Testing Equivalence to Design Polynomials

Omkar Baraskar
Indian Institute of Science, Bengaluru, India

Agrim Dewan
Indian Institute of Science, Bengaluru, India

Chandan Saha
Indian Institute of Science, Bengaluru, India

Abstract

An n-variate polynomial g of degree d is a (n, d, t) *design polynomial* if the degree of the gcd of every pair of monomials of g is at most $t - 1$. The power symmetric polynomial $\mathsf{PSym}_{n,d} := \sum_{i=1}^{n} x_i^d$ and the sum-product polynomial $\mathsf{SP}_{s,d} := \sum_{i=1}^{s} \prod_{j=1}^{d} x_{i,j}$ are instances of design polynomials for $t = 1$. Another example is the Nisan-Wigderson design polynomial NW, which has been used extensively to prove various arithmetic circuit lower bounds. Given black-box access to an n-variate, degree-d polynomial $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$, how fast can we check if there exist an $A \in \mathrm{GL}(n, \mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^n$ such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial? We call this problem "testing equivalence to design polynomials", or alternatively, "equivalence testing for design polynomials".

In this work, we present a randomized algorithm that finds $(A, \mathbf{b})$ such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial, if such A and $\mathbf{b}$ exist, provided $t \leq d/3$. The algorithm runs in $(nd)^{O(t)}$ time and works over any sufficiently large $\mathbb{F}$ of characteristic 0 or $> d$. As applications of this test, we show two results – one is structural and the other is algorithmic. The structural result establishes a polynomial-time equivalence between the graph isomorphism problem and the polynomial equivalence problem for design polynomials. The algorithmic result implies that Patarin's scheme (EUROCRYPT 1996) can be broken in quasi-polynomial time if a random sparse polynomial is used in the key generation phase.

We also give an efficient learning algorithm for n-variate random affine projections of multilinear degree-d design polynomials, provided $n \geq d^4$. If one obtains an analogous result under the weaker assumption "$n \geq d^\epsilon$, for any $\epsilon > 0$", then the NW family is *not* VNP-complete unless there is a VNP-complete family whose random affine projections are learnable. It is not known if random affine projections of the permanent are learnable.

The above algorithms are obtained by using the vector space decomposition framework, introduced by Kayal and Saha (STOC 2019) and Garg, Kayal and Saha (FOCS 2020), for learning non-degenerate arithmetic circuits. A key technical difference between the analysis in the papers by Garg, Kayal and Saha (FOCS 2020) and Bhargava, Garg, Kayal and Saha (RANDOM 2022) and the analysis here is that a certain adjoint algebra, which turned out to be trivial (i.e., diagonalizable) in prior works, is non-trivial in our case. However, we show that the adjoint arising here is triangularizable which then helps in carrying out the vector space decomposition step.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Polynomial equivalence, design polynomials, graph isomorphism, vector space decomposition

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.9

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/004/` [7]

Funding *Chandan Saha*: Partially supported by a MATRICS grant of the Science and Engineering Research Board, DST, India.

Acknowledgements We thank the reviewers for their valuable feedback, which has helped us improve the presentation of this work. In particular, we thank one of the reviewers who pointed us to appropriate citations for the adjoint algebra.

1 Introduction

The polynomial equivalence problem (PE) is fundamental in algebraic complexity theory. Given two polynomials, f and g, how fast can we check if one is in the orbit of the other? *Orbit* of a polynomial $f \in \mathbb{F}[\mathbf{x}]$ is the set $\{f(A\mathbf{x}) : A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})\}$. In other words, PE is the problem of checking if f and g are the same function up to a change of the coordinate system. It can be regarded as the algebraic analogue of the graph isomorphism (GI) problem.

Much is unknown about the exact complexity of PE. Over finite fields, PE is unlikely to be NP-complete [37, 35], but no polynomial-time algorithm is known unless f and g are quadratic forms [29, 4]. PE for cubic forms over $\mathbb{Q}$ is not even known to be decidable. Cubic form equivalence (CFE) is polynomial-time equivalent to several other fundamental problems in algebra and linear algebra [18, 19, 3]. GI reduces to CFE in polynomial time [2], but the converse is not known to be true. A natural question emerges at this point:

Is GI polynomial-time equivalent to PE for some <u>natural</u> class of polynomials?

We provide an affirmative answer to this question in Theorem 11 by studying the problem of testing equivalence to design polynomials which we also refer to as the equivalence testing problem for the family of design polynomials (see Definitions 1 and 2).

Equivalence testing (ET) is closely related to the PE problem. ET for a polynomial family or a circuit class $\mathcal{F}$ is a problem wherein we are given a polynomial f, and we wish to check if f is in the orbit of some polynomial or circuit in $\mathcal{F}$.[1] Efficient ET algorithms are known for a variety of polynomial families and a few circuit classes, namely the permanent [24], the determinant [24, 15], the iterated matrix multiplication polynomial family [26, 33], the elementary and power symmetric polynomials [23], the sum-product polynomial family [31], the continuant [31], and read-once formulas [21]. One important family that is missing from the above list is the Nisan-Wigderson design polynomial family NW (see Equation 1). The NW family has been used in many results on arithmetic circuit lower bounds in the last decade. But, unlike the other families, NW had no known ET. In fact, ET for NW over $\mathbb{Q}$ was not known to be decidable. We ask a more general question:

Is there an ET algorithm for the family of (general) design polynomials?

Our main result, given in Theorem 3, is an ET algorithm for design polynomials over *any* field of zero or sufficiently large characteristic. The algorithm reveals a structural property of invertible transformations between design polynomials that enables us to prove Theorem 11. The running time of the algorithm also helps us point out a vulnerability of Patarin's authentication scheme if a random sparse polynomial is chosen in the key generation phase.

Patarin [34] proposed a zero-knowledge authentication scheme based on the presumed hardness of PE for random cubic forms (more generally, constant-degree forms). A random n-variate cubic form $f(\mathbf{x})$ is chosen in the key generation phase along with two random transforms $A_1, A_2 \in \mathrm{GL}(n, \mathbb{F})$. The polynomials $g_1 := f(A_1\mathbf{x})$ and $g_2 := f(A_2\mathbf{x})$ are made public; the secret is the transform $A_1^{-1}A_2$, which maps g_1 to g_2. A random cubic form has sparsity (i.e., number of monomials) $O(n^3)$. It is natural to ask: What if we choose a *random* $O(n^3)$-sparse polynomial f of a *higher degree* (see Definition 12) in the key generation step?

Can Patarin's scheme be broken if a random $n^{O(1)}$-sparse polynomial is chosen as the key?

[1] Note that the ET problem for $\mathcal{F}$ is not the same as the PE problem for $\mathcal{F}$. In the latter case, we are given two polynomials $f, g \in \mathcal{F}$, and we wish to check if one is in the orbit of the other. One may alternatively call the ET problem for $\mathcal{F}$ as "testing equivalence to $\mathcal{F}$" (as in the title of this article).

It turns out that a random sparse polynomial is a design polynomial and has no nontrivial permutation symmetry with high probability (see Lemma 36 and Proposition 37). These features let us invoke Theorem 3 and answer the above question in Theorem 13 via a reduction to GI.

Our final result is a learning algorithm for random affine projections of design polynomials. An equivalence test for NW is a special case of learning affine projections of NW. Consider the (qd, d, t) design polynomial $\mathsf{NW}_{q,d,t}$ as defined in Equation 1. A polynomial $f = \mathsf{NW}_{q,d,t}(A\mathbf{x} + \mathbf{b})$, where $|\mathbf{x}| = n \leq qd$, $A \in \mathbb{F}^{qd \times n}$ and $\mathbf{b} \in \mathbb{F}^{qd}$, is an n-variate affine projection of $\mathsf{NW}_{q,d,t}$. Given access to f, can we learn the unknown A and $\mathbf{b}$? If $n = qd$ and $A \in \mathrm{GL}(n, \mathbb{F})$, then the problem is the same as ET for NW. However, for arbitrary $n < qd$ and $A \in \mathbb{F}^{qd \times n}$, the problem can be rather difficult, even for $t = 1$, as every depth-3 circuit is an affine projection of $\mathsf{NW}_{q,d,1}$ for some q and d. Learning depth-3 circuits in the worst-case is a challenging problem due to known depth reduction results (see the discussion in [27]). But does the task of discovering A become easier if A is randomly chosen? In other words:

Can we learn random affine projections of NW *efficiently?*

Random affine projections of special design polynomials, such as the power symmetric polynomial and the sum-product polynomial, have been studied, and efficient learning algorithms have been provided in [27]. But it is unclear what to expect for NW. The reason is that, unlike the determinant and the permanent, we do not have a good understanding of the expressive power of affine projections of NW. The permanent is VNP-complete under p-projections[2] [38], but NW is not known to be so. For $d = n^{O(1)}$, no learning algorithm is known for n-variate random affine projections of the $d \times d$ permanent which has time complexity polynomial in $\binom{n+d}{n}$ – the maximum sparsity of any n-variate, degree-d polynomial. In fact, it is conjectured in [1] that n-variate random affine projections of the $d \times d$ *determinant* form a pseudorandom function family when $d = n^{O(1)}$. If true, then there is no $\binom{n+d}{n}^{O(1)}$-time learning algorithm for random affine projections of the determinant. If such a conclusion holds for the determinant, which is VBP-complete, then we expect the same to hold for the permanent or any other VNP-complete family, as VBP $\subseteq$ VNP. So, an efficient learning algorithm for random affine projections of NW may indicate NW is *not* VNP-complete.

In Theorem 18, we give an efficient learning algorithm for n-variate random affine projections of multilinear degree-d design polynomials, provided $n \geq d^4$. If we obtain an analogous result under the weaker assumption "$n \geq d^\epsilon$, for any $\epsilon > 0$", then the NW family is not VNP-complete assuming that there is no VNP-complete family whose random affine projections are efficiently learnable. On the other hand, if NW happens to be VNP-complete, then it is unlikely that we will be able to weaken the $n \geq d^4$ condition significantly without compromising the other parameters of the theorem considerably.

1.1 Our results

We now state our results formally. Assume that an efficient univariate polynomial factoring algorithm over $\mathbb{F}$ is available; this assumption is well justified over $\mathbb{Q}$ and $\mathbb{F}_q$ [30, 8].

