\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n$, which we denote by $a^{(j,k)}, b_1^{(j,k)}, \ldots, b_n^{(j,k)}$, as follows. If the label is the bit 0 or 1, then $a^{(j,k)}$ is equal to the bit, and all $b_i^{(j,k)} = 0$. If the label is x_i, then $a^{(j,k)} = 0, b_i^{(j,k)} = 1$, and $b_{i'}^{(j,k)} = 0$ for $i' \neq i$. If the label is $\neg x_i$, then $a^{(j,k)} = 1, b_i^{(j,k)} = -1$, and $b_{i'}^{(j,k)} = 0$ for $i' \neq i$.

We will inject entropy into the branching program by replacing each of these width-1 ADPs with a random width-2 ADP computing the same function. There are many possible such local substitution operations and generalizations, but we present here a particularly simple realization, which maintains the property that the resulting matrices have all entries in $\{-1, 0, 1\}$.

In the graph representation of the original branching program, the effect of replacing each each width-1 ADP with a width-2 ADP amounts to adding a vertex $v_{j,k}$ for each pair (v_j, v_k). Thus, if the width of the original ADP was ℓ, the width of the resulting ADP $\mathbf{A}', \mathbf{B}'_1, \ldots, \mathbf{B}'_n$ will be $\ell + \binom{\ell+1}{2}$. With this view it is easy to see what transformations are possible. For any pair (v_j, v_k), we consider the set of 2×2 submatrices of $\mathbf{A}', \mathbf{B}'_1, \ldots, \mathbf{B}'_n$ with rows indexed by $v_j, v_{j,k}$ and columns indexed by $v_{j,k}, v_k$. Denote this set of 2×2 matrices as $\mathbf{A}'^{(j,k)}, \mathbf{B}_1'^{(j,k)}, \ldots, \mathbf{B}_n'^{(j,k)}$. If the label of (v_j, v_k) was 0, we set all $\mathbf{B}_i'^{(j,k)} = \mathbf{0}$, and have the following possibilities for $\mathbf{A}'^{(j,k)}$:

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

If the label of (v_j, v_k) was 1, we set all $\mathbf{B}_i^{(j,k)} = \mathbf{0}$, and have the following possibilities for $\mathbf{A}^{(j,k)}$:

$$\mathbf{A}_1^{(1)} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \mathbf{A}_2^{(1)} = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}, \mathbf{A}_3^{(1)} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}, \mathbf{A}_4^{(1)} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

If the label of (v_j, v_k) was x_i, we can set $\mathbf{A}'^{(j,k)}$ to be $\mathbf{A}_c^{(0)}$ for some $c \in \{1, 2, 3\}$, and $\mathbf{B}_i'^{(j,k)}$ to be $\mathbf{A}_d^{(1)} - \mathbf{A}_c^{(0)}$ for some $d \in \{1, 2, 3, 4\}$. This gives a total of 12 possible substitutions. Finally, if the label of (v_j, v_k) was $1 - x_i$, we again obtain 12 possible substitutions by fixing $\mathbf{A}'^{(j,k)}$ to be $\mathbf{A}_c^{(1)}$ for some $c \in \{1, 2, 3, 4\}$, and $\mathbf{B}_i'^{(j,k)}$ to be $\mathbf{A}_d^{(0)} - \mathbf{A}_c^{(1)}$, for some $d \in \{1, 2, 3\}$.

The operation we perform will, for each vertex pair, pick uniformly at random from the set of possibilities described above. For convenience, we denote this *random local substitution* operation by $\mathsf{ADP}' = \mathsf{RLS}(\mathsf{ADP})$.

We stress that our particular method of performing random local substitutions is only one potential candidate, and there are many possible transformations to explore. Our candidate was designed specifically to thwart attacks on *iO* described in Section 9.3, which take advantage of the fact that the structure of the underlying branching program is known.

8.1.2 Transformation 2: Small Even-Valued Noise

Our next transformation assumes the following about the input $\mathsf{ADP} = (\mathsf{M}, \mathsf{Eval})$.

- $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n)$ is such that each entry of $\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n$ is in $\{-1, 0, 1\}$.
- $\mathsf{Eval} = \mathsf{Eval}_{\neq 0}$, and furthermore, on any input $\mathbf{x} \in \{0,1\}^n$, $\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) \in \{0, 1\}$.

Note that any ADP output by the [IK02, AIK04] encoding and then subjected to the random local substitution above satisfies these properties.

$\mathsf{AddNoise}(\mathsf{ADP})$:

- Let ℓ be the width of the matrices in ADP and n be the input length. Based on n, ℓ, fix a prime modulus p and error distribution χ as explained below. Consider each matrix $\mathbf{A}, \mathbf{B}_i$ as now being over $\mathbb{F}_p^{\ell \times \ell}$.
- Sample matrices $\mathbf{U}, \mathbf{V}$ randomly in $\mathbb{F}_p^{\ell \times \ell}$ such that $\det(\mathbf{U}) \cdot \det(\mathbf{V}) = 1$.
- Sample matrices $\mathsf{Err}_i \leftarrow \chi^{\ell \times \ell}$.
- Set $\mathbf{A}' = \mathbf{U} \cdot (\mathbf{A} + 2 \cdot \mathsf{Err}_0) \cdot \mathbf{V}$ and $\mathbf{B}'_j = \mathbf{U} \cdot (\mathbf{B}_j + 2 \cdot \mathsf{Err}_j) \cdot \mathbf{V}$ for all $j \in [n]$.
- Finally, in the resulting ADP, set $\mathsf{M} = (\mathbf{A}', \mathbf{B}'_1, ..., \mathbf{B}'_n)$ and set $\mathsf{Eval} = \mathsf{Eval}_{\text{parity}}$.

