Beating Brute Force for Systems of Polynomial Equations over Finite Fields[*]

Daniel Lokshtanov[†] Ramamohan Paturi[‡] Suguru Tamaki[§] Ryan Williams[¶]
Huacheng Yu[¶]

Abstract

We consider the problem of solving *systems of multivariate polynomial equations of degree k* over a finite field. For every integer $k \geq 2$ and finite field $\mathbb{F}_q$ where $q = p^d$ for a prime p, we give, to the best of our knowledge, the first algorithms that achieve an exponential speedup over the brute force $O(q^n)$ time algorithm in the worst case. We present two algorithms, a randomized algorithm with running time $q^{n+o(n)} \cdot q^{-n/O(k)}$ time if $q \leq 2^{4ekd}$, and $q^{n+o(n)} \cdot (\frac{\log q}{dek})^{-dn}$ otherwise, and a deterministic algorithm for *counting* solutions with running time $q^{n+o(n)} \cdot q^{-n/O(kq^{6/7d})}$. For the important special case of quadratic equations in $\mathbb{F}_2$, our randomized algorithm has running time $O(2^{0.8765n})$.

For systems over $\mathbb{F}_2$ we also consider the case where the input polynomials do not have bounded degree, but instead can be efficiently represented as a $\Sigma\Pi\Sigma$ circuit, i.e., a sum of products of sums of variables. For this case we present a deterministic algorithm running in time $2^{n-\delta n}$ for $\delta = 1/O(\log(s/n))$ for instances with s product gates in total and n variables.

Our algorithms adapt several techniques recently developed via the polynomial method from circuit complexity. The algorithm for systems of $\Sigma\Pi\Sigma$ polynomials also introduces a new *degree reduction* method that takes an instance of the problem and outputs a subexponential-sized set of instances, in such a way that feasibility is preserved and every polynomial among the output instances has degree $O(\log(s/n))$.

Keywords: exponential time algorithms, circuit satisfiability, counting, polynomial method, derandomization

[*]This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, Berkeley, CA.

[†]University of Bergen `daniello@ii.uib.no. Supported by the Beating Hardness by Pre-processing grant of the Bergen Research Foundation`

[‡]University of California, San Diego `paturi@cs.ucsd.edu`. This research is supported by NSF grant CCF-1213151 from the Division of Computing and Communication Foundations. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[§]Kyoto University `tamak@kuis.kyoto-u.ac.jp`. Supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (25240002, 26330011); the John Mung Advanced Program of Kyoto University. Part of the work performed while the author was at Department of Computer Science and Engineering, University of California, San Diego.

[¶]Stanford University. `rrw@cs.stanford.edu, yuhch123@gmail.com`. Supported by an Alfred P. Sloan Fellowship and NSF grants CCF-1212372 and CCF-1552651 (CAREER). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

1 Introduction

We consider the problem of solving *systems of multivariate polynomial equations* over a finite field of order $q = p^d$. For brevity, we call the problem SysPolyEqs(q). An instance P of the problem has the form $\{p_1, p_2, \dots, p_m\}$, where each $p_i \in \mathbb{F}_q[x_1, x_2, \dots, x_n]$ and is represented as a sum of monomials where the exponent of each variable is at most $q-1$ (due to the identity $x^q = x$). Our task is to decide whether P is *feasible*, i.e., whether there exists an $x \in \mathbb{F}_q^n$ such that $p_1(x) = p_2(x) = \cdots p_m(x) = 0$ holds. We call such an x a *satisfying assignment* to P.

We define the degree of an instance as $\max_{1 \le i \le m} \deg(p_i)$, where $\deg(p_i)$ is the maximum degree of the monomials of p_i and the degree of a monomial is defined as the sum of the exponents of the variables in it. For instances of degree one, the problem is solvable in polynomial time by Gaussian elimination. An easy reduction from the circuit satisfiability problem shows that SysPolyEqs(2) is NP-complete for degree-2 instances. In addition, Håstad [18] showed that, given a degree-k instance, it is NP-hard to find an assignment to a feasible SysPolyEqs(2) instance that satisfies a $(2^{1-k} - 2^{1-2k} + \varepsilon)$-fraction of equations for any $k \ge 2$ and $\varepsilon > 0$. Thus the problem is extremely hard to approximate.

In terms of exact solvability, there are few positive results known. Although several algorithms for SysPolyEqs(2) are known in the crypto community based on plausible average-case assumptions (see the section on related work), to the best of our knowledge no algorithm running in $q^{\delta n}$ time (for some fixed $\delta < 1$) has been reported for the general case of SysPolyEqs(q), for any prime power q and any degree larger than 1.

Our Results. We present algorithms for the problem that beat brute force search decisively for bounded degree instances in all finite fields.

Theorem 1.1 (Solving Low-Degree Systems of Polynomial Equations)**.** *Let p be a prime, and $q = p^d$ for $d \ge 1$. There is a randomized algorithm that, given an instance of SysPolyEqs(q) with m polynomial equations of degree at most k in n variables, decides the satisfiability of the system correctly with high probability. The running time of the algorithm is bounded by*

- $O^\star(2^{0.8765n})$ *time*[1] *when* $q = k = 2$,
- $O^\star(q^{(1-\frac{1}{5k})n} \cdot n^{3k})$ *when* $p = 2$ *(but* $q > 2$ *or* $k > 2$*),*
- $O^\star(q^{(1-(1/200k))n} \cdot n^{3qk})$ *when* $p > 2$ *and* $\log p < 4ek$*,*
- $O^\star\left(q^n \cdot \left(\frac{\log q}{ekd}\right)^{-dn}\right)$ *when* $p > 2$ *and* $\log p \ge 4ek$

The running time bounds of Theorem 1.1 can be interpreted in the following way. When the base of the field size is "small" relative to the degree, that is, $p < 2^{4ek}$, the algorithm outperforms brute force by a multiplicative factor of $q^{n/O(k)}$. This corresponds to not having to guess on a $1/O(k)$ fraction of the variables, and is qualitatively the same kind of "savings" as that of the fastest known algorithms for k-SAT [32]. It would be quite surprising to see an algorithm for SysPolyEqs(q) beating brute force by a factor more than $q^{n/O(k)}$. When the base p of the field size becomes "very large" compared to the degree bound, i.e., $\log p \ge 4ek$, the algorithm no longer achieves the multiplicative factor $q^{n/O(k)}$ improvement over brute force, instead the improvement is a factor $\left(\frac{\log q}{ekd}\right)^{dn}$, which is still substantial and for example much larger than $q^{O(n/kq)}$.

The algorithm of Theorem 1.1 is randomized, a natural goal is to obtain a deterministic algorithm with the same running time. We are currently unable to achieve this goal, but we do obtain a deterministic

[1]The $O^\star$ notation omits polynomial factors.

algorithm that decisively beats brute force and also solves the problem of counting the number of satisfying assignments.

Theorem 1.2 (Counting Solutions to Low-Degree Systems of Polynomial Equations $\langle\star\rangle$[2])**.** *Let $q = p^d$ for a prime p and integer $d \geq 1$. There is a deterministic algorithm that, given an instance of SysPolyEqs(q) with m polynomial equations of degree at most k in n variables, runs in time $q^{n\left(1-\frac{1}{300kq^{6/7d}}\right)+o(n)} \cdot m^{O(qk)}$ and counts the number of satisfying assignments for the system.*

We then proceed to consider the case of polynomial equations in $\mathbb{F}_q$ where the input polynomials do *not* have bounded degree, but instead can be encoded efficiently as a $\Sigma\Pi\Sigma$ circuit; i.e., a linear form of products of linear forms on the variables. Specifically, we require that an encoding of each p_i as a $\Sigma\Pi\Sigma$ circuit is provided as input. We call this variant of the problem GenSysPolyEqs(q). For $q = 2$ we obtain an algorithm that achieves exponential speedup over 2^n as long as the total number of product gates in the input polynomials is linear in the number n of variables. Again we obtain one deterministic and one randomized algorithm, with the deterministic algorithm counting the number of satisfying assignments. However, in this case the difference between the running times of the deterministic and the randomized algorithms is only in the constant factor in the savings.

Theorem 1.3 (Solving Systems of $\Sigma\Pi\Sigma$ Polynomials)**.** *There is a randomized algorithm that, given an instance of GenSysPolyEqs(2) of size u with s products and n variables, runs in time* $\mathrm{poly}(u) \cdot 2^{\left(1-\frac{1}{10\log(s/n)+O(1)}\right)n}$, *and decides the satisfiability of the system correctly with high probability.*

Theorem 1.4 (Counting Solutions to Systems of $\Sigma\Pi\Sigma$ Polynomials)**.** *There is a deterministic algorithm that, given an instance of GenSysPolyEqs(2) of size u with s products and n variables, runs in time* $\mathrm{poly}(u) \cdot 2^{\left(1-\frac{1}{1100\log(s/n)+O(1)}\right)n}$ *and counts the number of satisfying assignments for the system.*

Note that if the degree of each $\Sigma\Pi\Sigma$ polynomial is at most k, then one can in principle use the algorithms of Theorems 1.1 and 1.2 to solve GenSysPolyEqs(2). However, in general the degree of each $\Sigma\Pi\Sigma$ circuit in our instances can be larger than n, and our algorithm for GenSysPolyEqs(2) can still run in time that is super-polynomially faster than 2^n.

Related Work, Solving systems of multivariate polynomial equations is a fundamental problem in mathematics, science and engineering; see for example [11, 36]. The problem of detecting rational points in $\mathbb{F}_q$ (finding a non-zero point that makes a polynomial zero) is widely studied ([8, 19, 25, 26]).