▶ **Definition 1** (Design polynomial)**.** *An n-variate, degree-d polynomial $g = \sum_{i \in [s]} c_i m_i$, where m_i is a monomial and $c_i \in \mathbb{F}$, is a (n, d, s, t) design polynomial if $\forall i \neq j$, deg gcd$(m_i, m_j) < t$.*

[2] If every row of A has at most one nonzero entry, then it is a p-projection.

Some well-known polynomials that are also design polynomials are the sum-product polynomial $\mathsf{SP}_{s,d} := \sum_{i=1}^{s}\prod_{j=1}^{d} x_{i,j}$, which is a $(sd, d, s, 1)$ design polynomial, and the power symmetric polynomial $\mathsf{PSym}_{n,d} := \sum_{i=1}^{n} x_i^d$, which is a $(n, d, n, 1)$ design polynomial. The most relevant example is the Nisan-Wigderson design polynomial $\mathsf{NW}_{q,d,t}$, defined as:

$$\mathsf{NW}_{q,d,t} := \sum_{h\in\mathbb{F}_q[y],\ \deg h<t}\ \prod_{i=0}^{d-1} x_{i,h(i)}\ , \qquad \text{for } t \le d \le q, \text{ where } q \text{ is a prime.} \tag{1}$$

As two univariate polynomials of degree $< t$ agree at $\le t-1$ points, $\mathsf{NW}_{q,d,t}$ is a (qd, d, q^t, t) design polynomial. Whenever the parameter s is not required, we will write (n, d, t) design polynomial by omitting s. Also, for simplicity, we assume that design polynomials are homogeneous. Our results hold for non-homogeneous design polynomials as well.

▶ **Definition 2** (The ET problem for design polynomials). *Given black-box access to an n-variate, degree-d polynomial $f \in \mathbb{F}[\mathbf{x}]$, check if there exist an $A \in \mathrm{GL}(n,\mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^n$ such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial, and if so, recover A and $\mathbf{b}$.*

▶ **Theorem 3** (ET for design polynomials). *Let $n, d, s, t \in \mathbb{N}$, $d \ge 3t$, $\mathrm{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| > max(s^3, d^7)$. There is a randomized, $\mathrm{poly}((nd)^t)$-time[3] algorithm that takes input black-box access to an n-variate, degree-d polynomial $f \in \mathbb{F}[\mathbf{x}]$, with the promise that there exist some (n, d, s, t) design polynomial g and some $A \in \mathrm{GL}(n,\mathbb{F})$ such that $f = g(A\mathbf{x})$, and outputs, with high probability, a $B \in \mathrm{GL}(n,\mathbb{F})$ and a (n, d, s, t) design polynomial h such that $B = PSA$ and $f = h(B\mathbf{x})$, where P, S are permutation and scaling matrices, respectively.*

▶ Remark 4. If g is multilinear, then the condition $d \ge 3t$ can be improved to $d > 2t$.

▶ Remark 5. The condition $|\mathbb{F}| > \max(s^3, d^7)$ arises due to the use of the Schwartz–Zippel lemma[4] in our analysis and in the factorization algorithm of [22]. If f is given as a circuit, the algorithm can work with an extension field, irrespective of the size of $\mathbb{F}$, and still obtain a B with entries in $\mathbb{F}$. A finite field extension can be constructed efficiently (see Section 14.9 in [39]). This feature of the algorithm is explained in [7].

▶ Remark 6. The theorem gives an ET algorithm for NW (see Theorem 50). The algorithm also works over $\mathbb{Q}$, where ET for NW was not known to be decidable.

▶ Remark 7. A random sparse polynomial is a $(n, d, s, d/3)$ design polynomial with high probability (see Lemma 36). Thus, we have ET for random sparse polynomials.

In Section 3, we prove Theorem 3 and elaborate on how to handle non-homogeneous design polynomials and transforms of the form $A\mathbf{x} + \mathbf{b}$, where $A \in \mathrm{GL}(n,\mathbb{F})$ and $\mathbf{b} \in \mathbb{F}^n$.

▶ **Definition 8** (Symmetries of a polynomial). *Let $f \in \mathbb{F}[\mathbf{x}]$ be an n-variate, degree-d polynomial. The set $\mathcal{G}_f := \{A \in \mathrm{GL}(n,\mathbb{F}) : f(A\mathbf{x}) = f\}$ is the group of symmetries of f.*

The authors of [20] studied the symmetries of NW by examining the Lie algebra associated with it, while these corollaries of Theorem 3 (which is proved using a different technique) hold for general design polynomials.

▶ **Corollary 9** (Symmetries of design polynomials). *Let $d \ge 3t$, f be a (n, d, t) design polynomial and $A \in \mathcal{G}_f$. Then, $A = PS$, where P is a permutation matrix and S is a scaling matrix.*

[3] Here, "time" means number of field operations. Over $\mathbb{Q}$, the complexity is $\mathrm{poly}((nd)^t, \beta)$, where β is the bit complexity of the coefficients of f.
[4] also known as the DeMillo–Lipton–Schwartz–Zippel lemma [14, 41, 36].

▶ **Corollary 10** (Equivalent design polynomials). *Let $d \geq 3t$, $A \in \mathrm{GL}(n, \mathbb{F})$ and f, g be (n, d, t) design polynomials such that $f = g(A\mathbf{x})$. Then, $A = PS$, where P, S are as stated above.*

Our second result, proven in Section 3.5, shows that GI $\equiv_p$ PE for design polynomials with all-one coefficients. Here, $\equiv_p$ denotes polynomial-time, many-one equivalence.

▶ **Theorem 11** (GI and PE). *GI $\equiv_p$ PE for $(n, 6, 2)$ design polynomials with all-one coefficients.*

Theorem 11 holds for (n, d, t) design polynomials for any $d \geq 6$ and $t \leq d/3$, and also for (n, d, t) multilinear design polynomials with $d \geq 5$ and $t < d/2$. Thus, PE for (n, d, t) design polynomials with all-one coefficients, for $d \geq 6$ and $t \leq d/3$, polynomial-time reduces to PE for $(n, 6, 2)$ design polynomials with all-one coefficients.[5]

Our third result, proven in Section 3.5, shows that if a random sparse polynomial of sufficiently large constant degree is used in Patarin's scheme for public and private key generations, then the private key can be recovered in quasi-polynomial time.

▶ **Definition 12** (Random sparse polynomial). *An n-variate, degree-d, s-sparse polynomial $f(\mathbf{x})$ is a random s-sparse polynomial if each monomial is formed by picking d variables uniformly and independently at random from $\mathbf{x}$; the coefficients are then chosen arbitrarily.*

▶ **Theorem 13** (A vulnerability of Patarin's scheme). *Let $n, s, d, q \in \mathbb{N}$, $n > d^8$, $n^3 \leq s < \left(\frac{n}{d^2}\right)^{d/6}$, $d \geq 25$ be a constant, and $q = n^{O(1)}$. Let f be a random s-sparse polynomial over $\mathbb{F}_q$. If f is used in Patarin's scheme for key generation, then the scheme can be broken in* quasi-poly(n) *time.*

▶ Remark 14. The bound on s, stated in the theorem, is for simplicity. The precise bound is $n^2 \log(n) \leq s \leq \sqrt{\epsilon}\left(\frac{n}{d^2}\right)^{d/6}$, where ϵ is the constant from Lemma 36. The lower bound on d can be derived from the inequality $n^2 \log(n) \leq \sqrt{\epsilon}\left(\frac{n}{d^2}\right)^{d/6}$ and fixing $\epsilon = 0.01$.

▶ Remark 15. Patarin's scheme was shown to be vulnerable in [23] when using a random constant-degree *multilinear* polynomial for key generation. In [23] a random polynomial was defined by selecting the coefficients of *all* multilinear monomials randomly and independently, while we allow arbitrary coefficients, non-multilinear monomials, and a lower number of monomials.

Our fourth and final result, proven in Section 4, gives an algorithm to learn random affine projections of multilinear design polynomials such as the polynomials in the NW family.

▶ **Definition 16** (Affine projections). *Let $m, n \in \mathbb{N}$, $m \geq n$. Let f and g be polynomials in n and m variables, respectively. If $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ for some $A \in \mathbb{F}^{m \times n}$ and $\mathbf{b} \in \mathbb{F}^m$, then f is an affine projection of g. An affine projection is random if $A \in_r \mathbb{F}^{m \times n}$, where $\in_r$ denotes that the entries of A are chosen randomly and independently from a sufficiently large subset of $\mathbb{F}$.*

▶ **Definition 17** (Learning affine projections of design polynomials). *Given black-box access to an n-variate $f \in \mathbb{F}[\mathbf{x}]$, which is an affine projection of an unknown (m, d, s, t) design polynomial g, recover $B \in \mathbb{F}^{m \times n}, \mathbf{c} \in \mathbb{F}^m$ and a (m, d, s, t) design polynomial h such that $f = h(B\mathbf{x} + \mathbf{c})$.*

[5] It is worth noting, in [2], the authors showed a reduction from PE for degree-d forms to PE for cubic forms over fields containing d-th roots. However, the reduction there does not seem to preserve the design condition.

▶ **Theorem 18** (Learning random affine projections of multilinear design polynomials). *Let $m, n, d, s, t \in \mathbb{N}$, $m \geq n \geq d^{4+\epsilon}$, where $\epsilon > 0$, $d \geq 3t$, $s < \left(\frac{\sqrt{n}}{d^2}\right)^{\frac{d}{13}}$. Let $\mathrm{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| \geq \mathrm{poly}(sd)d^t$. There is a randomized, $\mathrm{poly}(m, s, n^t)$-time algorithm that takes input black-box access to an n-variate, degree-d polynomial $f \in \mathbb{F}[\mathbf{x}]$, with the promise that there exist some multilinear (m, d, s, t) design polynomial g and some $A \in_r \mathbb{F}^{m\times n}$ such that $f = g(A\mathbf{x})$, and outputs, with high probability, a $B \in \mathbb{F}^{m\times n}$ and a multilinear (m, d, s, t) polynomial h such that $B = PSA$ and $f = h(B\mathbf{x})$, where P, S are permutation and scaling matrices, respectively.*

▶ Remark 19. For $\mathsf{NW}_{q,d,t}$ with $t \leq d/1300$, $m = qd = n^{10}$ and $n \geq d^5$, it holds that $s = q^t < \left(\frac{\sqrt{n}}{d^2}\right)^{\frac{d}{13}}$. Thus, we can learn random affine projections of $\mathsf{NW}_{q,d,t}$ for $m = \mathrm{poly}(n)$, *assuming* $n \geq d^{4+\epsilon}$. The precise bound on $|\mathbb{F}|$ is stated in [7].