Parameters. We set χ to be the distribution that samples uniformly from the range $[-B(\ell), +B(\ell)]$, where $B(\ell) = \ell^{\omega(1)}$. We set p to be a prime modulus such that p is $\Theta((n \cdot B(\ell) \cdot \sqrt{\ell})^{\ell})$. This can be done by setting the bit length of p as $\ell^{1+\epsilon}$ for any constant $\epsilon > 0$, assuming $\ell > n$.

Correctness. Let $\mathbf{x}$ be any input in $\{0,1\}^n$ and let $L(\mathbf{x}) := \mathbf{A} + \sum_i x_i \mathbf{B}_i$ be the branching program that is being obfuscated. Observe that

$$\begin{aligned}
\det(\mathsf{M}(\mathbf{x})) &= \det(\mathbf{U} \cdot (L(\mathbf{x}) + 2 \cdot \mathsf{Err}_0 + \sum_i x_i \cdot 2 \cdot \mathsf{Err}_i) \cdot \mathbf{V}) \\
&= \det(\mathbf{U}) \det(\mathbf{V}) \det(L(\mathbf{x}) + 2 \cdot \mathsf{Err}_0 + \sum_i x_i \cdot 2 \cdot \mathsf{Err}_i) \\
&= \det(L(\mathbf{x}) + 2(\mathsf{Err}_0 + \sum_i x_i \cdot \mathsf{Err}_i)).
\end{aligned}$$

Correctness follows from the following observations.

- The value of $\det(L(\mathbf{x}) + 2(\mathsf{Err}_0 + \sum_i x_i \cdot \mathsf{Err}_i))$ over the integers is of the form $L(\mathbf{x}) + 2z$ for some $z \in \mathbb{Z}$.

- The number of monomials in the degree ℓ polynomial defined by the determinant is (loosely) bounded by ℓ^ℓ.

- Each entry in $L(\mathbf{x}) + 2(\mathsf{Err}_0 + \sum_i x_i \cdot \mathsf{Err}_i)$ is bounded by $n \cdot B(\ell) + 1$ in absolute value, so the determinant is smaller than $(n \cdot B(\ell) + 1)^\ell \cdot \ell^\ell$ in absolute value. If $p > (n \cdot B(\ell) + 1)^\ell \cdot \ell^\ell$, correctness holds.

One can also carry out a more refined calculation as follows. For a boolean input x, let $Q_x = L(\mathbf{x}) + 2(\mathsf{Err}_0 + \sum_i x_i \cdot \mathsf{Err}_i)$. With overwhelming probability over the choice of x, Q_x has entries bounded by $n^{1/2+\delta} B(\ell)$ in absolute value for any $\delta > 0$. This can be shown using a Chernoff bound. The signs of the entries of this matrix Q_x, are random and independent, so one can use the matrix Bernstein inequality (see for example, Theorem 6.6.1 [Tro15]) to bound the maximum eigenvalue and hence the determinant (as determinant is the product of eigenvalues), to be $(n^{1/2+\delta} \cdot B(\ell) \cdot \ell^{1/2+\delta})^\ell$. This implies that p can be a $\Theta(\ell^{1+\epsilon})$ bit prime for any $\epsilon > 0$.

8.1.3 Transformation 3: Block-Diagonal Matrices

Ideally, the obfuscation of a circuit over n bits should not leak anything other than its input/output behavior on $\mathbf{x} \in \{0,1\}^n$. However, consider evaluating the ADP that results from the above transformation on a short but non-binary input $\mathbf{x}$. Due to the setting of parameters necessary for correctness, the determinant of the matrix $\mathbf{A} + \sum_i x_i \mathbf{B}_i$ will not be large enough to wrap around the modulus. Intuitively, the even-valued noise should ensure that the only useful information gained from this determinant is the evaluation of the circuit on input $\mathbf{x}$ mod 2. However, the fact that the determinant does not wrap around p is nevertheless worrisome, and we present a simple method to potentially block any attacks that might make use of short non-binary evaluations, such as the polynomial extension attacks described in Section 9.1.

The idea will be to post-process any $\mathsf{M} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ in a way that forces the determinant on non-binary inputs to be large and random. This can be accomplished using $2n$ 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 ith slot along the diagonal. After appending, we rerandomize as before with $\mathbf{U}$ and $\mathbf{V}$. The determinant on any binary input will be the product of the determinant of the original ADP times the product of the determinant of $\mathbf{G}_i$ or $\mathbf{H}_i$ for each i, which are both 1. On any non-binary input, some block diagonal will be a linear combination of $\mathbf{G}_i$ and $\mathbf{H}_i$ that results in a large and random determinant. The following transformation also takes as input a parameter d, which determines the size of the random $\mathbf{G}_i, \mathbf{H}_i$ matrices. In our obfuscation construction, it is reasonable to set $d = 2$.