Systems of Degree-Two. For fields of characteristic two and the polynomials of degree two, the problem arises in breaking certain cryptosystems based on the presumed hardness of quadratic polynomial equations [4, 16]. This underscores the importance of beating exhaustive search in the degree-2 case. For the purposes of this paragraph, we call the problem Deg2-SysPolyEqs(q). Woods [41] gave an interesting non-deterministic proof system for Deg2-SysPolyEqs(q), showing how one can prove that a system of quadratic equations is infeasible with an $O^\star(q^{n/2})$-length proof verifiable in $O^\star(q^{n/2})$ time. Under several algebraic assumptions, Yang and Chen [42] estimate an $O(2^{0.875n})$ time bound for Deg2-SysPolyEqs(2). Miura *et al.* [30] show how to solve Deg2-SysPolyEqs(2) in polynomial time when the system is sufficiently under-determined (in particular, the number of variables $n \geq \Omega(m^2)$, where m is the number of equations). Bardet *et al.* [5] gave algorithms for Deg2-SysPolyEqs(2) running in deterministic $2^{0.841n}$ time and Las Vegas $2^{0.792n}$ time for the case $m = n$ under certain algebraic assumptions on the instances. In general, considerable research in modern cryptanalysis is centered around solving multivariate systems of low-degree polynomial equations (see the books [17, 3]).

[2]Proofs of statements labeled with $\langle\star\rangle$ can be found in the appendix.

Efficient Algorithms For Special Cases. Lu [28] gave a deterministic algorithm for finding a solution to a *single* polynomial equation that runs in time $\text{poly}(s)$ for polynomials with s monomials. Building on a randomized algorithm of Huang and Wong [19], Kayal [22] gave a deterministic algorithm for SysPolyEqs(q) that has running time $d^{n^{O(n)}} \cdot \text{poly}(m, \log q)$, where n is the number of variables, d is the degree, and m is the number of equations. This algorithm is only non-trivial when q is extremely large relative to the degree, in particular $d^{n^{O(n)}} \ll q^n$.

Conditional Lower Bounds Based on SysPolyEqs(2). In some papers, the conjecture that SysPolyEqs(2) in degree-2 *cannot* be solved in $2^{n(1-\varepsilon)}$ time for any $\varepsilon > 0$ was used to justify the optimality of certain algorithms. Björklund *et al.* [10] showed that the conjecture implies that their algorithm for listing triangles in sparse graphs is optimal. Vassilevska and Williams [37] prove that the conjecture implies that finding a zero-edge-sum triangle over $\mathbb{F}_2$ requires $n^{3-o(1)}$ time; this problem is closely related to the 3XOR problem of Jafargholi and Viola [21]. Clearly, Theorem 1.1 refutes the above conjecture, and opens up the possibility for faster algorithms for the above problems.

Relationship to SAT. SysPolyEqs(2) and GenSysPolyEqs(2) may be seen as generalizations of the satisfiability problem for CNF formulas: The former is equivalent to SAT of unbounded-fan-in AND-PARITY-AND circuits, and the latter is SAT of unbounded-fan-in AND-PARITY-AND-PARITY circuits. Small-depth unbounded-fan-in circuits with AND, OR and PARITY gates (i.e., $\text{AC}^0[\oplus]$) are widely studied in Boolean circuit complexity; see, e.g., [24, 15, 14]. To the best of our knowledge, faster-than-2^n SAT algorithms were not known even for depth-3 unbounded-fan-in circuits with a *linear* number of AND and PARITY gates.

Williams [39] gave an algorithm for the ACC^0-SAT problem, which includes SysPolyEqs(2) as a special case; however, the algorithm of Williams only runs in $\text{poly}(n,m) \cdot 2^{n-n^{\varepsilon}}$ time, where ε is a small positive constant. Chapter 4.3 of Matthews' PhD thesis [29] cites Lokshtanov and Paturi (two authors of the present paper) with an unpublished algebraic algorithm for k-SAT. Our algorithms for systems of degree-k equations can be seen as a considerable extensions of their method. Williams and Lokshtanov-Paturi were based on the polynomial method in Boolean circuit complexity [6] (see the survey [40] for more).

Techniques Given a set of degree-k polynomials $S = \{p_1, \ldots, p_m\} \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$, we can define a single polynomial capturing all of them, namely $P_S(x) := 1 - \prod_{i=1}^{m}(1 - p_i(x)^{q-1})$. For all $a \in \mathbb{F}_q^n$, note that $P_S(a) = 0$ holds if $p_1(a) = \cdots = p_m(a) = 0$, and $P_S(a) = 1$ otherwise. For some appropriately chosen $n' < n$, we may define a polynomial $R \in \mathbb{F}_q[x_1, \ldots, x_{n-n'}]$ as $R(x_1, \ldots, x_{n-n'}) := \prod_{a \in \mathbb{F}_q^{n'}} P_S(x_1, \ldots, x_{n-n'}, a)$. Observe that there is an $a \in \mathbb{F}_q^n$ such that $P_S(a) = 0$ if and only if there is a $b \in \mathbb{F}_q^{n-n'}$ such that $R(b) = 0$. Therefore, evaluating R on all points $a \in \mathbb{F}_q^{n-n'}$ will determine if the original system is feasible or not. Evaluation of $R(a)$ would be relatively easy if it were merely a sum of products, but it is more complex: a product of sums of products of sums of monomials. However, since $R(y)$ is a product of $q^{n'}$ functions, it takes time $q^{n'} \cdot \text{poly}(n)$ to evaluate $R(y)$ at a single point y. Hence the straightforward way of evaluating R on $q^{n-n'}$ points would lead to a $q^n \cdot \text{poly}(n)$ running time.

Despite the complexity of R, we can in fact evaluate on $q^{n-n'}$ points in $q^{n-n'} \cdot \text{poly}(n,m)$ time by deftly applying the probabilistic polynomial constructions of Razborov and Smolensky together with an algorithm for efficient sums-of-monomials evaluation. These are among the couple of ideas required to prove Theorem 1.1; attaining a deterministic algorithm requires substantially more work. We borrow some tools from recent work of Chan and Williams [13], who give a more efficient deterministic algorithm for counting SAT assignments. Generalizing the ideas of their #SAT algorithm to fit the SysPolyEqs(q) setting requires some care, in particular, the restriction to prime fields. To make a deterministic algorithm that works for all fields we give a reduction that transforms a system of m polynomial equations of degree k with n variables over $\mathbb{F}_{p^d}$ for $d \geq 2$ into an equivalent system of md polynomial equations of degree k with nk variables over $\mathbb{F}_p$.

This reduction turns out to be also useful to obtain an additional speedup for the randomized algorithm in the case that $q = p^d$.

Our algorithms for solving systems of $\Sigma\Pi\Sigma$ polynomials (Theorems 1.3 and 1.4) consist of two steps. First, given an instance, we run a *degree reduction algorithm* that produces a set of instances of degree at most $O(\log(s/n))$ such that the original instance is satisfiable if and only if at least one of them is satisfiable. Then, we apply the algorithms of Theorems 1.1 or 1.2 to each instance. Our degree reduction algorithm can be seen as a generalization of Schuler's width reduction algorithm for CNF-SAT [34].

The degree reduction algorithm implies that an AND-PARITY-AND-PARITY circuit can be represented as a "small-size" algebraic decision tree whose internal nodes and leaves correspond to indicator functions of affine subspaces and low-degree polynomials respectively. Such representation might be useful in proving average-case lower bounds for AND-PARITY-AND-PARITY circuits, as Impagliazzo, Matthews and Paturi [20] used Schuler's width reduction algorithm to obtain correlation bounds for AC^0 circuits (i.e., bounded-depth unbounded-fan-in circuits with AND and OR gates) for approximating the PARITY function.

2 Preliminaries

We use random access machines as our computation model. For a positive integer n, $[n]$ denotes the set $\{1,2,\dots,n\}$. For rational numbers $a < b$, (a,b) denotes the open interval between a and b. For a finite set S, $|S|$ denotes the cardinality of S. We use the following notation: $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}_{\geq 0}$ denotes the set of non-negative integers, $\mathbb{Z}_m$ denotes the quotient ring of integers modulo m, identified with $\{0,1,\dots,m-1\}$ and $\mathbb{F}_q$ denotes the finite field of order q. We use 0 and 1 to denote the additive identity and the multiplicative identity of $\mathbb{F}_q$.

Let $x_1,x_2,\dots,x_n$ be formal variables. A *monomial* is a product of powers of variables and a constant. For $\gamma \in \mathbb{Z}^n_{\geq 0}$, we define $x^\gamma := \prod_{i\in[n]} x_i^{\gamma_i}$. We can represent a *polynomial* $P(x)$ as a sum of monomials of the form $\sum_\gamma a_\gamma x^\gamma$ where a_γ is the *coefficient* of x^γ. In this paper, we consider polynomials over $\mathbb{F}_q$ and over $\mathbb{Z}$.

When dealing with polynomials over $\mathbb{F}_q$, we will only be concerned with them for the purpose of eventually evaluating them over $\mathbb{F}_q$. The identity $x_i^q = x_i$ then implies that every monomial can be represented as x^γ for $\gamma \in \mathbb{Z}^n_q$, without changing what the monomial evaluates to when the variables takes values from $\mathbb{F}_q$.