▶ Remark 20. Theorem 18 *does not* imply that either NW is not VNP-complete or there is a VNP-complete family whose random affine projections are learnable. The reason is that the algorithm assumes $n \geq d^{4+\epsilon}$, but it may be the case that a VNP polynomial family is a projection of NW in the setting $n < d^4$.

▶ Remark 21. As mentioned earlier, our motivation for designing a learning algorithm for random affine projections of multilinear design polynomials originates from NW, which is also multilinear and design. We believe that a similar theorem holds for non-multilinear design polynomials as well. Theorem 3 provides evidence towards this belief since it holds for non-multilinear design polynomials as well.

In Section 4, we elaborate on how non-homogeneous multilinear design polynomials and general transforms of the form $A\mathbf{x} + \mathbf{b}$, where $A \in_r \mathbb{F}^{m\times n}$ and $\mathbf{b} \in \mathbb{F}^m$, are handled.

1.2 Proof techniques

The core underlying technique, used to prove Theorems 3 and 18, is based on the vector space decomposition framework introduced in [16, 27]. Suppose that f can be expressed as:

$$f = T_1 + T_2 + \ldots + T_s \ , \tag{2}$$

and we wish to learn the *terms* $T_1, \ldots, T_s$ that are *simple* in some sense. For example, in our setting, each T_i is a product of linear forms (see details on next page). The authors in [16, 27] reduce the task of learning the T_i's to the vector space decomposition (VSD) problem. We define the VSD problem first and then discuss the reduction.

Vector space decomposition (VSD) for $(\mathcal{L}, U, V)$. Given bases of vector spaces U, V and a set of linear maps $\mathcal{L}$ from U to V, output a (further indecomposable) decomposition of U, V as:

$$U = U_1 \oplus \cdots \oplus U_s \ \text{ and } \ V = V_1 \oplus \cdots \oplus V_s \ , \text{ such that } \langle \mathcal{L} \circ U_i\rangle \subseteq V_i \text{ for all } i \in [s].$$

Reducing the learning problem to VSD. We choose appropriate sets of operators $\mathcal{L}_1$ and $\mathcal{L}_2$ to get spaces $U = \langle \mathcal{L}_1 \circ f\rangle$ and $V = \langle \mathcal{L}_2 \circ U\rangle$ such that the following are satisfied:

- $U = U_1 \oplus \cdots \oplus U_s$ and $V = V_1 \oplus \cdots \oplus V_s$, where $U_i = \langle \mathcal{L}_1 \circ T_i\rangle$ and $V_i = \langle \mathcal{L}_\in \circ U_i\rangle$.
- Each pair (U_i, V_i) is *indecomposable* with respect to $\mathcal{L}_2$.
- The (further indecomposable) decomposition is *unique* up to a reordering of U_i's and V_i's.
- T_i's can be recovered efficiently from the bases of the U_i's.

If such two sets of operators can be found, then learning $T_1, \ldots, T_s$ reduces to the VSD problem for $(\mathcal{L}_2, U, V)$. The (U_i, V_i)'s need to be indecomposable and unique otherwise a VSD algorithm might output some other decomposition, making the recovery of the T_i's hard.

Solving the VSD problem. The authors of [13] gave a polynomial-time algorithm for the symmetric case of the problem when $U = V$. The algorithm works over finite fields, $\mathbb{C}, \mathbb{R}$ but not over $\mathbb{Q}$ (if we wish to output a decomposition over $\mathbb{Q}$). The authors of [16] showed a reduction of VSD to the symmetric case. The authors of [9] and [16] exploited the structure of U and V (arising in their settings) to get a VSD algorithm that also works over $\mathbb{Q}$.

The VSD framework above gives rise to a meta algorithm for learning the "terms". To use the algorithm for Theorems 3 and 18, we need appropriate sets of operators $\mathcal{L}_1$ and $\mathcal{L}_2$ satisfying the above-mentioned conditions which point to four technical steps in the analysis, which we state first and then discuss how to execute them for Theorems 3 and 18.

Algorithm 1 Meta algorithm [16].

- **Input**: $f = T_1 + T_2 + \ldots + T_s$, where T_i's are unknown "simple" terms.
- **Output**: $T_1', T_2' \ldots T_s'$ such that $T_i' = T_{\pi(i)}$ for some permutation π on $[s]$.

```
1. Compute $U = \langle \mathcal{L}_1 \circ f \rangle$ and $V = \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ f \rangle$.
2. Solve VSD for $(\mathcal{L}_2, U, V)$; find the decompositions $U = U_1 \oplus \cdots \oplus U_s$, $V = V_1 \oplus \cdots \oplus V_s$.
3. Recover $T_i'$ from $U_i$.
```

1. **Direct sum structure:** This means establishing $U = U_1 \oplus \cdots \oplus U_s$ and $V = V_1 \oplus \cdots \oplus V_s$.
2. **Uniqueness of decomposition**: Once the direct sum holds, it needs to be shown that the decomposition of U and V is indecomposable with respect to $\mathcal{L}_2$ and unique up to permutations of the U_i's and V_i's. Inspired by the Krull-Schmidt theorem [28], [16] analysed the *adjoint algebra*[6] associated with $(\mathcal{L}_2, U, V)$ and pointed out a sufficient condition for uniqueness to hold. The adjoint algebra for $(\mathcal{L}, U, V)$ is defined as $\text{Adj}(\mathcal{L}, U, V) := \{(D, E) \mid D : U \to U,\ E : V \to V \text{ are linear maps, and } \forall L \in \mathcal{L},\ \ LD = EL\}$. The authors of [16] noted that if $\text{Adj}(\mathcal{L}, U, V)$ is block diagonalizable, i.e., $\forall (D, E) \in \text{Adj}(\mathcal{L}, U, V)$ if $D(U_i) \subseteq U_i$ and $E(V_i) \subseteq V_i$, then the decomposition is unique. So, we need to show that $\text{Adj}(\mathcal{L}_2, U, V)$ is block diagonalizable.
3. **Vector space decomposition:** With 1 and 2 satisfied, an algorithm is required to decompose U and V. As mentioned before, the vector space decomposition algorithm in [13] does not quite work over $\mathbb{Q}$. But fortunately, the adjoint algebra comes to the rescue again. Suppose, $\text{Adj}(\mathcal{L}_2, U, V)$ is block diagonalizable. If it is further block *equi-triangularizable* (refer Definition 23) and has an element (D, E), where D has s distinct eigenvalues, then the U_i's are the generalized eigenspaces of D which can be computed efficiently[7]. Thus, if $\text{Adj}(\mathcal{L}_2, U, V)$ is block equi-triangularizable, then computing vector space decomposition reduces to computing generalized eigenspaces.
4. **Recovery of T_i:** Finally, once $U_i = \langle \mathcal{L}_1 \circ T_i \rangle$ is obtained, we need to derive T_i from it.

[6] The authors of [16] attributed the use of adjoint algebra in their work to a suggestion by Youming Qiao.
[7] The existence of such an element implies that for a random $(D, E) \in \text{Adj}(\mathcal{L}_2, U, V)$, D has s distinct eigenvalues with high probability by the Schwartz-Zippel lemma.

Connecting our problems with the learning problem given by Equation (2). Let $g = \sum_{i\in[s]} c_i m_i$ be a design polynomial and $f = g(A\mathbf{x}) = \sum_{i\in[s]} T_i$, where $T_i = c_i m_i(A\mathbf{x})$ is a product of linear forms. Then, we can learn the unknown transformation A (up to permutation and scaling) by learning and factoring the terms $T_1, \dots T_s$. We do so by implementing the above steps for Theorems 3 and 18; we discuss this next. We compare our work with relevant previous works in Section A.1.

1.2.1 Implementing the four steps for Theorem 3

We choose $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$. For a (n, d, s, t) design polynomial g with $d \geq 3t$, we show Step 1 in Lemma 25 by leveraging the design property, Step 2 by showing that the adjoint is block diagonalizable (Lemma 28), and Step 3 by showing further that the adjoint is block equi-triangularizable (Proposition 30). Since the input f is in the orbit of g, these properties also hold for f by Lemma 27. A term T_i of f is in the orbit of a monomial of g, and so, T_i is a product of linear forms. The recovery process for T_i from U_i is the same as that in [27]. The transform and the design polynomial are then obtained from the T_i's (Proposition 24).

1.2.2 Implementing the four steps for Theorem 18

We choose $\mathcal{L}_1 = \partial^k$ and $\mathcal{L}_2 = \partial^2$, where $k > t$ is appropriately chosen in Lemma 45. First, we show that assuming the two non-degeneracy conditions stated in Section 4.1, all the steps can be implemented. Step 1 is immediate from the first non-degeneracy condition. Lemma 41 shows that $\mathrm{Adj}(\partial^2, U, V)$ is block diagonalizable (in fact, block equi-triangularizable) for non-degenerate affine projections. Hence, both Steps 2 and 3 hold. The process of recovering the terms (i.e., Step 4) is the same as that for Theorem 3.

Second, we show that random affine projections of design polynomials are non-degenerate with high probability. The second non-degeneracy condition holds with high probability given the restriction on the field size. If we show that $\dim U = s\binom{d}{k}$, then the direct sum structure holds[8]. By the Schwartz-Zippel lemma , it is sufficient to show that for every design polynomial, there exists an affine projection such that $\dim U = s\binom{d}{k}$. We do this by showing that the probability that $\dim U = s\binom{d}{k}$ is non-zero if f is chosen from a specific class of affine projections as described by the *two-phase* random process in the proof of Lemma 45 (which can be found in [7]).