$\mathsf{AddBlockDiagonals}(\mathsf{ADP}, d)$:

- Let ℓ be the width of the matrices in ADP and n be the input length.

- Sample $2n$ matrices $\{\mathbf{G}_i, \mathbf{H}_i\}_{i \in [n]}$ uniformly from $\mathbb{F}_p^{d \times d}$ conditioned on their determinant being equal to 1.

- Sample $\mathbf{U}, \mathbf{V}$ uniformly from $\mathbb{F}_p^{(\ell+nd)\times(\ell+nd)}$ conditioned on $\det(\mathbf{U}) \cdot \det(\mathbf{V}) = 1$.

- Set $\mathbf{A}' = \mathbf{U} \cdot \mathsf{diag}(\mathbf{A}, \mathbf{G}_1, \ldots, \mathbf{G}_n) \cdot \mathbf{V}$, and $\mathbf{B}'_i = \mathbf{U} \cdot \mathsf{diag}(\mathbf{B}_i, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{H}_i - \mathbf{G}_i, \mathbf{0}, \ldots, \mathbf{0}) \cdot \mathbf{V}$, and return the resulting ADP, consisting of $\mathsf{M} = (\mathbf{A}', \mathbf{B}'_1, \ldots, \mathbf{B}'_n)$ and $\mathsf{Eval} = \mathsf{Eval}_{\text{parity}}$.

In summary, to obfuscate a deterministic (alternatively, $\mathbb{F}_2$ counting) branching program, we first convert it into an ADP as described in Section 4.3. We then output AddBlockDiagonals(AddNoise(RLS(ADP)), 2) as the final obfuscation.

8.2 Extensions of the Basic Construction

8.2.1 Obfuscating Affine $\mathbb{F}_2$ Counting Branching Programs

Motivated by the goal of improved concrete efficiency, we extend the branching program model to allow labeling of edges by any *affine function* over the input bits. This is captured by the following notion of Affine $\mathbb{F}_2$ Counting Branching Programs.

Definition 11 (Affine Counting Branching Programs [IK02])**.** An Affine $\mathbb{F}_2$ Counting Branching Program computing $f : \{0,1\}^n \to \{0,1\}$ is specified by a directed acyclic graph $G = (V, E)$ and a labeling $\phi(\cdot,\cdot)$ where each edge $(u, v) \in E$ is labeled with an affine function $\phi(u, v) = f_{u,v}(\mathbf{x})$, and two special source and sink vertices are labeled s and t respectively. $f_{u,v}$ has the form

$$f_{u,v}(\mathbf{x}) = \sum_{i \in S_{u,v}} x_i + c_{u,v} \mod 2,$$

where $c_{u,v} \in \mathbb{F}_2$. Its size is $|V| - 1$. Any input $\mathbf{x} \in \{0,1\}^n$ induces a sub-graph $G_{\mathbf{x}}$ limited to edges consistent with $\mathbf{x}$ (i.e. edges that evaluate to 1 on $\mathbf{x}$). An accepting path on input $\mathbf{x}$ is a directed $s - t$ path in $G_{\mathbf{x}}$. An Affine $\mathbb{F}_2$ Counting Branching Program computes the function $g : \{0,1\}^n \to \{0,1\}$ such that $g(\mathbf{x})$ is the number of accepting paths in $G_{\mathbf{x}}$ mod 2.

We now import the following theorem.

Lemma 4 ([IK02, AIK04])**.** *Suppose there is an Affine $\mathbb{F}_2$ Counting Branching Program of size ℓ computing a boolean function f. Suppose $L(\mathbf{x})$ satisfies the following*

- *$L(\mathbf{x})$ has -1 along the second diagonal (right below the main diagonal).*
- *$L(\mathbf{x})$ is 0 below the second diagonal.*
- *Each entry of $L(\mathbf{x})$ is a degree (at most) 1 polynomial in a single input variable x_i.*

  More precisely, $L(\mathbf{x})$ is defined as follows. Fix a topological ordering of the vertices in V, and label the columns / rows (from left to right / top to bottom) according to this ordering of vertices. In particular we want s labeled 1 and t labeled ℓ. We first define a matrix $A(\mathbf{x})$ of dimension $\ell \times \ell$. For entry (i, j) entry of $A(\mathbf{x})$ write affine function $\phi(i, j)$ (written as a degree one polynomial over the reals) if (i, j) is an edge in G and 0 otherwise. Note that $A(\mathbf{x})$ will be 0 on and below the main diagonal. Now consider $A(\mathbf{x}) - I$, and delete its first column and last row to obtain the $(\ell - 1) \times (\ell - 1)$ dimensional matrix $L(\mathbf{x})$.

Then for all $\mathbf{x} \in \{0,1\}^n$, we have $\det(L(\mathbf{x})) \mod 2 = f(\mathbf{x})$.