The *degree* of a monomial x^γ is $\sum_i \gamma^i$. Thus, the degree of a monomial is the sum of the exponents of the variables in the monomial. We define $M(n,k,q)$ to be the number of different monomials of degree at most k on n variables in $\mathbb{F}_q$, and $M(n,k)$ to be the number of different monomials of degree at most k on n variables over any (fixed, possibly infinite) field. The degree of a polynomial P is the maximum degree of a monomial of P. We use the following facts in Section 3.

Lemma 2.1. *The number of monomials of degree at most k can be upper bounded as follows: $M(n,k,q) \leq M(n,k) \leq \binom{n+k}{n} \leq (1+\frac{n}{k})^k(1+\frac{k}{n})^n \leq e^n(1+\frac{k}{n})^n$.*

Lemma 2.2 (Fast Evaluation in Finite Fields $\langle\star\rangle$)**.** *There is an algorithm that, given an $\mathbb{F}_q$ polynomial P in n variables represented as a sum of monomials, runs in $\mathrm{poly}(n)\cdot q^n$ time and prints a q^n-dimensional vector V such that for all $x \in \mathbb{F}_q^n$, $V[x] = P(x)$ holds.*

Lemma 2.3 (Fast Evaluation of Integer Polynomials $\langle\star\rangle$)**.** *Let n-variate integer polynomial P have at most $p^{n/7}$ monomials such that the maximum absolute value of $P(x)$ over all $x \in \{0,1,\dots,p-1\}^n$ is at most M. Then we can evaluate $P(x)$ over all points in $\{0,1,...,p-1\}^n$ in $\mathrm{poly}(\log M)\cdot p^{n+o(n)}$ time.*

3 Randomized Algorithms for Systems of Polynomial Equations

In this section we give a proof of Theorem 1.1. A degree k instance of SysPolyEqs(q) is $\{p_1,p_2,\dots,p_m\}$, where each p_i is an $\mathbb{F}_q$ polynomial in formal variables $x_1,x_2,\dots,x_n$ and represented as $p_i(x) = a_i + \sum_{j=1}^{s_i} b_{i,j} x^{\gamma_{i,j}}$

for $a_i, b_{i,j} \in \mathbb{F}_q$, $s_i \geq 0$ and $\gamma_{i,j} \in \mathbb{Z}_q^n$ with $\sum_l (\gamma_{i,j})_l \leq k$. Before proceeding with our algorithm for SysPolyEqs(q), we describe the approximation of polynomials by low-degree probabilistic polynomials due to Razborov and Smolensky [33, 35]. We begin with the following lemma whose proof is elementary:

Lemma 3.1. *Let $\mathbb{F}_q$ be a finite field and $v = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_q^n \setminus \{(0,0,\ldots,0)\}$. Select $r = (r_1, r_2, \ldots, r_n)$ from $\mathbb{F}_q^n$ uniformly at random. Then, $\sum_{i \in [n]} r_i v_i$ is distributed uniformly at random over $\mathbb{F}_q$.*

Let $P : \mathbb{F}_q^n \to \mathbb{F}$ be the function such that $P(x) = 1$ if $x = (0,0,\ldots,0)$ and $P(x) = 0$ otherwise, i.e., $P(x) = \prod_{i \in [n]} (1 - x_i^{q-1})$. For $s_1, s_2, \ldots, s_l \in \mathbb{F}_q^n$, define a polynomial $\widetilde{P}_{\{s_i\}_{i=1}^l}(x) := \prod_{i=1}^l \left\{ 1 - \left(\sum_{j \in [n]} (s_i)_j \cdot x_j \right)^{q-1} \right\}$. Razborov and Smolensky showed that if we select random elements $s_1, s_2, \ldots, s_l \in \mathbb{F}_q^n$ uniformly and independently, then $\widetilde{P}_{\{s_i\}_{i=1}^l}(x)$ approximates $P(x)$ with high probability.

Lemma 3.2 ([33, 35])**.** *Select random $s_1, s_2, \ldots, s_l \in \mathbb{F}_q^n$ uniformly and independently. Then, for all $x \in \mathbb{F}_q^n$, $\widetilde{P}_{\{s_i\}_{i=1}^l}(x) \in \{0,1\}$. Furthermore, (i) If $x = (0,0,\ldots,0)$, then $\widetilde{P}_{\{s_i\}_{i=1}^l}(x) = 1$ and (ii) if $x \neq (0,0,\ldots,0)$, then $\mathbf{Pr}_{\{s_i\}_{i=1}^l}[\widetilde{P}_{\{s_i\}_{i=1}^l}(x) = 0] = 1 - q^{-l}$.*

We are now ready to give a randomized algorithm for SysPolyEqs(q) beating brute force. We remark that the constants in the exponent of Lemma 3.3 are not optimized (with the exception for the case that $q = k = 2$), but chosen so as to simplify presentation.

Lemma 3.3. *There is a randomized algorithm that, given an instance of SysPolyEqs(q) with m polynomial equations of degree at most k in n variables, decides the satisfiability of the system correctly with high probability. The running time of the algorithm is bounded by (i) $O^\star(2^{0.8765n})$ when $q = k = 2$, (ii) $O^\star(2^{\left(1-\frac{1}{5k}\right)n} \cdot n^{3k})$ when $q = 2$ and $k > 2$, (iii) $O^\star(q^{(1-(1/(200k)))n} \cdot n^{3qk})$ when $3 \leq q$ and $\log q < 4ek$, and (iv) $O^\star\left(q^n \cdot \left(\frac{\log q}{ek}\right)^{-n}\right)$ when $\log q \geq 4ek$.*

Proof. Let $P : \mathbb{F}_q^n \to \mathbb{F}_q$ be the function such that $P(x) = 1$ if $p_1(x) = p_2(x) = \cdots = p_m(x) = 0$ and $P(x) = 0$ otherwise. We select an integer $n' = \lfloor \delta \cdot n \rfloor$ where the exact value of δ will be set at the end of the proof to be strictly between 0 and 1 depending on k and q. For formal variables $y = (y_1, y_2, \ldots, y_{n-n'})$ and constants $a = (a_1, a_2, \ldots, a_{n'}) \in \mathbb{F}_q^{n'}$, we define $Q(y,a) := P(y_1, y_2, \ldots, y_{n-n'}, a_1, a_2, \ldots, a_{n'})$. Let $R : \mathbb{F}_q^{n-n'} \to \mathbb{F}_q$ be the function such that $R(y) = 0$ if $Q(y,a) = 0$ for every $a \in \mathbb{F}_q^{n'}$ and $R(y) = 1$ otherwise. $P(x)$ is identically 0 if and only if $R(y)$ is identically 0.

We would like to check whether there exists an assignment to the variables such that $R(y) = 1$. For this purpose, we represent $R(y)$ as a sum of monomials and apply the fast evaluation algorithm for polynomials (Lemma 2.2). Note that if we write $R(y)$ as a polynomial in the straightforward manner, e.g., $R(y) = 1 - \prod_{a \in \mathbb{F}_q^{n'}} (1 - Q(y,a))$, and represent it as a sum of monomials, it might take time more than q^n time. This is because $R(y)$ has degree $q^{n'}$ as a polynomial in formal variables $\{Q(y,a)\}_{a \in \mathbb{F}_q^{n'}}$. To reduce the degree of $R(y)$, we use Lemmas 3.2 and 3.1. We start by setting $l = n' + 2$, for each $a \in \mathbb{F}_q^{n'}$ selecting uniformly at random l random vectors $s_{a,1}, s_{a,1}, \ldots, s_{a,l}$ from $\mathbb{F}_q^m$, and defining

$$\widetilde{Q}_{\{s_{a,i}\}_{i=1}^l}(y,a) := \prod_{i=1}^{l} \left\{ 1 - \left(\sum_{j \in [m]} (s_{a,i})_j \cdot p_j(y,a) \right)^{q-1} \right\}.$$

Next, select uniformly at random a $q^{n'}$-dimensional vector s over $\mathbb{F}_q$ and define

$$\widetilde{R}_{s,\{s_{a,i}\}}(y) := \sum_{a \in \mathbb{F}_q^{n'}} s_a \cdot \widetilde{Q}_{\{s_{a,i}\}_{i=1}^l}(y,a).$$

By Lemma 3.2, for all $y \in \mathbb{F}_q^{n-n'}$ and $a \in \mathbb{F}_q^{n'}$, we have that $Q(y,a) = 1$ implies $\widetilde{Q}_{\{s_{a,i}\}_{i=1}^{l}}(y,a) = 1$, while $Q(y,a) = 0$ yields $\mathbf{Pr}_{\{s_{a,i}\}_{i=1}^{l}}[\widetilde{Q}_{\{s_{a,i}\}_{i=1}^{l}}(y,a) = 0] = 1 - q^{-l}$. By Lemma 3.1, for all $y \in \mathbb{F}_q^{n-n'}$ and $\{s_{a,i}\}$, we have that, (i) for all $a \in \mathbb{F}_q^{n'}$, if $\widetilde{Q}_{\{s_{a,i}\}_{i=1}^{l}}(y,a) = 0$ then $\widetilde{R}_{s,\{s_{a,i}\}}(y) = 0$, and "conversely", (ii) if there exists an $a \in \mathbb{F}_q^{n'}$ such that $\widetilde{Q}_{\{s_{a,i}\}_{i=1}^{l}}(y,a) = 1$, then $\mathbf{Pr}_s[\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0] = 1 - \frac{1}{q}$. Thus, for all $y \in \mathbb{F}_q^{n-n'}$, we have

$$R(y) \neq 0 \quad \Rightarrow \quad \mathop{\mathbf{Pr}}_{s,\{s_{a,i}\}}[\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0] \geq 1 - \frac{1}{q} \geq \frac{1}{2},$$

$$R(y) = 0 \quad \Rightarrow \quad \mathop{\mathbf{Pr}}_{s,\{s_{a,i}\}}[\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0] \leq q^{n'} \cdot q^{-l} \leq \frac{1}{4},$$

where we use the union bound for the second implication.