2 Preliminaries

2.1 Notations and definitions

For $n \in \mathbb{N}$, $[n]$ is the set $\{1, 2 \dots n\}$. We use b.b.a to refer to black-box access. The set of $n \times n$ invertible matrices over $\mathbb{F}$ is denoted as $\mathrm{GL}(n, \mathbb{F})$. For two polynomials f and g, $f \sim g$ denotes that f is in the orbit of g. Variable sets are denoted as $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$. Permutation and scaling matrices are denoted as P and S, respectively. A monomial in $\mathbf{x}$ is denoted as $\mathbf{x}^{\boldsymbol{\alpha}} := x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$, which has total degree $|\boldsymbol{\alpha}| := \sum_{i=1}^{n} \alpha_i$. The set of degree t derivatives in $\mathbf{x}$ is denoted as ∂^t while $\partial^t f$ denotes the set of degree t derivatives of the polynomial f. The vector space spanned by a set of polynomials S over $\mathbb{F}$ is denoted as $\langle S \rangle$. Typically,

[8] as $U \subseteq U_1 + \dots + U_s$ and $\dim U_i \leq \binom{d}{k}$.

g denotes a (n,d,s,t) design polynomial $g = g_1 + g_2 + \cdots + g_s$ where g_i are monomials of degree d, while f denotes the input polynomial $f = g(A\mathbf{x}) = T_1 + T_2 + \cdots + T_s$, where $T_i = g_i(A\mathbf{x})$ for $A \in \mathrm{GL}(n,\mathbb{F})$. Define the spaces $U, U_i, V, V_i, U', U'_i, V', V'_i$ as follows:

$$U := \langle \mathcal{L}_1 \circ f \rangle, \quad U' := \langle \mathcal{L}_1 \circ g \rangle, \quad V := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ f \rangle, \quad V' := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ g \rangle,$$

$$U_i := \langle \mathcal{L}_1 \circ T_i \rangle, \quad U'_i := \langle \mathcal{L}_1 \circ g_i \rangle, \quad V_i := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ T_i \rangle, \quad V'_i := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ g_i \rangle,$$

where $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$ (as defined in Section 1.2.1). These spaces will be used in Section 3. For the algorithmic preliminaries, refer Section A.2.

▶ **Definition 22** (Adjoint Algebra). *The adjoint algebra associated with* $(\mathcal{L}_2, U, V)$ *is defined as* $Adj(\mathcal{L}_2, U, V) := \{(D, E) \mid D : U \to U,\ E : V \to V$ *are linear maps, and* $\forall L \in \mathcal{L}_2,\ LD = EL\}$.

Adjoint algebra was introduced in Section 4.3 of [40] as an associative ring to study the decompositions of bilinear maps and has been used in [11] for developing a fast isomorphism testing algorithm for a subclass of finite p-groups. A meta-framework for designing learning algorithms for arithmetic circuits was given in [16], where the learning problem was reduced to the vector space decomposition problem, and the uniqueness of vector space decomposition was proved by analyzing a certain adjoint algebra (refer [16] for details).

▶ **Definition 23** (Equi-triangular matrix). *An equi-triangular matrix is triangular with equal diagonal entries. Block equi-triangularizability and simultaneous block equi-triangularizability are defined similarly as block diagonalizability and simultaneous block diagonalizability.*

3 Equivalence testing for design polynomials

We state the ET algorithm in Section 3.1. The precise time complexity of the ET algorithm is $\mathrm{poly}(\binom{n+t}{n} d^t)$ we discuss this further in Section 3.2 which analyses it. Section 3.3 analyses the adjoint algebra of a design polynomial g. Based on the structure of the adjoint, Section 3.4 develops and analyses a VSD algorithm. In Section 3.5, Theorems 11 and 13 are proven.

Algorithm 2 Equivalence testing for design polynomials.

- **Input:** Black-box access to an $f \in \mathbb{F}[\mathbf{x}]$, where $f = g(A\mathbf{x})$ for some unknown (n,d,s,t) design polynomial g and $A \in \mathrm{GL}(n,\mathbb{F})$.
- **Output:** A matrix B and a (n,d,s,t) design polynomial h, where $B = PSA$ and $f = h(B\mathbf{x})$ for some permutation matrix P and scaling matrix S.

```
1. Compute black-box access to bases of $U = \langle \partial^t f \rangle$ and $V = \langle \partial^t U \rangle$.
2. Perform VSD for $(\partial^t, U, V)$ using Algorithm 3. Let $U = U_1 \oplus \cdots \oplus U_s$ be the decomposition
   returned by Algorithm 3.
3. Express $\frac{\partial^t f}{\partial \mathbf{x}^{\boldsymbol\alpha}} = u_{1\boldsymbol\alpha} + \cdots + u_{s\boldsymbol\alpha}$, where $u_{i\boldsymbol\alpha} \in U_i$ and obtain black-box access to $u_{i\boldsymbol\alpha}$ for all
   $i \in [s]$ and $\mathbf{x}^{\boldsymbol\alpha}$ of degree $t$. The black-box for $T_i$ is given by $\frac{(d-t)!}{d!} \sum_{|\boldsymbol\alpha|=t} \binom{t}{\alpha_1 \ldots \alpha_n} \mathbf{x}^{\boldsymbol\alpha} u_{i\boldsymbol\alpha}(\mathbf{x})$.
4. From black-box access to the $T_i$'s, recover and return a $B \in \mathrm{GL}(n,\mathbb{F})$ and a $(n,d,s,t)$
   design polynomial $h$ such that $f = h(B\mathbf{x})$ using Proposition 24.
```

3.1 The algorithm

1. **Step 1:** As discussed in Section 1.2, $\mathcal{L}_1 = \partial^t$ and $\mathcal{L}_2 = \partial^t$ are used to define U and V. Black-box access to bases of U and V is computable using Facts 46 and 47.
2. **Step 2:** The VSD algorithm of Section 3.4 gives b.b.a to bases of U_i's.
3. **Step 3:** Each $\frac{\partial^t f}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$ is expressible as a sum of $u_{i\boldsymbol{\alpha}} \in U_i$'s, where each $u_{i\boldsymbol{\alpha}}$ is $\frac{\partial^t T_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$. Using Fact 48, b.b.a to $\frac{\partial^t f}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$ and U_i's, black-boxes to $u_{i\boldsymbol{\alpha}}$ can be obtained. It can be verified using Lagrange's formula that the described black-box for T_i is the correct one.
4. **Step 4:** The proof of Proposition 24 details the recovery process for B and h.

▶ **Proposition 24.** *Assuming b.b.a to the T_i's, B and h can be recovered in* $\mathrm{poly}(n, s, d)$ *time.*

3.2 Analysis of the algorithm

Each step is randomized with a small probability of error. For the analysis, we assume that each step executes correctly. An implicit check is made at the end to see if h is a (n, d, s, t) design polynomial, B is invertible and $f = h(B\mathbf{x})$, failing which the algorithm is repeated.

The correctness of Algorithm 2 holds if it executes each of the four steps listed in Section 1.2 correctly. By Lemmas 25 and 27 (given below), U and V have the required direct sum structure. The correctness of the vector space decomposition follows from the correctness of Facts 46 and 47 and that of the vector space decomposition algorithm of Section 3.4. The uniqueness of decomposition follows from Proposition 30 and Lemmas 27 and 32. The correctness of the recovery of T_i's, B and h follows from Fact 48 and Proposition 24. Let U', U'_i, V', V'_i be as defined in Section 2.1.

▶ **Lemma 25.** *For $d \geq 3t$, $U' = U'_1 \oplus U'_2 \cdots \oplus U'_s$ and $V' = V'_1 \oplus V'_2 \cdots \oplus V'_s$.*

The following proposition gives the time complexity of the algorithm.

▶ **Proposition 26.** *Algorithm 2 has a running time of* $\mathrm{poly}(\binom{n+t}{n} d^t)$.

Typically $n > d$, in which case the running time is $\mathrm{poly}(n^t)$.

3.3 Structure of the adjoint algebra

Once we fix bases of the spaces U, U', V and V', a linear operator from one of these spaces to another can be naturally viewed as a matrix. Thus, $\mathrm{Adj}(\partial^t, U, V)$ and $\mathrm{Adj}(\partial^t, U', V')$ are sets of tuples of matrices with respect to appropriately chosen bases. We now show that $\mathrm{Adj}(\partial^t, U, V)$ is block equi-triangularizable, i.e., the set of matrices $\{D : (D, E) \in \mathrm{Adj}(\partial^t, U, V)\}$ is simultaneously block equi-triangularizable. By Lemma 27, it suffices to show this for $\mathrm{Adj}(\partial^t, U', V')$. Note, after fixing bases of U' and V', the operators $\partial^t : U' \to V'$ are also matrices. When we say $\mathrm{Adj}(\partial^t, U, V)$ is equal to $\mathrm{Adj}(\partial^t, U', V')$ in Lemma 27, we mean they are equal as sets of tuples of matrices with respect to appropriately chosen bases for U, V, U' and V'.

▶ **Lemma 27.** *Let U, V, U_i, V_i, U', V', U'_i and V'_i be vector spaces as defined in Section 2.1 with $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$. Define the invertible map $\phi : \mathbb{F}[\mathbf{x}] \to \mathbb{F}[\mathbf{x}]$ as $\phi(p) := p(A\mathbf{x})$ for $p \in \mathbb{F}[\mathbf{x}]$, where $A \in \mathrm{GL}(n, \mathbb{F})$ is as in Algorithm 2. Then,*

1. *$U' \cong U$, $V' \cong V$, $U'_i \cong U_i$ and $V'_i \cong V_i$ via the map ϕ. In other words, if $\mathcal{B}$ is a basis of U', $\phi(\mathcal{B})$ is a basis of U. This holds similarly for the other spaces.*
2. *Let $\mathcal{B}'_1$ and $\mathcal{B}'_2$ be bases for U' and V' respectively, with $\phi(\mathcal{B}'_1)$ and $\phi(\mathcal{B}'_2)$ being basis of U and V respectively. Then, Adj(∂^t, U' , V'), with respect to bases $\mathcal{B}'_1$ and $\mathcal{B}'_2$, is equal to Adj(∂^t, U, V), with respect to bases $\phi(\mathcal{B}'_1)$ and $\phi(\mathcal{B}'_2)$.*

▶ **Lemma 28.** *If $(D', E') \in Adj(\partial^t, U', V')$, then $D'(U_i') \subseteq U_i'$ and $E'(V_i') \subseteq V_i'$, $\forall i \in [s]$.*

The direct sum structure of U' and V' implies that every $\partial^t : U' \to V'$ is block diagonal, with respect to the monomial basis of U' and V', and by Lemma 28, $\text{Adj}(\partial^t, U', V')$ is block diagonal with respect to these bases. This and the adjoint condition imply that $\text{Adj}(\partial^t, U', V')$ comprises the adjoints of the g_i's. Thus, it suffices to analyse the adjoint of the g_i's. The adjoint of a monomial need not be trivial, as shown in Section add ref. We show that the adjoint algebra of a monomial is equi-triangularizable. For g_i, an arbitrary monomial of g, $\mathcal{B}_i' := \{\partial^t g_i\}$ is a basis[9] for U_i'. For $(D', E') \in \text{Adj}(\partial^t, U', V')$, let D_i' and E_i' be the restriction of D' and E' to U_i' and V_i' respectively. Represent D_i' as a $\dim(U_i') \times \dim(U_i')$ matrix with respect to $\mathcal{B}_i'$, where $D_i'[\mathbf{x}^{\boldsymbol{\alpha}}][\mathbf{x}^{\boldsymbol{\beta}}]$ is the coefficient of $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$ in $D_i'\left(\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}}\right)$. Lemma 29 shows when $D_i'[\mathbf{x}^{\boldsymbol{\alpha}}][\mathbf{x}^{\boldsymbol{\beta}}]$ is 0 and that all entries $D_i'[\mathbf{x}^{\boldsymbol{\alpha}}][\mathbf{x}^{\boldsymbol{\alpha}}]$ are equal. We use the notation $\text{mon}(c \cdot \mathbf{x}^{\boldsymbol{\alpha}}) := \mathbf{x}^{\boldsymbol{\alpha}}$, where $c \in \mathbb{F} \backslash \{0\}$; the definition is naturally extended to a set of monomials. For e.g., $\text{mon}\{2x_1^4, 5x_1x_2\} = \{x_1^4, x_1x_2\}$.