Observe that this class of branching programs contain $\mathbb{Z}$ branching programs, since the evaluation of a $\mathbb{Z}$ branching program is exactly its evaluation modulo 2. What we observe is that in this case, our obfuscation scheme described in Section 8 is already capable of obfuscating affine $\mathbb{F}_2$ counting branching programs. The idea is that we can apply transformation 2 given in Section 8 to the matrix $L(\mathbf{x})$ representing an Affine $\mathbb{F}_2$ Counting Branching Program as follows: Each edge in $L(\mathbf{x})$ is an affine function of the form $f(\mathbf{x}) = \sum_{i \in S} x_i + c \mod 2$. We construct a new matrix $L'(\mathbf{x})$ where the edge is replaced by $f'(\mathbf{x}) = \sum_{i \in S} x_i + c$, and the computation is now over integers. The range of this new affine function is $[0, n+1]$. If the parameters are chosen appropriately, what we end up computing by the evaluation procedure is $\det(L'(\mathbf{x})) \bmod 2$. We set p so that this can be computed without a wrap-around. Note that $\det(L(\mathbf{x})) \mod 2 = \det(L'(\mathbf{x})) \bmod 2$ as the error that is added to the matrices is even. This ensures correctness.

In the next section, we refine the notion of random local substitution and propose another alternative safeguard that relies on the ideas developed in this section.

8.2.2 Using Embedded Affine $\mathbb{F}_2$ Branching Programs

In Section 8.1.1, we discussed the notion of random local substitutions, which is our safeguard for preventing attacks such as the one described in Section 9.3. However, that solution incurs a quadratic blow up in the size of the branching program. In this section, we propose an alternate safeguard. Let $\mathsf{BP} : \{0,1\}^n \to \{0,1\}$ be a $\mathbb{Z}$ branching program with ℓ vertices, including source s and target t. First, sample a $\mathsf{BP}' : \{0,1\}^n \to \{0,1\}$ as follows:

- Fix ℓ vertices, and select two vertices s', t' among these ℓ vertices to act as the source vertex and target vertex of BP'.
- Induce a random directed acyclic graph among these vertices. This is done by connecting all vertices $i \in [\ell]$ to vertices j where $j \geq i+1$. In this numbering, the starting vertex s' is the 1st vertex and t' is the ℓ^{th} vertex. For every edge u, v sample a random affine function $f_{u,v}(\mathbf{x})$ which is used to label the edges.

We now combine BP and BP' into the final transformed BP'' as follows. BP'' consists of $2\ell + 2$ vertices: the ℓ vertices of BP, the ℓ vertices of BP', and a new source s'' and target t''. Form edges from s'' to s, from s'' to s', and from t to t'', all with label 1. Note that there will be no edge between t' and t''. Observe that BP'' computes the same function as BP'. Indeed, any additional paths that go via BP' are never connected to the target vertex t''. There can be either 0 or 1 path going via BP since we started with a deterministic $\mathbb{Z}$ branching program. This resulting new BP'' can now be input to transformations 2 and 3 outlined in Section 8.

9 Cryptanalysis and Parameter Estimation for the Branching Program Obfuscation Candidate

9.1 Polynomial Extension Attacks

As an instructive example, consider the ADP obfuscation of a point function $\mathcal{I}_v$, without the even-valued error. Recall that on input $\mathbf{x} \in \{0,1\}^n$, $\mathcal{I}_v$ outputs 1 if $\mathbf{x} = \mathbf{v}$ and 0 otherwise. A simple branching program computing $\mathcal{I}_v$ contains $n+1$ vertices, with a single edge from each vertex i to vertex $i+1$ that is either set to v_i or $1 - v_i$. Thus, $L(\mathbf{x})$ can be written as the following matrix:

$$\begin{bmatrix} 1+2v_1x_1-v_1-x_1 & 0 & \dots & \dots & 0 \\ -1 & 1+2v_2x_2-v_2-x_2 & \dots & \dots & 0 \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -1 & 1+2v_nx_n-v_n-x_n \end{bmatrix}$$

Hence, for $\mathbf{x} \in \{0,1\}^n$, $\det(L(\mathbf{x}))$ is 1 if $\mathbf{x} = \mathbf{v}$ and 0 otherwise. Writing

$$L(\mathbf{x}) = L_0 + x_1 L_1 + \ldots + x_n L_n,$$

we see that an attacker given

$$\mathsf{ADP} = (\mathbf{A} = \mathbf{U} \cdot L_0 \cdot \mathbf{V}, \mathbf{B}_1 = \mathbf{U} \cdot L_1 \cdot \mathbf{V}, ..., \mathbf{B}_n = \mathbf{U} \cdot L_n \cdot \mathbf{V}),$$

can evaluate

$$\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) = \det(\mathbf{U} \cdot L(\mathbf{x}) \cdot \mathbf{V}) = \det(L(\mathbf{x})).$$

Thus, the attacker has input-output access to the polynomial $\det(L(\mathbf{x}))$ over the field $\mathbb{F}_p$, and is not restricted to evaluation on binary inputs $\mathbf{x} \in \{0,1\}^n$. This leads to the following attack.

1. Observe that $\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i)$ is the unique multilinear polynomial over $\mathbb{F}_p^n$ agreeing with $\prod_{i\in[n]}(1 + 2v_ix_i - v_i - x_i)$ on inputs from $\{0,1\}^n$. This implies

$$\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) = \prod_{i\in[n]}(1 + 2v_ix_i - v_i - x_i)$$

   over $\mathbb{F}_p^n$.