The algorithm repeats the following procedure $t = 100n\log q$ times: It draws the random vectors $\{s_{a,i}\}$ and s, and computes a representation of $\widetilde{R}_{s,\{s_{a,i}\}}$ as a sum of monomials. The procedure then evaluates $\widetilde{R}_{s,\{s_{a,i}\}}(y)$ for all $y \in \mathbb{F}_q^{n-n'}$ using the algorithm of Lemma 2.2. For each $y \in \mathbb{F}_q^{n-n'}$, the algorithm keeps a counter that keeps track of the number of times the above procedure resulted in $\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0$. The algorithm returns that the input instance is satisfiable if there exists a y for which the counter is at least 40 percent of the number of runs, that is, at least $0.4t$.

For the success probability analysis, suppose that the input instance is satisfiable. Then there exists a y such that $R(y) \neq 0$. Thus, in each of the runs of the procedure $\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0$ with probability at least $1/2$. Since each of the runs of the procedure are independent, the probability that the counter for y will be at most $0.4t$ is at most $t \cdot \binom{t}{.4t} 2^{-t} \leq t \cdot \left(\frac{1.961}{2}\right)^t \leq \frac{100n\log q}{q^{2n}}$.

Suppose now that the input instance is not satisfiable. Then $R(y) = 0$ for all choices of y and hence, in each run of the procedure the probability that $\widetilde{R}_{s,\{s_{a,i}\}}(y) \neq 0$ is at most $\frac{1}{4}$. For any fixed y, the probability that the counter of y reaches above $0.4t$ is therefore at most $t \cdot \binom{t}{.4t} \left(\frac{1}{4}\right)^{.4t} \left(\frac{3}{4}\right)^{.6t} \leq t \cdot \left(\frac{1.961}{2.06}\right)^t \leq \frac{100n\log q}{q^{2n}}$. The union bound taken over all $q^{n-n'} \leq q^n$ choices of y yields that the probability of false positives is upper bounded by $\frac{100n\log q}{q^n}$. Hence the algorithm outputs the correct answer with high probability.

We now proceed with the running time analysis. The running time is upper bounded by t times the time taken to execute the main procedure once, we now upper bound this. By Lemma 2.2 it takes $O^\star(q^{n-n'})$ time to evaluate $\widetilde{R}_{s,\{s_{a,i}\}}$ for all y once a representation of $\widetilde{R}_{s,\{s_{a,i}\}}$ as a sum of monomials is given.

To upper bound the time taken to compute the representation of $\widetilde{R}_{s,\{s_{a,i}\}}$ as a sum of monomials it is sufficient to observe that $\widetilde{R}_{s,\{s_{a,i}\}}$ is a polynomial in $n-n'$ variables of total degree at most $k(q-1)l$. We compute the representation of $\widetilde{R}_{s,\{s_{a,i}\}}$ using the definitions directly, applying the naive algorithm for polynomial multiplication. However, we make sure that whenever we are multiplying two polynomials, at least one of them has degree at most kq. Note that this is achievable, because the only multiplications in the definition of $\widetilde{R}_{s,\{s_{a,i}\}}$ occur in the definition of $\widetilde{Q}_{\{s_{a,i}\}_{i=1}^{l}}(y,a)$, which is a product of polynomials of degree at most kq. Thus, the total number of operations (polynomial additions or multiplications) needed to compute $\widetilde{R}_{s,\{s_{a,i}\}}$ is at most $O^\star(q^{n'})$, and each such operation takes time $O^\star(M(n-n', k(q-1)(n'+2), q) \cdot n^{qk})$. Using the observation that $M(n, r+1, q) \leq n \cdot M(n,r,q)$ for every n and r we conclude that the total time taken by the algorithm is upper bounded by

$$O^\star(q^{n-n'} + q^{n'} \cdot M(n-n', k(q-1)n', q) \cdot n^{3qk}) = O^\star(q^{(1-\delta)n} + q^{\delta n} \cdot M((1-\delta)n, k(q-1)\delta n, q) \cdot n^{3qk}).$$

We now discuss the choice of δ for different possible values of q and k. By always picking δ such that $M((1-\delta)n, k(q-1)\delta n, q) = O^\star(q^{(1-2\delta)n})$, we ensure that the running time is upper bounded by $O^\star(q^{(1-\delta)n} \cdot n^{3qk})$. We divide the analysis into three cases, first the case that $q = 2$, then when $q \geq 3$ but q is still "small enough" compared to d, and finally when q is "large" compared to d.

For $q = 2$ and $k = 2$, we set $\delta = 0.1235$ to satisfy $M((1-\delta)n, k(q-1)\delta n, q) \leq \binom{n-\delta n'}{\delta n} \leq O^\star(2^{(1-2\delta)n})$. This implies that a degree-2 instance of SysPolyEqs(2) can be solved in time $O^\star(2^{0.8765n})$.

For $q = 2$ we have that $M(n,k,2) \leq \binom{n}{k}$, thus $M((1-\delta)n, k(q-1)\delta n, q) \leq \binom{n-\delta n}{k\delta n}$. Setting $\delta = 1/(5k)$ we obtain $M((1-\delta)n, k(q-1)\delta n, q) \leq O^\star\left(2^{\left(1-\frac{2}{10}\right)n}\right) \leq O^\star\left(2^{(1-2\delta)n}\right)$. Thus, for $q = 2$ we obtain an algorithm with running time $O^\star(2^{\left(1-\frac{1}{5k}\right)n} \cdot n^{3k})$.

When $q \geq 3$ and $\log q < 4ek$, we set $\delta = 1/(200k)$, then $M((1-\delta)n, k(q-1)\delta n, q) \leq O^*\left(e^n\left(\frac{q}{3} + \frac{q}{200}\right)^n\right)$, using $\log q < 4ek$ we obtain $O^*\left(e^n\left(\frac{q}{3} + \frac{q}{200}\right)^n\right) \leq O^*\left(q^n \cdot 2^{-\frac{\log q \cdot 0.12n}{4ek}}\right) \leq O^*\left(q^{(1-2\delta)n}\right)$. Hence, in this case we obtain an upper bound of $O^\star(q^{(1-(1/200k))n} \cdot n^{3qk})$ on the running time.

When $\log q \geq 4ek$, we set $\delta = \frac{\log\log q^{(1/ek)}}{4\log q}$, which yields $M((1-\delta)n, k(q-1)\delta n, q) \leq O^*\left((2ekq\delta)^n\right)$. Inserting the definition of δ we obtain $M((1-\delta)n, k(q-1)\delta n, q) \leq O^*\left(\left(q \cdot \frac{\log\log q^{(1/ek)}}{2\cdot\log q^{(1/ek)}}\right)^n\right)$. Now, $q^{1/ek} \geq 16$ implies that $\left(\frac{2\cdot\log q^{(1/ek)}}{\log\log q^{(1/ek)}}\right)^{\frac{2\cdot\log q^{(1/ek)}}{\log\log q^{(1/ek)}}} \geq q^{1/ek}$. Thus we have

$$O^*\left(\left(q \cdot \frac{\log\log q^{(1/ek)}}{2\cdot\log q^{(1/ek)}}\right)^n\right) \leq O^*\left(\left(q^{1-\frac{1}{ek}\cdot\frac{\log\log q^{(1/ek)}}{2\cdot\log q^{(1/ek)}}}\right)^n\right) \leq O^*\left(q^{(1-2\delta)n}\right).$$

Hence, in this case we get an upper bound of $O^\star\left(q^n \cdot \left(\frac{\log q}{ek}\right)^{-n}\right)$ on the running time. □

Next we show that any system of m polynomial equations of degree k with n variables over $\mathbb{F}_{p^d}$ for $d \geq 2$ can be reduced in polynomial time to an equivalent system of md polynomial equations of degree k with nk variables over $\mathbb{F}_p$. This allows us to substantially improve over the running time of the algorithm of Lemma 3.3 for the case when q is large compared to d, and q is a prime power p^d, $d \geq 2$.

Let p be prime. In the following, we will assume we possess an irreducible polynomial $P(X)$ of degree k in $\mathbb{F}_p[X]$. A standard way of efficiently constructing such $P(X)$ is to choose degree-k polynomials P at random, then test them for irreducibility. The irreducibility of P can then be checked by running Kedlaya-Umans' deterministic irreducibility test in $k^{1+o(1)}\log^{2+o(1)} p$ time ([23], Section 8.2); a standard algebraic fact is that a random polynomial over $\mathbb{F}_p$ of degree k is irreducible with probability at least $1/k$. We are now ready to give the aforementioned reduction.