▶ **Lemma 29.** *Let $|\boldsymbol{\alpha}| = |\boldsymbol{\beta}| = t$, $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}} \neq 0$ and $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}} \neq 0$. Let D_i' be as above. Then,*

1. $D_i'[\mathbf{x}^{\boldsymbol{\alpha}}][\mathbf{x}^{\boldsymbol{\beta}}] = 0$, *if* $mon\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}})\} \not\subseteq mon\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}})\}$, *and*
2. $D_i'[\mathbf{x}^{\boldsymbol{\alpha}}][\mathbf{x}^{\boldsymbol{\alpha}}] = D_i'[\mathbf{x}^{\boldsymbol{\beta}}][\mathbf{x}^{\boldsymbol{\beta}}]$.

The following proposition shows that $\text{Adj}(\partial^t, U', V')$ is block equi-triangularizable.

▶ **Proposition 30.** *A basis $\mathcal{B}'$ for U' exists with respect to which any D', where $(D', E') \in Adj(\partial^t, U', V')$ for some E', is block equi-triangular.*

As detailed in the proof of Proposition 30, reordering each $\mathcal{B}_i'$ and concatenating them gives $\mathcal{B}'$. For each $\mathcal{B}_i'$, a directed acyclic graph G_i is constructed with vertices as $\mathcal{B}_i'$. For $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}, \frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}} \in \mathcal{B}_i'$, if $\text{mon}\{\partial^t\left(\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}\right)\} \subseteq \text{mon}\{\partial^t\left(\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}}\right)\}$ then an edge from $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$ to $\frac{\partial^t g_i}{\partial \mathbf{x}^{\boldsymbol{\beta}}}$ exists in G_i. The topological sort of G_i gives the reordering of $\mathcal{B}_i'$. By Lemma 29, D_i' is equi-triangular with respect to the reordered $\mathcal{B}_i'$ implying D' is block equi-triangular with respect to $\mathcal{B}'$.

3.4 Vector space decomposition

Algorithm 3 Vector space decomposition algorithm.

Input: B.b.a to bases of spaces U and V.
Output: B.b.a to bases of spaces $W_1, \ldots, W_s$, where $W_i = U_{\pi(i)}$ for some permutation π.

```
1. Compute a basis $D^{(1)}, \ldots, D^{(b)}$ of $\text{Adj}(\partial^t, U, V)_1 := \{D \mid (D, E) \in \text{Adj}(\partial^t, U, V)\}$.
2. Pick $c_1, \ldots, c_b \in_r S \subseteq \mathbb{F}$, where $|S| = s^3$. Let $D := c_1 D^{(1)} + \cdots + c_b D^{(b)}$.
3. Factorize the characteristic polynomial of $D$ and obtain its eigenvalues $\lambda_1, \cdots, \lambda_s$.
4. Compute $W_i := \text{Ker}((D - \lambda_i I)^{\dim U})$ for all $i \in [s]$ and output b.b.a to bases of
   $W_1, \ldots, W_s$.
```

Step 1 is executed by solving the linear system arising from $LD = EL$, for $L \in \mathcal{L}_2$, with the entries of D and E as the variables. In Step 2, a random linear combination of the $D^{(i)}$'s gives a random operator D. The eigenvalues of D can be found by factorizing the

[9] Assume that the set $\{\partial^t g_i\}$ consists of only the nonzero derivatives.

characteristic polynomial.[10] Step 4 involves the computation of the generalized eigenspaces of D. Note, Lemma 32 proves the uniqueness of decomposition and the indecomposability of U_i and V_i with respect to ∂^t.

▶ **Lemma 31.** *D has s distinct eigenvalues with probability $\geq 1 - \frac{\binom{s}{2}}{|S|}$.*

▶ **Lemma 32.** *Let $D \in Adj(\partial^t, U, V)$ such that it has s distinct eigenvalues $\lambda_1, \ldots, \lambda_s$. Then, $Ker((D - \lambda_i I)^{\dim\ U}) = U_{\pi(i)}$ for all $i \in [s]$, and for some permutation π on $[s]$.*

▶ **Proposition 33.** *Algorithm 3 has a running time of* $\mathrm{poly}(\binom{n+t}{n} d^t)$.

Handling non-homogeneous polynomials and translations. For non-homogeneous (n, d, s, t) design polynomials, if the degree of *all* monomials is $\geq 3t$, Lemmas 25, 28 and Proposition 30 hold. When $f(\mathbf{x}) = g(A\mathbf{x}+\mathbf{b})$ for $\mathbf{b} \in \mathbb{F}^n$, then since this transform is also invertible, a lemma similar to Lemma 27 holds. The analysis then proceeds in the same way. Proposition 24 can also recover translations. For multilinear (n, d, s, t) design polynomials, $\mathcal{L}_2 = \partial^1$ with the adjoint $\mathrm{Adj}(\partial^1, U, V)$ improves the bound on t to $d \geq 2t + 1$. Lemmas 25, 28 and 29 hold with some minor changes, and $\mathrm{Adj}(\partial^1, U', V')$ can be shown to be trivial.

3.5 Applications of the equivalence test

3.5.1 $GI \equiv_p PE$ for design polynomials: Proof of Theorem 11

GI $\leq_p$ PE for design polynomials. Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two n-vertex simple graphs with e edges each. Let there be an arbitrary ordering on the edges of both graphs with $\boldsymbol{I}_1$ an index function mapping E_1 to $[e]$ and $\boldsymbol{I}_2$ similarly mapping E_2 to $[e]$. Introduce variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_e$. Construct

$$M_1 := \{x_i x_j y^4_{\boldsymbol{I}_1(i,j)} : (i,j) \in E_1\} \text{ and } M_2 := \{x_l x_k y^4_{\boldsymbol{I}_2(l,k)} : (l,k) \in E_2\}.$$

Let $h_1(\mathbf{z}) := \sum_{m \in M_1} m$ and $h_2(\mathbf{z}) := \sum_{m \in M_2} m$, where $\mathbf{z} = \mathbf{x} \sqcup \mathbf{y}$. Both h_1 and h_2 are $(n + e, 6, e, 2)$ design polynomials with all-one coefficients constructible in $\mathrm{poly}(n)$ time. Proposition 34 follows from Corollary 10, the coefficients being 1, and $\mathbf{y}$ variables having degree 4.

▶ **Proposition 34.** *For h_1, h_2 as above, $G_1 \cong G_2 \iff h_1 \sim h_2$.*

PE for design polynomials $\leq_p$ GI. Let $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$ be (n, d, s, t) design polynomials with all-one coefficients, satisfying $d \geq 3t$. If h_1 and h_2 are multilinear, construct hypergraphs H_1 and H_2 with $\mathbf{x}$ as the vertices and subsets of vertices corresponding to the monomials of h_1 and h_2 as the hyperedges, respectively. Observe that $h_1 \sim h_2$ iff $H_1 \cong H_2$ as, by Corollary 10, if $h_1 \sim h_2$, then they are equivalent via a permutation matrix. It is well-known that hypergraph isomorphism reduces to GI (refer [32]). If h_1 and h_2 are non-multilinear, the argument is more elaborate: Now the monomials correspond to multisets of $\mathbf{x}$, while hyperedges need to be subsets of vertices. This can be handled by examining a standard reduction from hypergraph isomorphism to GI that uses bipartite graphs (see the opening paragraph of [5]). By introducing in-between vertices to handle parallel edges, graphs G_1 and G_2 can be constructed in $\mathrm{poly}(s)$ time. For details, refer to the proof of Proposition 35 in [7].

▶ **Proposition 35.** *For graphs G_1, G_2 as above, $G_1 \cong G_2 \iff h_1 \sim h_2$.*

[10] Here, we need an efficient univariate polynomial factorization algorithm over $\mathbb{F}$.

3.5.2 Cryptanalysis of Patarin's scheme: Proof of Theorem 13

Patarin's authentication scheme [34], described in Section A.3, is based on the presumed hardness of PE for random polynomials of constant degree. The attack on the scheme is as:

1. **Invoking Theorem 3:** Invoke Theorem 3 on f_1 and f_2 to obtain $h_1 \sim f_1$ and $h_2 \sim f_2$.
2. **Recovering P:** Use Theorem 11 to construct graphs G_1 and G_2 corresponding to h_1 and h_2 and use Babai's algorithm [6] for GI to recover a permutation matrix P.
3. **Recovering S:** Solve the system of monomial equations arising from $h_2(\mathbf{x}) = h_1(PS\mathbf{x})$, with the entries of S as variables.

The attack relies on Lemma 36, Propositions 37, 38, Theorems 3, 11 and Corollary 10.