2. To recover v_1, compute

$$\det(\mathbf{A} + x_1 \cdot \mathbf{B}_1 + \sum_{i\geq 2} 2^{-1}\mathbf{B}_i) = (1-2^{-1})^{n-1}(1 + 2v_1x_1 - v_1 - x_1).$$

   We can find $x_1 = 1 - v_1$ by equating this quantity to 0. This recovers the first bit of the point function, and the others can be computed in the same way.

The reason that this particular attack succeeds is that the polynomial extending the boolean function $\det(L(\mathbf{x}))$ is *multilinear*. Read-once branching programs give rise to such polynomials. More generally, Klivans and Shpilka [KS03] showed that *any* read-once, oblivious branching program can be learned efficiently. In their model, the attacker is given membership and equivalence query access to an unknown polynomial P computed by a read once branching program over a field $\mathbb{F}_p$. This means the attacker is able to evaluate the program on any input $\mathbf{x} \in \mathbb{F}_p^n$, as well as submit a hypothesis H and learn that either H is equal to P or receive a point $\mathbf{y}$ on which $H(\mathbf{y}) \neq P(\mathbf{y})$. In our setting, we just argued that the attacker has membership query access to the polynomial computed by the branching program, and equivalence queries can be simulated by evaluation on random points and appealing to Schwartz-Zippel.

For our construction to satisfy the security notion of indistinguishablity obfuscation, we need to be able to successfully obfuscate simple classes of functions computable by read once branching programs. Thus, we have to include some noise in the obfuscation procedure that destroys the read once nature of any such program. One can view the random even-valued error as adding an edge between every pair of vertices in the branching program, labeled with a random, small, even linear combination of the entire input vector. The resulting program is very far from read-once, as evaluating every edge requires reading the entire input. Thus, learning results that apply to restricted classes on branching programs will not apply. As an example, [KS03] state that their techniques would not apply to a polynomial as simple as $f(x_1, \dots, x_n) = \prod_{i=2}^{n}(x_1 + x_i)$.

9.2 The Need for Super-polynomial Error and Modulus

Now we present a contrived example consisting of two ADPs which will motivate the need for super-polynomial noise for transformation 2. While these specific ADPs will not arise from the [IK02] transformation, we will set our error-size to err on the side of caution.

The attack works as follows. Suppose we have an ADP $\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n$ that is in one of the two following forms.

- In the first form, $\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n = \mathbf{0}$.
- In the second form, $\mathbf{A} = \mathbf{0}$, and $\mathbf{B}_i$ for $i \in [n]$ is the matrix

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}.$$

Now consider applying the $\mathsf{AddNoise}$ operation to one of the ADPs, hoping that the choice of ADP is masked by this operation. Assume that the even-valued error added by the $\mathsf{AddNoise}$ operation is actually chosen in $\{-2, 2\}$, so that each entry of the matrices $\mathsf{Err}_0, \mathsf{Err}_1, \dots, \mathsf{Err}_n$ is uniform in $\{-1, +1\}$. Observe that in both cases $\det(\mathbf{A} + \sum_i x_i \mathbf{B}_i) = 0$ for all $\mathbf{x} \in \{0, 1\}^n$, so the parity of the determinant will not reveal which ADP was chosen. A relevant result of [TV06] states that the determinant of a matrix where the entries are chosen uniformly at random from $\{-1, +1\}$ concentrates in absolute value around $\sqrt{n!}$. This fact can be used to estimate the determinant of the matrix $\mathbf{A} + \sum_i \mathbf{B}_i$ in both cases, which is the ADP evaluated on input $x = [1, \dots, 1]$.

Case 1: The value $\det(\mathbf{A} + \sum_i \mathbf{B_i})$ is exactly $\det(2(\mathsf{Err}_0 + \sum_i x_i \cdot \mathsf{Err}_i))$, whose entries are independently signed, with standard deviation about $\sqrt{n}$. Thus, the determinant will concentrate around $\sqrt{n!} \cdot (\sqrt{n})^n$ in absolute value.

Case 2. To estimate the value $\det(\mathbf{A} + \sum_i \mathbf{B_i})$ in this case, we apply a Chernoff bound and find that the elements in the top row are typically in the range $[n - \sqrt{n}\log^2 n, n + \sqrt{n}\log^2 n]$. If we divide the top row by $\sqrt{n}$, we are left with a matrix whose entries are distributed as the matrix in case 1, with the only difference being that the top row is positively signed. Heuristically, this will result in $\det(\mathbf{A} + \sum_i \mathbf{B_i})$ being a $\sqrt{n}$ factor larger.

Thus, the magnitude of the determinant on input $[1, \dots, 1]$ will give a distinguisher. Note however that neither ADP considered here is a valid [IK02, AIK04] encoding of a branching program. We chose these particular ADPs for simplicity, to illustrate the need for super-polynomial error; similar attacks can be carried out on ADPs that result from encodings of real branching programs.