Lemma 3.4. *There is a polynomial-time algorithm that given as input a system S of m degree-k n-variate polynomial equations over $\mathbb{F}_{p^d}$, together with an irreducible polynomial $P(X)$ of degree d in $\mathbb{F}_p[X]$, outputs an equivalent system S_p of dm degree-k dn-variate equations over $\mathbb{F}_p$. That is, S has a solution over $\mathbb{F}_{p^d}$ if and only if S_p has a solution over $\mathbb{F}_p$.*

Proof. For every $0 \leq \ell \leq (d-1)k$, compute degree-$(d-1)$ polynomials $P_\ell \in \mathbb{F}_p[X]$ such that $X^\ell \equiv P_\ell(X) \pmod{P(X)}$. These P_ℓ can be determined by simple polynomial division, in poly$(d\cdot k)$ time. Let x_i be the i'th variable in the system S; we intend to set x_i to a value in $\mathbb{F}_{p^d}$. $\mathbb{F}_{p^d}$ is isomorphic to $\mathbb{F}_p[X]/P$, that is, the elements of $\mathbb{F}_{p^d}$ can be thought of as equivalence classes of polynomials in $\mathbb{F}_p[X]$ modulo P(X). It is a basic fact that every element r of $\mathbb{F}_p[X]/P$ can be written as $r = \sum_{l=0}^{d-1} r_i X^\ell$ for $r_i \in \mathbb{F}_p$ in a unique way. Thus, for each $i \leq n$ and $0 \leq \ell \leq d-1$ we can define $x_{i,\ell} \in \mathbb{F}_p$ such that $x_i \equiv \sum_{l=0}^{d-1} x_{i,\ell} X^\ell$. That is, we think of the value of x_i as a d-dimensional vector with entries from $\mathbb{F}_p$, where $x_{i,\ell}$ is the ℓ'th component of this vector. Consider now the product $x_i x_j$, we have that $x_i x_j \equiv \sum_{l_1,l_2=0}^{d-1} x_{i,\ell_1} x_{i,\ell_2} P_{\ell_1+\ell_2}$. Since all P_ℓ's are fixed polynomials in $\mathbb{F}_p[X]$, the coefficients of every X^ℓ are quadratic forms over the variables $\{x_{i,\ell}\}_{0\leq\ell<d}$ and $\{x_{j,\ell}\}_{0\leq\ell<d}$.

In general, a product of k variables can be viewed as a degree-d polynomial of X, whose coefficients are degree-k forms of $\{x_{i,\ell}\}_{0\le i<n, 0\le \ell<d}$. Therefore, a single degree-k polynomial equation over the variables $\{x_i\}_{0\le i<n}$ over $\mathbb{F}_{p^d}$ can be viewed as a system of d degree-k polynomial equations over the variables $\{x_{i,\ell}\}_{0\le i<n, 0\le \ell<d}$ over $\mathbb{F}_p$. Doing this for every equation in the input system increases the number of variables and the number of equations by a factor of d, but reduces the underlying field from $\mathbb{F}_{p^d}$ to $\mathbb{F}_p$. □

We may now directly combine the algorithm of Lemma 3.3 with the reduction of Lemma 3.4 to obtain improved savings for SysPolyEqs(q) when q is large compared to k and not prime. In particular, applying Lemma 3.4 and then solving the output instance using Lemma 3.3 yields a proof of Theorem 1.1.

4 A Degree Reduction Algorithm

In this section we prove Theorems 1.3 and 1.4. Specifically, we present algorithms for a generalization of SysPolyEqs(2), where each p_i is a polynomial of the form

$$p_i(x) = a_i + \sum_{j=1}^{s_i} \prod_{k=1}^{t_{i,j}} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right) \tag{1}$$

for $a_i, b_{i,j,k} \in \{0,1\}$, $s_i, t_{i,j} \ge 1$ and $\emptyset \ne U_{i,j,k} \subseteq \{1,2,\dots,n\}$. Let $s := \sum_i s_i$ and $u := \sum_{i,j,k} |U_{i,j,k}|$ denote the *number of products* and the *size* respectively. When the polynomials in the input system are given in the form 1, we refer to the problem as GenSysPolyEqs(2). Our algorithm works by reducing systems of polynomial equations over $\mathbb{F}_2$ where each polynomial is in the form 1 to systems of polynomial equations over $\mathbb{F}_2$ where the degree of the polynomials of the output system depends on the number of products in the input system. This reduction together with Theorems 1.1 and 1.2 will complete the proofs of Theorems 1.3 and 1.4, respectively.

Note that because we are working in $\mathbb{F}_2$, $x_\ell^d = x_\ell$ for all $d \ge 1$, and that therefore the degree of a monomial is equal to the number of variables in it. The degree of a polynomial p_i is at most $\max t_{i,j}$, however it is possible that it is actually less. In this section we will abuse terminology and for each i refer to the degree of the polynomial p_i as $\deg(p_i) := \max t_{i,j}$.

Tools from Linear Algebra. We need the following standard notions and properties of linear independence of $\mathbb{F}_2$. Let V be a set of vectors $\{v_1, v_2, \dots, v_t\} \subseteq \{0,1\}^n$. We say V is *linearly dependent* if there exists a non-zero vector $(a_1, a_2, \dots, a_t) \in \{0,1\}^t$ such that $\sum_{i=1}^t a_i v_i = (0,0,\dots,0)$ holds. Otherwise, V is *linearly independent*. If V is linearly independent and $V \cup \{v\}$ is linearly dependent for all $v \in \{0,1\}^n \setminus V$, then V is *maximally linearly independent*. The *rank* of V, denoted by rank(V), is the maximum cardinality of a linearly independent subset of V. For any maximally linearly independent subset V' of V, the cardinality of V' is equal to rank(V).

Let V be a subset of $\{0,1\}^n$ and V' be a linearly independent subset of $\{0,1\}^n$. The *rank of V relative to V'* is the maximum cardinality of a subset V'' of V such that $V' \cup V''$ is linearly independent. Note that any vector in V can be written as a linear combination of vectors in $V' \cup V''$. In what follows, it is convenient to identify the vector $v \in \{0,1\}^n$ with the set $S_v = \{i \in [n] \mid v_i = 1\}$ and the linear form $L = \sum_{i=1}^n v_i x_i$. Thus, we use terms such as linearly independent and rank for a set of subsets of $[n]$ or a set of linear forms in a natural way. For linear forms $L_1, L_2, \dots, L_t$ and $a_1, a_2, \dots, a_t \in \{0,1\}$, we identify the system of linear equations $\{L_i = a_i\}_{i=1}^t$ with the affine subspace $\{x \in \{0,1\}^n \mid L_1 = a_1, L_2 = a_2, \dots, L_t = a_t\}$. The following lemma gives a way to reduce the degree of polynomials all of whose products of sums of variables have "low rank."

Lemma 4.1 (Degree reduction relative to a system of linear equations $\langle\star\rangle$)**.** *Let $U_1, U_2, \dots, U_t$ be subsets of $[n]$ and $L_1, L_2, \dots, L_{t'}$ be linear forms such that $\{L_i\}_{i=1}^{t'}$ is linearly independent and the rank of $\{U_i\}_{i=1}^t$ relative to $\{L_i\}_{i=1}^{t'}$ is d. Then, for all $a_1, a_2, \dots, a_{t'}, b_1, b_2, \dots, b_t \in \{0,1\}$, there exists a polynomial p of degree at most d such that $p(x) = \prod_{k=1}^t \left(b_k + \sum_{l \in U_k} x_l\right)$ holds for all $x \in \{L_i = a_i\}_{i=1}^{t'}$.*

Simplification of systems of polynomial equations. Before describing our main algorithm, we introduce a procedure that simplifies instances of GenSysPolyEqs(2). Let $P = \{p_1, p_2, \ldots, p_m\}$ be an instance of GenSysPolyEqs(2). Let $\deg(P) = \max \deg(p_i)$. We partition P into P_1 and P_2 such that $P_1 = \{p \in P | \deg(p) \leq 1\}$ and $P_2 = \{p \in P | \deg(p) \geq 2\}$. Note that we can check whether P_1 is satisfiable or not in polynomial time via Gaussian elimination. In what follows, we assume $p_i(x) = a_i + \sum_{j \in U_i} x_j$ if $\deg(p_i) \leq 1$ and

$$p_i(x) = a_i + \sum_{j=1}^{s_i} \prod_{k=1}^{t_{i,j}} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right)$$

if $\deg(p_i) \geq 2$. Given a positive integer d, we define the procedure **Simplify**(P, d) as follows:

1. Simplify P_1: Assume $P_1 = \{p_1, p_2, \ldots, p_{m'}\}$. First, check whether a system of linear equations $p_1 = p_2 = \cdots = p_{m'} = 0$ is satisfiable. If the system is not satisfiable, return unsatisfiable, and continue otherwise. Select a maximally linearly independent subset V of $\{U_i\}_{i=1}^{m'}$. Redefine $P_1 := \{p_i\}_{U_i \in V}$.
2. Simplify P_2 with P_1: Assume $P_1 = \{p_1, p_2, \ldots, p_{m'}\}$ and $P_2 = \{p_{m'+1}, p_{m'+2}, \ldots, p_m\}$. For $m' + 1 \leq i \leq m$, $1 \leq j \leq s_i$ and $1 \leq k \leq t_{i,j}$, if $U_{i,j,k}$ can be written as a linear combination of vectors in $\{U_i\}_{i=1}^{m'}$ as $U_{i,j,k} = \sum_{i=1}^{m'} c_i U_i$ for some $c_1, c_2, \ldots, c_{m'} \in \{0, 1\}$, replace $\sum_{l \in U_{i,j,k}} x_l$ by $\sum_{i=1}^{m'} c_i a_i$. If $\deg(p_i) \leq 1$ for some i, $m' + 1 \leq i \leq m$ after the substitution, remove p_i from P_2, add it to P_1 and go back to Step 1.
3. Degree Reduction: Again, assume $P_1 = \{p_1, p_2, \ldots, p_{m'}\}$ and $P_2 = \{p_{m'+1}, p_{m'+2}, \ldots, p_m\}$. For $m' + 1 \leq i \leq m$ and $1 \leq j \leq s_i$, if the rank of $\{U_{i,j,k}\}_{k=1}^{t_{i,j}}$ with respect to $\{U_i\}_{i=1}^{m'}$ is at most d, rewrite $\prod_{k=1}^{t_{i,j}} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right)$ as a polynomial of degree at most d by Lemma 4.1. If $\deg(p_i) \leq 1$ for some i, $m' + 1 \leq i \leq m$ after the above rewriting, remove p_i from P_2, add it to P_1 and go back to Step 1.