▶ **Lemma 36.** *A random s-sparse polynomial, as per Definition 12, is a (n, d, s, t) design polynomial with probability at least $1 - \epsilon$, if $n > d^2$ and $s \leq \sqrt{\epsilon}\left(\frac{n}{d^2}\right)^{\frac{t}{2}}$ where $0 < \epsilon < 1$.*

▶ **Proposition 37.** *If f is a random s-sparse polynomial as per Lemma 36 with $s \geq n^3$, $n > d^8$, $d \geq 25$, $t = \frac{d}{3}$ and $\epsilon = 0.01$, f has no non-trivial permutation symmetry with high probability.*[11]

▶ **Proposition 38.** *Let f be as in Proposition 37. For $h_1(\mathbf{x}) = f(P_1S_1\mathbf{x})$ and $h_2(\mathbf{x}) = f(P_2S_2\mathbf{x})$, where P_1, P_2 are permutation matrices and S_1, S_2 are scaling matrices, there exists a unique permutation matrix P such that $h_2(\mathbf{x}) = h_1(PS\mathbf{x})$ for some scaling matrix S.*

Invoking Theorem 3. For $n > d^8$, $\epsilon = 0.01$, $t = d/3$, $d \geq 25$ and $n^3 \leq s \leq 0.1\left(\frac{n}{d^2}\right)^{\frac{d}{6}}$, Lemma 36 implies f is, with high probability, a $(n, d, s, d/3)$ design polynomial. Thus, invoking Theorem 3 on f_1 and f_2 gives h_1, h_2, $P_1S_1A_1$ and $P_2S_2A_2$ where $f_1 = h_1(P_1S_1A_1\mathbf{x})$ and $f_2 = h_2(P_2S_2A_2\mathbf{x})$. Clearly, $h_1 \sim h_2$ by the transform $P_1S_1(P_2S_2)^{-1} = PS$ for appropriate P and S. If P and S can be recovered, then $A_1^{-1}A_2 = (P_1S_1A_1)^{-1}PS(P_2S_2A_2)$ can be recovered. As $n > d^8$ and d is a constant, this step requires poly(n) time.

Recovering P. Note that P maps the monomials of h_1 to h_2 while S scales the coefficients accordingly. To recover P, treat h_1 and h_2 as design polynomials with all-one coefficients and use Theorem 11 and the GI algorithm of [6]. This step can be done in quasi-poly(s) time, and the uniqueness of P, which holds by Propositions 38 and 37, implies the correctness.

Recovering S. Let $h_1(\mathbf{x}) = \sum_{i=1}^{s} c_i m_i$ and $h_2(\mathbf{x}) = \sum_{i=1}^{s} \tilde{c}_i m_i$. Now $h_2(\mathbf{x}) = h_1(PS\mathbf{x}) = h_1(S'P\mathbf{x})$ for an appropriate scaling matrix S'. Treat the diagonal entries of S' as variables $\{z_1, z_2 \ldots z_n\}$. Equating the coefficients of the monomials, we get $c_i m_i(z_1, \cdots, z_n) = \tilde{c}_j$ where $m_j = m_i(P\mathbf{x})$. If $m_i = x_1^{\alpha_{i,1}} x_2^{\alpha_{i,2}} \ldots x_n^{\alpha_{i,n}}$, we get the following monomial equations:

$$z_1^{\alpha_{i,1}} z_2^{\alpha_{i,2}} \ldots z_n^{\alpha_{i,n}} = \tilde{c}_j c_i^{-1} \quad \forall\ i \in [s]. \tag{3}$$

There are s such equations in n variables, which is converted to a system of linear equations by taking $\log(z_j)$ as variables and $\alpha_{i,j}$ and $\log(\tilde{c}_j c_i^{-1})$ as constants. Computing $\log(a)$ over $\mathbb{F}_q$ is finding the discrete logarithm of a with respect to a generator γ of $\mathbb{F}_q^{\times}$.[12] Since $q = O(\text{poly}(n))$, γ can be found and discrete log can be computed in $O(\text{poly}(n))$ time. We get a system of s linear equations over $\mathbb{Z}_{q-1}$ which can be solved in poly(s, q) time using the Chinese Remainder Theorem, refer Chapter 5 of [39] for details.

[11] meaning, for any permutation matrix P, $f(P\mathbf{x}) = f(\mathbf{x})$ implies P is the identity matrix.
[12] that is finding a $b \in [0, q-2]$ such that $\gamma^b = a$.

4 Learning random affine projections of design polynomials

In this section, Theorem 18 is proven. Section 4.1 lists the non-degeneracy conditions imposed on affine projections of design polynomials. In Sections 4.2 and 4.4, we state and analyse the learning algorithm and the vector space decomposition algorithm, respectively. The adjoint of non-degenerate affine projections is analysed in Section 4.3. In Section 4.5, we show random affine projections of multilinear design polynomials are non-degenerate with high probability.

4.1 Non-degeneracy conditions

Let $g(\mathbf{y}) = g_1 + \cdots + g_s$ be a (m, d, s, t) multilinear design polynomial with g_i's as monomials. Let $f(\mathbf{x}) = g(l_1, l_2 \ldots l_m) = T_1 + \cdots + T_s$ be an n-variate affine projection of g with $T_i = g_i(l_1, \cdots, l_m)$, a product of d linear forms and L_i be the set of linear forms in T_i. We say f is a random affine projection if the coefficients of the l_i's are randomly chosen from $\mathbb{F}$. Define:

$$U := \langle \partial^k f \rangle, \quad V := \langle \partial^{k+2} f \rangle, \quad U_i := \langle \partial^k T_i \rangle, \quad V_i := \langle \partial^{k+2} T_i \rangle,$$

where k is as in Lemma 45 and $d \geq 2k + 2$. We say f is non-degenerate if the following holds:
1. $U = U_1 \oplus \cdots \oplus U_s$ and $V = V_1 \oplus \cdots \oplus V_s$.
2. The set L_i is $\mathbb{F}$-linearly independent for all $i \in [s]$.

4.2 The algorithm and its analysis

Algorithm 4 is similar to Algorithm 2, except $\mathcal{L}_1 = \partial^k$ and $\mathcal{L}_2 = \partial^2$ define U and V respectively and Algorithm 5 for vector space decomposition computes eigenspaces. Each step of the algorithm is randomized with a small error probability. An implicit check is run at the end of Step 4 to see if h is a (m, d, s, t) multilinear design polynomial and the linear forms per T_i are linearly independent, failing which the algorithm is repeated.

Analysis and Time complexity. We assume that each step executes without error and f is non-degenerate, which holds for a random affine projection with high probability by Lemma 45. The correctness of Algorithm 4 holds if it executes each of the four steps listed in Section 1.2 correctly. Non-degeneracy condition 1 implies the direct sum structure of U and V. The correctness of VSD follows from that of Facts 46, 47 and Algorithm 5. The uniqueness of decomposition follows from Lemmas 41 and 43. The correctness of the recovery of T_i's, B and h follows from Fact 48 and Proposition 39. Proposition 40 gives the time complexity.

▶ **Proposition 39.** *Assuming b.b.a to T_i's, B and h are recoverable in* $\mathrm{poly}(m, n, s, d)$ *time.*

▶ **Proposition 40.** *Algorithm 4 has a running time of* $\mathrm{poly}(m, s, (nd)^t)$.

4.3 Structure of the adjoint algebra

Lemma 41 states that $\mathrm{Adj}(\partial^2, U, V)$ is trivial[13] for a non-degenerate f. The idea is to leverage that each T_i is an affine projection of a multilinear monomial. By condition 2, there exists an $A_i \in \mathrm{GL}(n, \mathbb{F})$, such that in $f(A_i\mathbf{x})$, $T_i(A_i\mathbf{x})$ is a multilinear monomial. By Lemma 27, the

[13] This is a special case of being block equi-triangularizable.

Algorithm 4 Learning random affine projections of multilinear design polynomials.

- **Input:** B.b.a to $f = g(A\mathbf{x})$, where $A \in \mathbb{F}^{m\times n}$ and g is some (m, d, s, t) polynomial.
- **Output:** A matrix B and (m, d, s, t) design polynomial h, with $B = PSA$ and $f = h(B\mathbf{x})$.

```
1. Compute black-box access to some bases of $U = \langle \partial^k f \rangle$ and $V = \langle \partial^2 U \rangle$.
2. Perform VSD for $(\partial^2, U, V)$: let $U = U_1 \oplus \cdots \oplus U_s$.
3. Express $\frac{\partial^k f}{\partial \mathbf{x}^{\boldsymbol\alpha}} = u_{1\boldsymbol\alpha} + \cdots + u_{s\boldsymbol\alpha}$, where $u_{i\boldsymbol\alpha} \in U_i$ and obtain black-box access to $u_{i\boldsymbol\alpha}$ for
   all $i \in [s]$ and $\mathbf{x}^{\boldsymbol\alpha}$ of degree $k$. The black box for $T_i$ is $\frac{(d-k)!}{d!} \sum_{|\boldsymbol\alpha|=k} \binom{k}{\alpha_1 \ldots \alpha_n} \mathbf{x}^{\boldsymbol\alpha} u_{i\boldsymbol\alpha}(\mathbf{x})$.
4. From black box access to the $T_i$'s, recover and return $B \in \mathbb{F}^{m\times n}$ and $(m, d, s, t)$ design
   polynomial $h$ using Proposition 39.
```

adjoint of $f(A_i\mathbf{x})$ and that of $f(\mathbf{x})$ are equal as sets of matrices with respect to appropriate bases of their respective derivative spaces.[14] The operators in the adjoint of $f(A_i\mathbf{x})$ are shown to be invariant on the derivative spaces of $T_i(A_i\mathbf{x})$ by using the fact that the derivative space of $T_i(A_i\mathbf{x})$ is the space of multilinear polynomials in d variables. The block diagonality then holds. Because of the direct sum structure, the block diagonality of the adjoint and the block diagonality of ∂^2 operators, it suffices to analyse the adjoint of individual T_i's. Since $T_i(A_i\mathbf{x})$ is a multilinear monomial, its adjoint is trivial and thus so is the adjoint of T_i.

▶ **Lemma 41.** *Let g and f be as defined in Section 4.1. If f is non-degenerate, then Adj(∂^2, U, V) is block-diagonal and is also trivial (thus, also block equi-triangular).*

4.4 Vector space decomposition algorithm

Algorithm 5 Vector Space Decomposition Algorithm.