Setting the Modulus. Once we fix the noise bound $B(\ell)$, we can choose p in a more refined manner as follows. Let $L = (L_0, ..., L_n)$ be the branching program to obfuscate. We would like to add noise to the matrices of L to form an ADP $(\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n)$, in such a way that for all $y_0, y_1, ..., y_n \in \{0, 1\}$,

$$\left(\det(y_0\mathbf{A} + \sum_i y_i\mathbf{B}_i) \mod p\right) \mod 2 = \det(y_0L_0 + \sum_i y_iL_i) \mod 2.$$

Let $t = \mathsf{poly}(\ell)$ be a parameter, and set $p = \Theta(t^\ell \cdot B(\ell)^\ell \cdot \sqrt{\ell!} \cdot \ell)$. Then we observe the following:

- If $t \geq n^2$, then with overwhelming probability we can correctly evaluate any boolean combination $y_0, y_1, ..., y_n \in \{0, 1\}^{n+1}$. To see this, first note that the entries of $y_0\mathbf{A} + \sum_i y_i\mathbf{B}$ are randomly signed, and typically lie in $[-2 \cdot \sqrt{n+1}\log^2 n \cdot B(\ell), +2 \cdot \sqrt{n+1}\log^2 n \cdot B(\ell)]$ due to a Chernoff bound. Then, we can appeal to Theorem 1.1 from [TV06], stating that the determinant of any matrix $\mathbf{M}$ of dimension ℓ, with entries chosen uniformly at randomly from $\{-1, +1\}$, satisfies

$$\sqrt{\ell!} \cdot e^{-c\sqrt{\ell \log \ell}} \leq |\det(\mathbf{M})| \leq \sqrt{\ell!} \cdot \ell,$$

  with overwhelming probability. Here $c > 0$ is any constant. In this case, the determinant will thus heuristically satisfy

$$|\det(y \cdot \mathbf{A} + \sum_i y_i \cdot \mathbf{B}_i)| \leq (2 \cdot B(\ell) \cdot \sqrt{n+1}\log^2 n)^\ell \sqrt{\ell!} \cdot \ell \leq p.$$

- Now, consider more general (non-binary) linear combinations $(y_0, y_1, \dots, y_n)$. If the ℓ_1 norm of the combination satisfies $|y_0| + \sum_i |y_i| > t^3$, then it holds that

$$|\det(y_0\mathbf{A} + \sum_i y_i\mathbf{B}_i)| \geq p.$$

  This overflow will hopefully help with security against attacks that evaluate the ADP on non-binary inputs. We can again observe heuristically from theorem 1.1 in [TV06]. If the weight is greater than t^3, then typically we expect each entries of the matrix to be randomly signed and having values larger than $t^{1.4} \cdot B(\ell)$. Thus with high probability,

$$|\det(y_0\mathbf{A} + \sum_i y_i\mathbf{B}_i)| \geq (t^{1.4} \cdot B(\ell))^\ell \sqrt{\ell!} \cdot e^{-c\sqrt{\ell \log \ell}} \geq p.$$

We set t to be n^2 to allow correct evaluation of boolean inputs. The above describes that setting the modulus appropriately can protect against attacks that involves evaluating programs on inputs of large (polynomial) weight. If the weight of the combination is bounded by some polynomial t, we claim the following.

Claim 3. *Fix any* $\mathsf{ADP}\ \mathbf{A}, \mathbf{B}_1, ..., \mathbf{B}_n$ *of width* ℓ *with entries in* $\{-1, 0, 1\}$*. For any* $x = (x_0, ..., x_n) \in \mathbb{Z}^{n+1}$*, Let* $L(\mathbf{x}) = x_0\mathbf{A} + \sum_i x_i\mathbf{B}_i$*, and let* $\mathsf{ADP}' = \mathsf{AddNoise}(\mathsf{ADP})$*. Let* $\mathsf{Err}_0, \mathsf{Err}_1, \ldots, \mathsf{Err}_n$ *be the matrices drawn during the real* $\mathsf{AddNoise}$ *procedure (with a super-polynomial bound), and for* $x \in Z^{n+1}$*, let* $\mathsf{Err}(x) \coloneqq x_0 \cdot 2\mathsf{Err}_0 + \sum_i x_i \cdot 2\mathsf{Err}_i$*. Then for any* $x \in \mathbb{Z}^{n+1}$ *where* $\|x\|_1 < t$*,*

$$\Pr\left[\frac{|\det(x_0\mathbf{A}' + \sum_i x_i\mathbf{B}'_i)| - |\det(L(\mathbf{x})) \mod 2 + \det(\mathsf{Err}(x))|}{|\det(\mathsf{Err}(x))|} \leq \mathsf{negl}(\ell)\right] \geq 1 - \mathsf{negl}(\ell).$$

where the probability is taken over the randomness of generating $\mathsf{Err}_0, \mathsf{Err}_1, \ldots, \mathsf{Err}_n$*.*

This also shows that if the error is chosen from a super-polynomially large error distribution, then a distinguishing attack which only uses the magnitude of $\det(\mathsf{M}(\mathbf{x}))$ will not apply.