Let P' be the simplified instance obtained by applying **Simplify** to P. The following is true: (P1) The number of satisfying assignments for P is equal to that for P'. (P2) Partition the resulting instance P' into P'_1 and P'_2 and assume $P'_1 = \{p_1, p_2, \ldots, p_{m'}\}$ and $P'_2 = \{p_{m'+1}, p_{m'+2}, \ldots, p_m\}$. Then, (i) $\{U_i\}_{i=1}^{m'}$ is linearly independent, and (ii) for $m' + 1 \leq i \leq m$ and $1 \leq j \leq s_i$, we have either $t_{i,j} \leq d$ or the rank of $\{U_{i,j,k}\}_{k=1}^{t_{i,j}}$ with respect to $\{U_i\}_{i=1}^{m'}$ is at least $d + 1$.

The algorithm and its analysis. We now describe the algorithm **Degree-Reduction**(P, d), which we will use to prove Theorems 1.3 and 1.4. We first finish the proof of Theorem 1.3 and then describe the necessary adjustments to prove Theorem 1.4.

1. Run **Simplify**(P, d).

2. If $\deg(P) \leq d$, assume $P_1 = \{p_1, p_2, \ldots, p_{m'}\}$ and select an arbitrary subset V of $\{0, 1\}^n$ such that $\{U_i\}_{i=1}^{m'} \cup V$ is maximally linearly independent, i.e., has rank n. Rewrite each variable x_i as a linear combination of vectors in $\{U_i\}_{i=1}^{m'} \cup V$. We regard each vector in V as a formal variable and each vector U_i as the constant a_i. Apply Theorem 1.1 to the resulting instance in $\left(n - \operatorname{rank}(\{U_i\}_{i=1}^{m'}) \right)$ variables.

3. If $\deg(P) > d$, assume $P_1 = \{p_1, p_2, \ldots, p_{m'}\}$ and select arbitrary $i, j', m' + 1 \leq i \leq m, 1 \leq j' \leq s_i$, such that $p_i(x) = a_i + \sum_{j=1}^{s_i} \prod_{k=1}^{t_{i,j}} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right)$ and $t_{i,j'} > d$. Select an arbitrary subset V of $\{U_{i,j',k}\}_{k=1}^{t_{i,j'}}$ of size d such that $V \cup \{U_i\}_{i=1}^{m'}$ is linearly independent. Define

$$q_i(x) := \prod_{U_{i,j,k} \in V} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right) \quad \text{and} \quad r_i(x) = a_i + \sum_{1 \leq j \leq s_i, j \neq j'} \prod_{k=1}^{t_{i,j}} \left(b_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l \right).$$

Define instances $P_L := \{P \setminus \{p_i\}\} \cup \{q_i, r_i\}$, and $P_R := P \cup \{\bar{b}_{i,j,k} + \sum_{l \in U_{i,j,k}} x_l\}_{U_{i,j,k} \in V}$. Then, run **Degree-Reduction**(P_L, d) and **Degree-Reduction**(P_R, d) recursively.

The correctness of **Degree-Reduction** is guaranteed since P is satisfiable if and only if at least one of P_L and P_R is satisfiable. This is because in Step 3, instances P_L and P_R correspond to conditions $q_i(x) = 0$ and $q_i(x) = 1$ respectively. Note that we can choose V in Step 3 due to Property (P2)-(ii) of **Simplify**. In what follows, we give the running time analysis of **Degree-Reduction**(P, d). The overall structure is the same as the analysis of Schuler's width reduction algorithm for SAT of CNF formulas in [12].

We regard the execution of **Degree-Reduction** as a rooted binary tree T. The root of T is labeled with an input instance P. For each node labeled with Q, its left (right, resp.) child is labeled with Q_L (Q_R, resp.) as defined in Step 3 of **Degree-Reduction**. If $\deg(Q) \le d$ holds, then the node labeled with Q is a leaf. Let us consider a path p from the root to a leaf v labeled with Q. We denote by L and R the number of left and right children p selects to reach v. We see that (1) $L \le s$ since the number of products with degree more than d is at most s, (2) $R \le n/d$ since a right branch increases the rank of P_1 (as a set of linear forms) by d and the rank of P_1 cannot be larger than n, and (3) Q is defined over at most $n - dR$ variables in the sense of Step 2 of **Degree-Reduction**. Furthermore, the number of leaves that are reachable by exactly R times of right branches is at most $\binom{s+R}{R}$. Let $T(n,m,d)$ denote the running time of the algorithm of Theorem 1.1 on instances of SysPolyEqs with m polynomial equations of degree at most d in n variables. We can upper bound the running time of **Degree-Reduction** as follows:

$$
\begin{aligned}
& O^\star\left(\sum_{R=0}^{\frac{n}{d}} \binom{s+R}{R} \cdot T(n-dR, m, d)\right) \le O^\star\left(\sum_{R=0}^{\frac{n}{d}} \binom{s+R}{R} \cdot 2^{\left(1-\frac{1}{5d}\right)(n-dR)}\right) \\
\le \quad & O^\star\left(\sum_{R=0}^{s+\frac{n}{d}} \binom{s+\frac{n}{d}}{R} \cdot 2^{\left(1-\frac{1}{5d}\right)(n-dR)}\right) = O^\star\left(2^{\left(1-\frac{1}{5d}\right)n} \cdot \left(1+2^{-d\left(1-\frac{1}{5d}\right)}\right)^{s+\frac{n}{d}}\right) \\
\le \quad & O^\star\left(2^{\left(1-\frac{1}{5d}\right)n} \cdot \exp\left\{2^{-d\left(1-\frac{1}{5d}\right)}\left(s+\frac{n}{d}\right)\right\}\right) \le O^\star\left(\mathrm{poly}(u) \cdot 2^{\left(1-\frac{1}{5d}\right)n} \cdot 2^{4s/2^{d-1/5}}\right),
\end{aligned}
$$

where we assume $s \ge n/d$. We set $d := 2\log(s/n) + c$ for sufficiently large $c > 0$, then $\left(1 - \frac{1}{5d}\right)n + \frac{4s}{2^{d-1/5}} \le \left(1 - \frac{1}{10\log(s/n)+5c} + \frac{1}{(s/n)2^{c-3}}\right)n$. This completes the proof of Theorem 1.3.

We see that (1) **Simplify** does not change the number of satisfying assignments by Property P-1, and (2) each branching of **Degree-Reduction** only partitions the solution space. This implies that if we replace Theorem 1.1 by Theorem 1.2 in Step 2 of **Degree-Reduction** and add the number of satisfying assignments of P_L and that of P_R in Step 3, we obtain Theorem 1.4.

5 Concluding Remarks

We have shown how multivariate systems of polynomial equations can be solved faster than exhaustive search in very generic settings. There are two natural extensions that we have not yet been able to crack:

Is there an algorithm for SysPolyEqs(q) with a better runtime exponent? Our savings over exhaustive search for SysPolyEq(p^d) in the exponent is $n/O(k)$ for degree-k polynomials when $p \le 2^{4ek}$, and $n \cdot O(\frac{\log\log q - \log 4ek}{\log q})$ otherwise. Can one achieve savings $n/O(k)$ even for large p compared to k? Can such savings be achieved by a deterministic algorithm? We remark that removing the factor of $1/k$ from the savings entirely would refute the Strong Exponential Time Hypothesis (k-SAT can easily be embedded into degree-k instances of SysPolyEqs(2)).

Is there an algorithm for SAT of large-depth arithmetic circuits over $\mathbb{F}_p$? Arbitrary arithmetic circuits? Our algorithm for GenSysPolyEqs(2) already solves the SAT problem for $\Pi\Sigma\Pi\Sigma$ circuits which is a considerable generalization of CNF-SAT. By results of Agrawal and Vinay [1] who reduce arbitrary small low-degree circuits to subexponential-size ΣΠΣΠ circuits, we can already conclude a non-trivial SAT algorithm for any $\mathbb{F}_2$-arithmetic circuit of degree less than $n^{2-\varepsilon}$ (by a randomized reduction).

Acknowledgments. The fourth author thanks Timothy Chan for helpful discussions on an earlier phase of this work.

References

[1] M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 67–75, 2008.

[2] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[3] G. Bard. *Algebraic cryptanalysis*. Springer Science & Business Media, 2009.

[4] G. V. Bard, N. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. *IACR Cryptology ePrint Archive*, 2007:24, 2007.

[5] M. Bardet, J. Faugère, B. Salvy, and P. Spaenlehauer. On the complexity of solving quadratic Boolean systems. *J. Complexity*, 29(1):53–75, 2013.

[6] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the 8th Annual Structure in Complexity Theory Conference*, pages 82–95, 1993.

[7] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

[8] A. Bhowmick and S. Lovett. Bias vs structure of polynomials in large fields, and applications in effective algebraic geometry and coding theory. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-22, 2015.

[9] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[10] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 223–234, 2014.

[11] M. Bronstein, A. M. Cohen, H. Cohen, D. Eisenbud, B. Sturmfels, A. Dickenstein, and I. Z. Emiris. *Solving Polynomial Equations: Foundations, Algorithms, and Applications*. Springer, 2005.