Input: B.b.a to bases of spaces U and V.
Output: B.b.a to bases of $W_1, \ldots, W_s$ where $W_i = U_{\pi(i)}$ for some permutation π.

```
1. Compute a basis $D^{(1)}, \ldots, D^{(b)}$ of Adj$(\partial^2, U, V)_1 = \{D \mid (D, E) \in$ Adj$(\partial^2, U, V)\}$.
2. Select $c_1, \ldots, c_b \in_r S \subseteq \mathbb{F}$, $|S| = s^3$ and let $D = c_1 D^{(1)} + \cdots + c_b D^{(b)}$.
3. Factorize the characteristic polynomial of $D$ to obtain eigenvalues $\lambda_1, \ldots, \lambda_s$.
4. Compute $W_i :=$ Ker$(D - \lambda_i I)$ for all $i \in [s]$ and output b.b.a to the bases of $W_1, \ldots, W_s$.
```

Analysis and Time complexity. Algorithm 5 is similar to Algorithm 3 except it uses ∂^2 operators and computes the eigenspaces of D, instead of generalized eigenspaces. Thus, the analysis for Algorithm 5 is the same as for Algorithm 3. Lemmas 41 and 42 prove that Algorithm 5 works with high probability. Proposition 44 gives the time complexity.

▶ **Lemma 42.** *D has s distinct eigenvalues with probability $\geq 1 - \frac{\binom{s}{2}}{|S|}$.*

▶ **Lemma 43.** *Let $D \in$ Adj(∂^k, U, V) such that it has s distinct eigenvalues denoted $\lambda_1, \ldots, \lambda_s$. Then Ker$(D - \lambda_i I) = U_{\pi(i)}$ for all $i \in [s]$, where π is some permutation on $[s]$.*

▶ **Proposition 44.** *Algorithm 5 has a running time of* poly(n, s, d^t).

[14] Lemma 27 holds more generally for any two polynomials equivalent by invertible linear transforms.

Handling non-homogeneous polynomials and translation. The non-degeneracy conditions are the same for non-homogeneous (m, d, s, t) multilinear design polynomials and when $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ for $A \in_r \mathbb{F}^{m\times n}$ and $\mathbf{b} \in \mathbb{F}^m$. For the non-homogeneous case, if the degree of *all* monomials is $\geq 2k+2$, then Lemmas 41 and 45 hold, and the analysis proceeds in the same way. Proposition 39 can also recover translations.

4.5 Random affine projections are non-degenerate

Lemma 45 states that a random affine projection f is non-degenerate with high probability. Showing f is non-degenerate reduces to showing certain matrices, with entries as the coefficients of the l_i's, are full rank with high probability when the entries are chosen randomly. The main technical challenge is in proving condition 1 because the g_i's share variables, thus T_i's share the l_i's. A two-stage random process is used to show the existence of an affine projection which satisfies condition 1. The Schwartz-Zippel lemma then implies that a random affine projection also satisfies condition 1 with high probability.

▶ **Lemma 45.** *Let g be a polynomial as defined in Section 4.1 and f be its random affine projection. For $k = t + \left\lfloor \frac{2\log(s)}{\log\left(\frac{\sqrt{n}}{d^2}\right)} \right\rfloor + 1$, f is non-degenerate with high probability.*

5 Conclusion

In this work, we design an ET algorithm for general design polynomials (Theorem 3) and a learning algorithm for random affine projections of multilinear design polynomials (Theorem 18). As an application of the ET algorithm, we show that GI is polynomial-time equivalent to PE for design polynomials with all-one coefficients (Theorem 11). As another application, we show that Patarin's authentication scheme can be broken if it uses a higher degree sparse polynomial for key generation (Theorem 13). We also give an ET algorithm for the NW design polynomial using Theorem 3 (Theorem 50). Theorem 18 is a significant generalization of the main result in [27] that gave a learning algorithm for random affine projections of the sum-product polynomial, which is a special multilinear design polynomial.

Both the algorithms are based on the vector space decomposition framework of [27, 16]. This work's main technical contributions include analysing a non-trivial adjoint algebra associated with design polynomials and developing a VSD algorithm based on generalized eigenspaces. We end by listing some related questions:

1. **ET for sparse polynomial:** What is the complexity of ET for the class of sparse polynomials? That is, given black-box access to a polynomial f and a parameter s, what is the complexity of testing whether f is in the orbit of some s-sparse polynomial? The authors of [12] showed that the shift equivalence problem for sparse polynomials (i.e., when $f = g(\mathbf{x} + \mathbf{b})$ for some sparse polynomial g and $\mathbf{b} \in \mathbb{F}^n$) is undecidable over $\mathbb{Z}$.
2. **Weakening $n \geq d^{4+\epsilon}$:** Can the condition $n \geq d^{4+\epsilon}$ in Theorem 18 be changed to $n \geq d^\delta$ for arbitrary $\delta > 0$? Doing so would give stronger evidence that NW is not VNP-complete.
3. **Efficient ET for NW:** Theorem 50 gives an ET for NW, but it is not a polynomial-time algorithm. Is there a polynomial-time ET algorithm for NW? Our ET algorithm is for general design polynomials; it is possible that analyzing the properties of the NW polynomial may yield an efficient ET algorithm specifically for NW.

References

1 Scott Aaronson. Arithmetic natural proofs theory is sought. http://www.scottaaronson.com/blog/?p=336, 2008.

2 Manindra Agrawal and Nitin Saxena. Automorphisms of finite rings and applications to complexity of problems. In *Proceedings of the 22nd Annual Conference on Theoretical Aspects of Computer Science*, STACS'05, pages 1–17, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31856-9_1.

3 Manindra Agrawal and Nitin Saxena. Equivalence of F-algebras and cubic forms. In *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science*, STACS'06, pages 115–126, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11672142_8.

4 Manuel Araújo. Classification of quadratic forms. https://www.math.tecnico.ulisboa.pt/~ggranja/manuel.pdf, 2011.

5 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Seinosuke Toda. Colored Hypergraph Isomorphism is Fixed Parameter Tractable. *Algorithmica*, 71(1):120–138, 2015. Conference version appeared in the proceedings of FSTTCS 2010. doi:10.1007/s00453-013-9787-y.

6 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

7 Omkar Baraskar, Agrim Dewan, and Chandan Saha. Testing equivalence to design polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 2024. URL: https://eccc.weizmann.ac.il/report/2024/004/.

8 Elwyn R. Berlekamp. Factoring polynomials over large finite fields. In Stanley R. Petrick, Jean E. Sammet, Robert G. Tobey, and Joel Moses, editors, *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, SYMSAC 1971, Los Angeles, California, USA, March 23-25, 1971*, page 223. ACM, 1971. doi:10.1145/800204.806290.

9 Vishwas Bhargava, Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning generalized depth three arithmetic circuits in the non-degenerate case. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.APPROX/RANDOM.2022.21.

10 Markus Bläser, B. V. Raghavendra Rao, and Jayalal Sarma. Testing polynomial equivalence by scaling matrices. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2017. doi:10.1007/978-3-662-55751-8_10.

11 Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. A fast isomorphism test for groups whose lie algebra has genus 2. *Journal of Algebra, Volume 473, Pages 545-590, ISSN 0021-8693*, 2017. doi:10.1016/j.jalgebra.2016.12.007.

12 Suryajith Chillara, Coral Grichener, and Amir Shpilka. On hardness of testing equivalence to sparse polynomials under shifts. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 22:1–22:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.22.

13 Alexander Chistov, Gábor Ivanyos, and Marek Karpinski. Polynomial time algorithms for modules over finite dimensional algebras. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 68–74, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258726.258751.

14 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

15 Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant equivalence test over finite fields and over Q. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.62.

16 Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 889–899. IEEE, 2020. doi:10.1109/FOCS46700.2020.00087.

17 Joshua A. Grochow. *Symmetry and equivalence relations in classical and geometric complexity theory*. PhD thesis, University of Chicago, Chicago, IL, 2012. URL: https://home.cs.colorado.edu/~jgrochow/grochow-thesis.pdf.

18 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: tensor isomorphism-completeness. *SIAM J. Comput.*, 52(2):568–617, 2023. Conference version appeared in the proceedings of ITCS 2021. doi:10.1137/21M1441110.

19 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials IV: linear-length reductions and their applications. *CoRR*, abs/2306.16317, 2023. doi:10.48550/arXiv.2306.16317.

20 Nikhil Gupta and Chandan Saha. On the symmetries of and equivalence test for design polynomials. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.MFCS.2019.53.

21 Nikhil Gupta, Chandan Saha, and Bhargav Thankey. Equivalence test for read-once arithmetic formulas. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4205–4272. SIAM, 2023. doi:10.1137/1.9781611977554.ch162.

22 Erich L. Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990. Conference version appeared in the proceedings of FOCS 1988. doi:10.1016/S0747-7171(08)80015-6.

23 Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1409–1421. SIAM, 2011. doi:10.1137/1.9781611973082.108.

24 Neeraj Kayal. Affine projections of polynomials: extended abstract. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 643–662. ACM, 2012. doi:10.1145/2213977.2214036.

25 Neeraj Kayal, Vineet Nair, and Chandan Saha. Average-case linear matrix factorization and reconstruction of low width algebraic branching programs. *Comput. Complex.*, 28(4):749–828, 2019. doi:10.1007/S00037-019-00189-0.

26 Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Trans. Comput. Theory*, 11(1):2:1–2:56, 2019. Conference version appeared in the proceedings of CCC 2017. doi:10.1145/3282427.

27 Neeraj Kayal and Chandan Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 413–424. ACM, 2019. doi:10.1145/3313276.3316360.

28 Krull-Schmidt Theorem. https://mathstrek.blog/2015/01/17/krull-schmidt-theorem/.

29 T. Y. Lam. *Introduction To Quadratic Forms Over Fields*. American Mathematical Society, 2004.

30 Arjen K Lenstra, Hendrik W Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

31 Dori Medini and Amir Shpilka. Hitting sets and reconstruction for dense orbits in vp_{e} and $\Sigma\Pi\Sigma$ circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 19:1–19:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.19.

32 Gary L. Miller. Graph isomorphism, general remarks. *J. Comput. Syst. Sci.*, 18(2):128–142, 1979. doi:10.1016/0022-0000(79)90043-6.

33 Janaky Murthy, Vineet Nair, and Chandan Saha. Randomized polynomial-time equivalence between determinant and trace-imm equivalence tests. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 72:1–72:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.MFCS.2020.72.

34 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.

35 Nitin Saxena. *Morphisms of rings and applications to complexity*. PhD thesis, Indian Institute of Technology, Kanpur, 2006. URL: https://www.cse.iitk.ac.in/users/manindra/Students/thesis_saxena.pdf.

36 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

37 Thomas Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chic. J. Theor. Comput. Sci.*, 1998, 1998. URL: http://cjtcs.cs.uchicago.edu/articles/1998/1/contents.html.

38 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

39 Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra (2. ed.)*. Cambridge University Press, 2003.