Proof. This follows from a result of Ipsen and Rehman [IR08] who showed that for any non-singular matrices $\mathbf{M}_1, \mathbf{M_2} \in \mathbb{R}^{\ell\times\ell}$,

$$|\det(\mathbf{M}_1 + \mathbf{M}_2) - \det(\mathbf{M}_1)|/|\det(\mathbf{M_1})| \leq \cdot \left(\kappa \cdot \frac{\|\mathbf{M}_2\|_2}{\|\mathbf{M}_1\|_2} + 1\right)^{\ell} - 1,$$

where $\|\mathbf{M}\|_2$ is the largest singular value of the matrix $\mathbf{M}$ and $\kappa \coloneqq \|\mathbf{M}_1\|_2 \cdot \|\mathbf{M}_1^{-1}\|_2$ (κ is sometimes referred to as the condition number of $\mathbf{M}_1$). For a random matrix of size $\ell \times \ell$, the condition number is around $\Theta(\ell)$ in expectation[CD]. Now, we can substitute $\mathbf{M}_1$ as $\mathsf{Err}(x)$, $\mathbf{M}_2$ as $L(\mathbf{x})$. The maximum singular value of $\mathsf{Err}(x)$ will be at least $\Omega(B(\ell)\sqrt{\ell})$ (see Exercise 14, [Tao]). Since branching programs have small entries, their singular values are also bounded by $\|x\|_1 \cdot O(\ell) = O(t \cdot \ell)$. Thus, the right-hand side of the previous equation is

$$\left(\kappa \cdot \frac{\|L(\mathbf{x})\|_2}{\|\mathsf{Err}(x)\|_2} + 1\right)^{\ell} - 1 = \left(O(\ell) \cdot \frac{O(t \cdot \ell)}{\Omega(B(\ell) \cdot \sqrt{\ell})} + 1\right)^{\ell} - 1.$$
$$\approx O(\ell^2 \cdot t \cdot B(\ell)^{-1})$$

Since $B(\ell)$ is super-polynomial in ℓ, this concludes the argument.

□

9.3 The Need for Random Local Substitutions

Consider any fixed branching program $L(\mathbf{x}) = \mathbf{A} + \sum_i x_i\mathbf{B}_i$ of width ℓ. Suppose we obfuscate $L(\mathbf{x})$ but skip the random local substitution step. Let $\mathsf{Err}^{(0)}, \mathsf{Err}^{(1)}, \ldots, \mathsf{Err}^{(n)}$ be the $n+1$ error matrices drawn by the $\mathsf{AddNoise}$ transformation. Write each $\mathsf{Err}^{(i)}_{j,k} = 2e^{(i)}_{j,k}$. We can recover the parity of each $e^{(i)}_{j,k}$ as follows. Let $\mathbf{A}', \mathbf{B}'_1, \ldots, \mathbf{B}'_n$ be the ADP after obfuscation. For any input $\mathbf{x}$,

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

where $L(\mathbf{x})_{(j,k)}$ is the (j,k) minor of $L(\mathbf{x})$. Note that everything is known except the parities of $e^{(i)}_{j,k}$, so this gives a linear equation mod 2 over these $(n+1)\ell^2$ parities. We have an exponential number of equations of this form, and can eventually obtain $(n+1)\ell^2$ linearly independent equations.

This readily gives an attack on $i\mathcal{O}$. To distinguish between two known and functionally equivalent branching programs L_1, L_2, simply run the above attack assuming the underlying BP is L_1 and see if there exists a solution or not. Now, the random local substitution is meant to randomize the underlying branching program $L(\mathbf{x})$, so an attacker does not know all of the $L(\mathbf{x})_{(j,k)}$ values. Without these, the attacker cannot set up and solve the above system of linear equations.

10 Applications and Future Work

In future work, we aim to explore applications of optimized variants of our candidates. We give some of the more promising directions below. Since our focus here is on concrete efficiency, we make a heuristic leap of faith of treating our $i\mathcal{O}$ candidates as *ideal* obfuscation schemes for the purpose of these applications.

Concretely Efficient Obfuscation for Circuits. If the safeguards described are secure, then in order to obfuscate a branching program of size ℓ, we need about $O(\ell^{4+\epsilon})$ bits for any $\epsilon > 0$. In concrete terms, if the size of the branching program exceeds 2^{13} vertices, the size is already around 100 terabytes. This motivates the study of efficient bootstrapping mechanisms and simple "obfuscation complete" families of branching programs. In particular, we would like to understand if the SNARG-based bootstrapping approach from [BISW17] or the PRF-based approach from [GIS+10, App14] are practically feasible for our candidates.

Obfuscating PRFs. Efficiently obfuscating PRFs and simple computations that employ them is a highly desirable goal for both theoretical and practical applications. For instance, the bootstrapping theorems of [GIS+10, App14] reduce the obfuscation of a circuit to multiple obfuscations of simple functions that each use a constant number of PRF calls. This and other applications motivate PRF candidates that have small *affine* branching programs, which can be efficiently handled by our construction. One such candidate could be based on the work of [BIP+18]. They propose a weak PRF[5] candidate based on Learning With Rounding (LWR) modulo a constant-size composite, which can be computed by a linear-size deterministic branching program. They also suggest a heuristic for converting a weak PRF into a strong one by evaluating the weak PRF on a suitable encoding of the input; for this particular weak PRF candidate, a linear encoding over a prime that does not divide the LWR modulus seems like a natural choice. For instance, one could use a linear code over $\mathbb{F}_2$ and LWR modulo 15 or linear code over $\mathbb{F}_3$ and LWR modulo 10. This approach yields (strong) PRF candidates that can be evaluated by a linear-size *affine* branching program.