[12] C. Calabro, R. Impagliazzo, and R. Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 252–260, 2006.

[13] T. M. Chan and R. Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

[14] M. Cheraghchi, E. Grigorescu, B. Juba, K. Wimmer, and N. Xie. $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ lower bounds for the Boolean inner product. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2016, to appear.

[15] G. Cohen and I. Shinkar. The complexity of DNF of parities. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 47–58, 2016.

[16] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Proceedings of the 19th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 392–407, 2000.

[17] J. Ding, J. E. Gower, and D. Schmidt. *Multivariate Public Key Cryptosystems*, volume 25 of *Advances in Information Security*. Springer, 2006.

[18] J. Håstad. Satisfying degree-d equations over GF$[2]^n$. *Theory of Computing*, 9:845–862, 2013.

[19] M. A. Huang and Y. Wong. Solving systems of polynomial congruences modulo a large prime (extended abstract). In *Proceedings of 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 115–124, 1996.

[20] R. Impagliazzo, W. Matthews, and R. Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 961–972, 2012.

[21] Z. Jafargholi and E. Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016.

[22] N. Kayal. *Derandomizing some number-theoretic and algebraic algorithms*. PhD thesis, Indian Institute of Technology Kanpur, 2006.

[23] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[24] S. Kopparty and S. Srinivasan. Certifying polynomials for $\mathrm{AC}^0[\oplus]$ circuits, with applications. In *Proceedings of the 32nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 36–47, 2012.

[25] S. Kopparty and S. Yekhanin. Detecting rational points on hypersurfaces over finite fields. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC)*, pages 311–320, 2008.

[26] A. G. B. Lauder and D. Wan. Counting points on varieties over finite fields of small characteristic. In J. P. Buhler and P. Stevenhagen, editors, *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, pages 579–612. Cambridge University Press, 2008.

[27] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, 2012.

[28] C. Lu. Hitting set generators for sparse polynomials over any finite fields. In *Proceedings of the 27th Conference on Computational Complexity (CCC)*, pages 280–286, 2012.

[29] W. G. Matthews. *A satisfiability algorithm for constant depth Boolean circuits with unbounded fan-in gates*. PhD thesis, UC San Diego, 2011.

[30] H. Miura, Y. Hashimoto, and T. Takagi. Extended algorithm for solving underdefined multivariate quadratic equations. In *Proceedings of the 5th International Workshop on Post-Quantum Cryptography (PQCrypto)*, pages 118–135, 2013.

[31] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[32] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[33] A. Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. *Mathematical Notes of the Academy of Sci. of the USSR*, 41(4):333–338, 1987.

[34] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

[35] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 77–82, 1987.

[36] B. Sturmfels. *Solving Systems of Polynomial Equations*. American Mathematical Society, 2002.

[37] V. Vassilevska Williams and R. Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

[38] R. Williams. A casual tour around a circuit complexity bound. *SIGACT News*, 42(3):54–76, 2011.

[39] R. Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2, 2014.

[40] R. Williams. The polynomial method in circuit complexity applied to algorithm design (invited talk). In *Proceedings of the 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 47–60, 2014.

[41] A. R. Woods. Unsatisfiable systems of equations, over a finite field. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 202–211, 1998.

[42] B. Yang and J. Chen. Theoretical analysis of XL over small fields. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy (ACISP)*, pages 277–288, 2004.

Appendix

A Fast Polynomial Evaluation

Lemma 2.2 (restated). *There is an algorithm that, given an $\mathbb{F}_q$ polynomial P in n variables, represented as a sum of monomials, runs in* $\mathrm{poly}(n) \cdot q^n$ *time and prints a q^n-dimensional vector V such that for all $x \in \mathbb{F}_q^n$, $V[x] = P(x)$ holds.*

Proof. This is a generalization of the algorithm from Section 6.2 in [38]. Note that we can write P as $P(x) = \sum_{i=0}^{q-1} P_i(x_2, \ldots, x_n) x_1^i$ for some $P_i : \mathbb{F}_q^{n-1} \to \mathbb{F}_q$. This gives us a way to decompose the problem into q subproblems. The running time $T(n)$ of the algorithm satisfies $T(n) = qT(n-1) + \mathrm{poly}(n) q^n$ and we have $T(n) = \mathrm{poly}(n) \cdot q^n$. $\square$

Lemma 2.3 (restated). *Let n-variate integer polynomial P have at most $p^{n/7}$ monomials such that the maximum absolute value of $P(x)$ over all $x \in \{0, 1, \ldots, p-1\}^n$ is at most M. Then we can evaluate $P(x)$ over all points in $\{0, 1, ..., p-1\}^n$, in* $\mathrm{poly}(\log M) \cdot p^{n+o(n)}$ *time.*

Lemma 2.3 does not follow from the usual techniques, such as dynamic programming [9] or divide and conquer [38], because P could have arbitrary degree. Below we give a proof sketch.

Proof. Let $t \leq p^{n/7}$ be the number of monomials. Let $a_1 \ldots, a_{p^{n/2}} \in \{0, ..., p-1\}^{n/2}$ be a list of all assignments to $n/2$ variables. Let $m_1 = c_1 x^{\gamma_1}, \ldots, m_t = c_t x^{\gamma_t}$ be a list of monomials in P. Prepare matrices A and B of $p^{n/2}$ by $p^{n/7}$ and $p^{n/7}$ by $p^{n/2}$ dimensions, respectively, with the following definitions: $A[i, j] = m_j(a_i, \vec{1})$ and $B[j, k] = m_j(\vec{1}, a_k)/c_j$, where $\vec{1}$ denotes the all-ones assignment on $n/2$ variables.

Observe that $A[i, j]B[j, k] = m_j(a_i, a_k)$. It follows that $(AB)[i, k] = \sum_j m_j(a_i, a_k) = P(a_i, a_k)$. Thus, a matrix multiplication of A and B yields the value of P on all points.

LeGall [27] gives a matrix multiplication algorithm which can multiply N by $N^{0.3}$ and $N^{0.3}$ by N matrices in $N^{2+o(1)}$ arithmetic operations, over any finite field. We can therefore multiply A and B over any field $\mathbb{F}_q$ in $O(p^{n+o(n)})$ arithmetic operations. Select q to be a prime greater than $2M + 1$. Our matrix multiplication now takes $\mathrm{poly}(\log M) p^{n+o(n)}$ time.

Now for all $(i, j) \in [p^{n/2}]^2$, consider $(AB)[i, j]$, and cast it as an integer $r \in \{0, 1, \ldots, q-1\}$. If $r \in \{0, 1, \ldots, M\}$ then output r as the value of $P(a_i, a_j)$. Otherwise, r must be in $\{q - M, \ldots, q-1\}$; output $r - q$ as the value of $P(a_i, a_j)$. $\square$

B Deterministic Algorithms for Systems of Polynomial Equations

We now show how to obtain a deterministic algorithm for SysPolyEqs(q) for that beats the naive $O^\star(q^n)$ time algorithm, in particular we prove Theorem 1.2. We will first give such an algorithm for SysPolyEqs(p) for prime p. We start with a "deterministic version" of the probabilistic polynomial constructions of Razborov and Smolensky following the lead of Chan and Williams [13]. We need a construction of *small-biased spaces*.

Definition B.1 (Naor and Naor [31])**.** *A set $S \subseteq \mathbb{F}_p^n$ of n-dimensional vectors is ε-*biased *if for all non-zero $v \in \mathbb{F}_p^n$ and all $a \in \mathbb{F}_p$,*

$$\Pr_{r \in S}\left[\sum_{i \in [n]} r_i v_i = a\right] \in (1/p - \varepsilon, 1/p + \varepsilon).$$

Theorem B.2 (Alon, Goldreich, Håstad and Peralta [2]). *For every positive integer n and $\varepsilon \in (0, 1/q)$, there is an ε-biased set $S_{n,\varepsilon} \subseteq \mathbb{F}_p^n$ of cardinality $O(n^3(\log_2^3 p)/\varepsilon^3)$, constructible in time $\mathrm{poly}(n(\log_2 p)/\varepsilon)$.*

We also need constructions of *modulus-amplifying polynomials*:

Lemma B.3 (Beigel and Tarui [7]). *For every positive integer ℓ, the degree $(2\ell - 1)$ integer polynomial*

$$F_\ell(y) = 1 - (1-y)^\ell \sum_{j=0}^{\ell-1} \binom{\ell + j - 1}{j} y^j$$

has the property for all $p \in \mathbb{Z}$:

- *if $y = 0 \bmod p$, then $F_\ell(y) = 0 \bmod p^\ell$,*
- *if $y = 1 \bmod p$, then $F_\ell(y) = 1 \bmod p^\ell$.*

In addition, for $0 \le i \le 2\ell - 1$, the coefficient of y^i in F_ℓ has magnitude at most $2^{O(\ell)}$.