40 James B. Wilson. Decomposing p-groups via Jordan algebras. *Journal of Algebra, Volume 322, Issue 8, Pages 2642-2679, ISSN 0021-8693,*, 2009. doi:10.1016/j.jalgebra.2009.07.029.

41 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

A Appendix

A.1 Comparison with previous work

As mentioned earlier, ET has been studied for various polynomial families. ET algorithms for power symmetric polynomials [23] and read-once formulas [21] were given by analyzing the factors of the Hessian determinant. Analyzing the Lie algebra of the determinant [24, 15], the permanent [24], and the iterated matrix multiplication [26, 33] polynomials led to ET algorithms for these families. For the elementary symmetric polynomials, the maximal dimension of the space of second-order partials gave an ET algorithm [23]. It was shown in [17] that over algebraically closed fields of characteristic 0, ET for polynomials characterized by the continuous part of their symmetries is equivalent to testing matrix isomorphism to their Lie algebras, implying over such fields an efficient ET exists for such polynomials, provided matrix isomorphism to their Lie algebra can be solved efficiently. However, for design polynomials, these techniques *do not* work. For example, the $(2, d, 2, 3)$ design polynomial $x_1x_2^{d-1} + x_1^{d-1}x_2$, where $d \geq 3$ is odd, has a trivial Lie algebra and an irreducible Hessian determinant over $\mathbb{Q}$. For $d \geq 3$, the $(2d, d, 2, 1)$ design polynomial $\prod_{i=1}^{d} x_i + \prod_{i=d+1}^{2d} x_i$ has second-order partials dimension $2\binom{d}{2}$, which is less than the maximum possible dimension $\binom{2d}{2}$. Design polynomials, particularly NW, are *not* characterized by the continuous part of their symmetries (see [20]); hence the ET algorithm implied from [17] does not apply. Thus, a different technique must be used for design polynomials.

The VSD framework was used previously in [27, 16, 9] to design learning algorithms. The authors of [31] showed that polynomials in the orbit of the sum-product polynomial satisfy the non-degeneracy conditions of [27], and so, we have an ET algorithm for this family. But note that a sum-product polynomial has a read-once formula; an ET algorithm can also be obtained via the Hessian determinant [21]. To our knowledge, our work is the first to use the VSD framework to develop an ET algorithm (in Theorem 3) for a natural family of polynomials for which none of the other three techniques work. Also, there is a notable difference between the analysis in [16, 9] and the analysis here. In [16, 9], the relevant adjoint algebra is diagonalizable, while in our case the adjoint is block equi-triangularizable. The VSD algorithms of previous works recovered the component spaces by computing eigenspaces, while we compute generalized eigenspaces to do so.

We learn affine projections of design polynomials (in Theorem 18) under certain non-degeneracy conditions similar to those of [27]. However, proving the non-degeneracy of random affine projections of design polynomials requires a more involved analysis than proving the non-degeneracy of random depth-3 circuits in [27]. The reason is, unlike the terms of the circuit models studied in [27, 16, 9], the terms in our case have *shared randomness* as the same random affine form can appear in multiple terms. Theorem 18 is a significant generalization of the main result in [27] that handles random affine projections of the sum-product polynomial, which is a special design polynomial.

The learning algorithm in Theorem 18 is proper as it outputs an appropriate affine map. It is also an average-case algorithm for learning affine projections of design polynomials as the input is a random affine projection. This average-case, proper learning algorithm exploits the property that the space of partial derivatives of an affine projection of a design polynomial is *low* dimensional (under the technical conditions mentioned in the theorem statement) to reduce the learning problem to VSD. A natural question arises at this point: is it always possible to design an average-case, proper learning algorithm (via a reduction to VSD) for affine projections of a model satisfying such low dimensional partial derivatives space property? The answer is unclear. The related version [7] gives an example of affine projections of low width algebraic branching programs (ABPs) satisfying the technical assumptions of Theorem 18 and with low dimensional partial derivatives spaces, and for which (to the best of our knowledge) no average-case, proper learning algorithm is known. The authors of [25] gave such an algorithm assuming that the widths are the same across the layers of the ABP.

A.2 Algorithmic preliminaries

▶ **Fact 46.** *Given black-box access to an n-variate degree d polynomial f, we can compute black-box access to $\frac{\partial^k f}{\partial \mathbf{x}^{\boldsymbol{\alpha}}}$ in $\mathrm{poly}(n, d^k)$ time, where $|\boldsymbol{\alpha}| = k$. (Refer [26] for a proof idea.)*

▶ **Fact 47.** *Given black-boxes to n-variate degree d polynomials $f_1, f_2 \ldots, f_l$, there is a randomized* $\mathrm{poly}(n, l, d)$ *time algorithm that computes a basis for the vector space*

$$(f_1, f_2 \ldots f_l)^{\perp} := \{(\alpha_1, \alpha_2, \ldots, \alpha_l) \in \mathbb{F}^l : \sum_{i=1}^{l} \alpha_i f_i = 0\}$$

Refer [23] for a proof of the above fact. A corollary of Fact 47 is:

▶ **Fact 48.** *Given black-box access to linearly independent polynomials $f_1, \ldots, f_l$ and an $f = \sum_{i=1}^{l} \beta_i f_i$, where $\beta_i \in \mathbb{F}$, the β_i's can be computed in randomized* $\mathrm{poly}(n, l, d)$ *time.*

▶ **Fact 49.** *Let $d \in \mathbb{N}$, $\mathrm{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| \geq d^6$. Given black-box access to an n-variate degree d polynomial f, black-box access to its irreducible factors can be computed in randomized* $\mathrm{poly}(n, d)$ *time. (Refer [22] for details.)*

A.3 Description of Patarin's scheme

Patarin's scheme is a provably perfect zero-knowledge authentication scheme; thus Alice can prove to Bob that she knows a secret without revealing *any* information about the secret. The key generation process is as follows:

1. Select an n-variate degree d polynomial $f(\mathbf{x}) \in_r \mathbb{F}_q[\mathbf{x}]$, where d is a constant.
2. Select two matrices $A_1, A_2 \in_r \mathbb{F}_q^{n \times n}$.[15]
3. Compute the public key $(f_1, f_2) := (f(A_1\mathbf{x}), f(A_2\mathbf{x}))$ and the private key $C = A_2^{-1}A_1$.

The authentication procedure is as follows:

1. Alice selects an $R \in_r \mathbb{F}_q^{n \times n}$ and computes $g := f_1(R\mathbf{x})$ and sends it to Bob.
2. Bob receives g and picks $h := f_1$ or f_2 with probability $\frac{1}{2}$, challenging Alice to show $h \sim g$.
3. Alice receives h. If $h = f_1$, she sends R. If $h = f_2$, she sends CR.

A.4 Equivalence Testing for NW

The PS-equivalence testing problem for NW is as follows: given a polynomial f, check if $f = \mathsf{NW}_{q,d,t}(PS\mathbf{x})$ for some permutation P and scaling S, and recover them if so. Theorem 50 (the proof can be found in [7]) follows from Theorems 3 and 11, and the GI algorithm in [6].

▶ **Theorem 50.** *Let $q, d, t \in \mathbb{N}$, $q \geq d$, $t \leq d/3$, $|\mathbb{F}| \geq q^{3t}$, $\mathrm{char}(\mathbb{F}) = 0$ or $> d$. ET for $\mathsf{NW}_{q,d,t}$ reduces to PS-equivalence testing for $\mathsf{NW}_{q,d,t}$ in* $\mathrm{poly}(q^t)$ *time. Further, PS-equivalence testing for $\mathsf{NW}_{q,d,t}$ reduces to S-equivalence testing for $\mathsf{NW}_{q,d,t}$ in* $\text{quasi-poly}(q^t)$ *time.*

The S-equivalence testing problem for NW is as follows: Given a polynomial f, check if $f = \mathsf{NW}_{q,d,t}(S\mathbf{x})$ for some scaling S, and recover it if so. Over $\mathbb{R}$ and $\mathbb{F}_p$, S-equivalence testing for NW can be done by the algorithm of [10] in $\mathrm{poly}(q, \beta)$ time, where β is the bit complexity of the coefficients of f, and also by an algorithm of [20]. Over $\mathbb{Q}$, S-equivalence testing of NW can be done in $\mathrm{poly}(q^t, \beta)$ time, assuming oracle access to integer-factoring. This combined with Theorem 50 gives a quasi-poly(q^t) time algorithm for ET for NW. Here is a proof sketch of S-equivalence test for $\mathsf{NW}_{q,d,t}$ over $\mathbb{Q}$: If f is S-equivalent to $\mathsf{NW}_{q,d,t}$, then $f = \sum_{h \in \mathbb{F}_q[y],\ \deg h < t} \ c_h \prod_{i=0}^{d-1} x_{i,h(i)}$, where $c_h \in \mathbb{F}$. With the entries of S as $\mathbf{z}$ variables, we get the following equations:

[15] A random matrix is invertible with high probability.

$$z_{0,h(0)}z_{1,h(1)}\dots z_{d-1,h(d-1)} = c_h^{-1} \qquad \text{for } h \in \mathbb{F}_q[y]_{<t} \ .$$

There are q^t many equations in qd variables. Since $c_h^{-1} \in \mathbb{Q}$, $c_h^{-1} = a/b$ for some $a, b \in \mathbb{Z}$. Using the integer-factoring oracle, factor the integers a, b into primes $p_1, p_2 \dots p_l$. Now, reduce it to solving an appropriate Diophantine linear system by taking logarithms to the base p_i, with $\log_{p_i}(z_{j,h(j)})$ as variables $w_{j,h(j),i}$. This Diophantine linear system has lqd variables and lq^t equations. Treating this system as a matrix, check its consistency and then determine the linearly independent rows (say there are k of them). A $k \times k$ submatrix with non-zero determinant m can be formed from these rows, which corresponds to expressing the Diophantine linear system as linear equations in k of the $w_{j,h(j),i}$ variables with constants as affine forms in the remaining $w_{j,h(j),i}$ variables. By Cramer's rule, a solution to such a system is a fraction with the numerator as the affine forms and the denominator as m. The problem then further reduces to solving a linear system determined by the affine forms over the ring $\mathbb{Z}_m$ since $w_{j,h(j),i}$ must be integers. This whole process can be done in $\text{poly}(q^t, \beta)$ time.

Therefore, ET for NW can be solved in time quasi-polynomial in the sparsity of $\mathsf{NW}_{q,d,t}$.