Optimally-Succinct Non-Interactive Arguments. The ability to efficiently obfuscate a PRF would give several compelling applications. One example, due to Sahai and Waters [SW14], is a succinct non-interactive argument (SNARG) with proof length that is the best one could hope for; namely, an s-bit proof suffices to obtain (roughly) 2^{-s} soundness error.

[5] A weak PRF is only guaranteed to be indistinguishable from a random function given its evaluation on *uniformly random* points (as opposed to on adversarially chosen points, as in the usual definition of a PRF).

To give slightly more detail, we briefly recall the obfuscation-based SNARG of [SW14]. Given a PRF and relation circuit $R_{\mathcal{L}}$ for language $\mathcal{L}$, the crs consists of the obfuscations of two programs $C_{\mathcal{P}}$ and $C_{\mathcal{V}}$. $C_{\mathcal{P}}$ takes as input an instance witness pair (x, w) and outputs $\mathsf{PRF}_k(x)$ (where k is a hard-coded PRF key) if and only if $\mathcal{R}_{\mathcal{L}}(x, w) = 1$. $C_{\mathcal{V}}$, which has the same k hard-coded, takes as input (x, π) and outputs 1 if and only if $\mathsf{PRF}_k(x) = \pi$. A prover wishing to prove that $x \in \mathcal{L}$ simply evaluates $\mathsf{Obf}(C_{\mathcal{P}})$ on (x, w) to obtain $\pi = \mathsf{PRF}_k(x)$. The verifier on input x and proof π runs $\mathsf{Obf}(C_{\mathcal{V}})$ on (x, π) and accepts if the output is 1.

If the obfuscation is an *ideal* obfuscation, then forging a proof on $x \notin \mathcal{L}$ requires predicting $\mathsf{PRF}(k, x)$. If the PRF is exponentially-secure, then the SNARG soundness error is (within polynomial factors of) 2^{-s}.

Acknowledgments

YI was supported by ERC Project NTSC (742754), NSF-BSF grant 2015782, BSF grant 2018393, and a joint Israel-India grant. MZ, FM, AS, and AJ were supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C- 0205. MZ and FM were also supported under NSF grant 1616442. AS and AJ were also supported in part by a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. AJ was also supported by a Google PhD Fellowship in Privacy and Security. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, the U.S. Government or Google.

References

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

[AIK07] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 92–110. Springer, Heidelberg, August 2007.

[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 Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2019, Part III*, LNCS, pages 284–332. Springer, Heidelberg, August 2019.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 162–172. Springer, Heidelberg, December 2014.

[App16] Benny Applebaum. Cryptographic hardness of random local functions - survey. *Computational Complexity*, 25(3):667–722, 2016.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *18th ACM STOC*, pages 1–5. ACM Press, May 1986.

[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.

[BIP$^+$18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Heidelberg, November 2018.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 247–277. Springer, Heidelberg, April / May 2017.

[BKM$^+$18] Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. A simple obfuscation scheme for pattern-matching with wildcards. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 731–752. Springer, Heidelberg, August 2018.

[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.

[CD] Zizhong Chen and Jack J. Dongarra. Condition numbers of gaussian random matrices. *SIAM J. Matrix Analysis Applications*, 27(3):603–620.

[CDM$^+$18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of Goldreich's pseudorandom generator. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 96–124. Springer, Heidelberg, December 2018.

[CLT13] Jean-Sébastien Coron, Tancrède 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.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 267–287. Springer, Heidelberg, December 2002.

[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.

[GGH$^+$13c] 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, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 479–499. Springer, Heidelberg, August 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.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, February 2010.

[GJK18] Craig Gentry, Charanjit S. Jutla, and Daniel Kane. Obfuscation using tensor products. Cryptology ePrint Archive, Report 2018/756, 2018. https://eprint.iacr.org/2018/756.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. http://eprint.iacr.org/2000/063.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 174–183. IEEE, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256. Springer, Heidelberg, July 2002.

[IR08] Ilse C. F. Ipsen and Rizwana Rehman. Perturbation bounds for determinants and characteristic polynomials. *SIAM J. Matrix Analysis Applications*, 30(2):762–776, 2008.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 19–30. Springer, Heidelberg, August 1999.

[KS03] Adam Klivans and Amir Shpilka. Learning arithmetic circuits via partial derivatives. volume 2777, pages 463–476, 01 2003.

[Sah19] Amit Sahai. New roads to cryptopia: A research agenda. New Roads to Cryptopia Workshop at CRYPTO 2019, 2019.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Tao] Terrence Tao. Operator norm of a random matrix. Wordpress Bog. https://terrytao.wordpress.com/2010/01/09/254a-notes-3-the-operator-norm-of-a-random-matrix/#utail-wigner.

[Tro15] Joel A. Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends in Machine Learning*, 8(1-2):1–230, 2015.

[TV06] Terence Tao and Van Vu. On random &plusmn;1 matrices: Singularity and determinant. *Random Struct. Algorithms*, 28(1):1–23, January 2006.

[TW12] Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, pages 156–171, 2012.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 439–467. Springer, Heidelberg, April 2015.