Now we are ready to "derandomize" the probabilistic polynomial constructions of Razborov and Smolenskyas follows. For a non-empty set $S \subseteq \mathbb{F}_p^n$ and a positive integer ℓ, define a polynomial $\widehat{p}_{S,\ell}(x) : \mathbb{F}_p^n \to \mathbb{Z}$ as

$$\widehat{p}_{S,\ell}(x) := \sum_{r \in S} F_\ell \left(\left(\sum_{i \in [n]} r_i x_i \right)^{p-1} \right),$$

where we regard $\mathbb{F}_p$ as the set of integers $\{0, 1, \dots, p-1\} \subset \mathbb{Z}$. Then, we have:

Lemma B.4. *Let $S \subseteq \mathbb{F}_p^n$ be an ε-biased set and ℓ be a positive integer such that $p^\ell > |S|$. Then:*

- *If $x = (0, 0, \dots, 0)$, then $\widehat{p}_{S,\ell}(x) = 0$.*
- *If $x \neq (0, 0, \dots, 0)$, then $(\widehat{p}_{S,\ell}(x) \bmod p^\ell) \in ((1 - 1/p - \varepsilon)|S|, (1 - 1/p + \varepsilon)|S|)$.*

Proof. The first item is by the fact that $F_\ell(0) = 0$. The second item is by the fact that for any $a \neq 0 \bmod p$, $a^{p-1} = 1 \bmod p$ holds and the definition of ε-biased set. □

We are now prepared to give the deterministic algorithm for fields of prime order:

Lemma B.5. *Let p be a prime. There is a deterministic algorithm that, given an instance of SysPolyEqs(p) with m polynomial equations of degree at most k in n variables, runs in time $p^{n\left(1 - \frac{1}{300kp^{6/7}}\right) + o(n)} \cdot m^{O(pk)}$ and counts the number of satisfying assignments for the system.*

Proof. Let $\{p_1, p_2, \dots, p_m\}$ be an instance of SysPolyEqs(p). We select an integer $n' = \lfloor \delta \cdot n \rfloor$, where the exact value of δ will be set at the end of the proof, strictly between 0 and 1, depending on k and q. Define a function $Q : \mathbb{F}_p^{n-n'} \times \mathbb{F}_p^{n'} \to \mathbb{F}_p$ as $Q(y, a) = 1$ if $p_1(y, a) = p_2(y, a) = \cdots = p_m(y, a) = 0$ and $Q(y, a) = 0$ otherwise. Also we define a function $K : \mathbb{F}_p^{n-n'} \to \mathbb{Z}$ as $K(y) := |\{a \in \mathbb{F}_p^{n'} \mid Q(y, a) = 0\}|$. Note that $K(y)$ represents the number of unsatisfying assignments when the first $n - n'$ variables are fixed to y. If we have the value of $K(y)$ for all $y \in \mathbb{F}_p^{n-n'}$, we can compute the number of satisfying assignments to the input instance, i.e., $p^n - \sum_{y \in \mathbb{F}_p^{n-n'}} K(y)$, in time $\mathrm{poly}(n) \cdot p^{n-n'}$.

In what follows, we show how to construct an integer polynomial in the same $n - n'$ variables as $K(y)$, such that for every $y \in \mathbb{F}_p^{n-n'}$, $K(y)$ can be efficiently determined from the value of the polynomial evaluated on y. For a nonempty set $S \subseteq \mathbb{F}_p^m$ and a positive integer ℓ, define the integer polynomials

$$\widehat{Q}_{S,\ell}(y, a) := \sum_{r \in S} F_\ell \left(\left(\sum_{i \in [m]} r_i \cdot p_i(y, a) \right)^{p-1} \right), \qquad \widehat{R}_{S,\ell}(y) := \sum_{a \in \mathbb{F}_p^{n'}} \widehat{Q}_{S,\ell}(y, a).$$

Here we regard $\mathbb{F}_p$ as the set of integers $\{0,1,\ldots,p-1\}$ and each p_i as an integer polynomial whose coefficients are from $\{0,1,\ldots,p-1\}$ in a natural way.

Let $\varepsilon := 1/(4\cdot p^{n'})$, and construct an ε-biased set $S \subseteq \mathbb{F}_p^m$ using Theorem B.2. We have that $|S|$ is at most $m^3 p^{3n'+O(1)}$, and that S is constructed in $O^\star(p^{O(n')})$ time. Now, let ℓ be the smallest integer greater than $|S|p^{n'}$, note that $\ell \le 4n' + \log m + O(1)$. By Lemma B.4, for all $y \in \mathbb{F}_p^{n-n'}$ and $a \in \mathbb{F}_p^{n'}$, we have that

$$\begin{aligned} Q(y,a) = 1 \quad &\Rightarrow \quad \widehat{Q}_{S,\ell}(y,a) = 0, \\ Q(y,a) = 0 \quad &\Rightarrow \quad (\widehat{Q}_{S,\ell}(y,a) \bmod p^\ell) \in ((1-1/p-\varepsilon)|S|, (1-1/p+\varepsilon)|S|). \end{aligned}$$

Then, for all $y \in \mathbb{F}_p^{n-n'}$, we have

$$(\widehat{R}_{S,\ell}(y) \bmod p^\ell) \in ((1-1/p-\varepsilon)|S|K(y), (1-1/p+\varepsilon)|S|K(y)).$$

Let $M := (1-1/p)|S|$ and recall that $\varepsilon := 1/(4\cdot p^{n'})$. For all $y \in \mathbb{F}_p^{n-n'}$, we have that

$$(\widehat{R}_{S,\ell}(y) \bmod p^\ell)/M \in (K(y)-1/2, K(y)+1/2).$$

The algorithm computes a representation of $(\widehat{R}_{S,\ell}(y))$ as a sum of monomials from the representations of $\{p_1,p_2,\ldots,p_m\}$ by directly applying the definitions of $\widehat{R}_{S,\ell}(y)$ and $\widehat{Q}_{S,\ell}(y,a)$. The degree of $(\widehat{R}_{S,\ell}(y))$ is at most $4k(p-1)n' + O(kp\log m)$. We will set $n' = \delta n$ in such a way that

$$M((1-\delta)n, 4k(p-1)\delta n) \le O^\star(p^{(n-n')/7}),$$

and that therefore the number of monomials in $(\widehat{R}_{S,\ell}(y))$ is $p^{(n-n')/7}\cdot m^{O(pq)}$. Hence we can obtain the value of $K(y)$ for all $y \in \mathbb{F}_p^n$ in time $p^{n-n'+o(n)}\cdot m^{O(pq)}$ by applying Lemma 2.3 to $\widehat{R}_{S,\ell}(y)$.

We now proceed to the running time analysis, for this we need to specify more precisely how the representation of $(\widehat{R}_{S,\ell}(y))$ as a sum of monomials is computed. In particular, all polynomial additions and multiplications are performed using the naive addition and multiplication algorithms, however, we make sure that whenever we are multiplying two polynomials, at least one of them has degree at most kq. Note that this is achievable, because all multiplications occur in the definition of $\widehat{Q}_{S,\ell}(y,a)$. Here a polynomial of degree at most k is taken to the $p-1$'th power, resulting in a polynomial of degree at most pq. Then F_ℓ is applied to this polynomial. Note that $F_\ell(t)$ can be computed using the definition of Lemma B.3 in such a way that in any multiplication, at least one of the two factors is either t or $1-t$. Thus, each multiplication takes time at most

$$O^\star(M(n, 4k(p-1)n' + O(kp\log m)))\cdot M(n,pq) \le M(n, 4k(p-1)n')\cdot (mn)^{O(pq)}.$$

Each $\widehat{Q}_{S,\ell}(y,a)$ is computed with $O^\star(|S|) \le O^\star(p^{3n'})$ polynomial additions and multiplications, and $(\widehat{R}_{S,\ell}(y))$ is the sum of $\widehat{Q}_{S,\ell}(y,a)$ over all $p^{n'}$ choices of a. Hence, the total number of operations needed to construct $(\widehat{R}_{S,\ell}(y))$ is upper bounded by $O^\star(p^{4n'})$. Evaluating $(\widehat{R}_{S,\ell}(y))$ using Lemma 2.3 takes time $p^{n-n'+o(n)}\cdot m^{O(pq)}$, while constructing $|S|$, which is done once, takes time $O^\star(p^{O(n')})$. Hence the total running time is upper bounded by

$$\begin{aligned} & p^{n-n'+o(n)} + p^{O(n')} + p^{4n'}\cdot M(n-n', 4k(p-1)n')\cdot (nm)^{O(qk)} \\ \le\ & p^{(1-\delta)n+o(n)} + p^{c\cdot\delta n} + p^{4\delta n}\cdot M((1-\delta)n, 4k(p-1)\delta n)\cdot (nm)^{O(pk)}. \end{aligned}$$

Here c is the constant in the big-Oh notation in the $p^{O(n')}$ term, c is independent of p and k.

We now discuss the choice of δ for different values of p and k. We will always pick δ to be at less than $1/(c+1)$ and less than $1/5$, thus the first term in the running time will always be larger than the second.

Further, we have already constrained the choice of δ such that $M((1-\delta)n, 4k(p-1)\delta n) \leq p^{(n-n')/7}$. For such a δ, the running time is upper bounded by

$$p^{(1-\delta)n+o(n)} + p^{4\delta n} \cdot p^{(1-\delta)/7} \cdot (nm)^{O(pk)} \leq p^{(1-\delta)n+o(n)}.$$

We first consider the case that $p \geq (2e)^7$. We set $\delta = 1/(10ekp^{6/7})$, then we have that

$$\begin{aligned} M((1-\delta)n, 4k(p-1)\delta n) &\leq M(1-\delta)n, 5kp\delta n(1-\delta)) \leq O^\star\left(\binom{n+5kp\delta n}{n}^{1-\delta}\right) \\ &\leq (e+5ekp\delta)^{(1-\delta)n} \leq p^{(1-\delta)n/7} = p^{(n-n')/7}. \end{aligned}$$

Suppose now that $p < (2e)^7$, we set $\delta = \frac{1}{kp^{6/7}\cdot 300}$. We have that

$$\begin{aligned} M((1-\delta)n, 4k(p-1)\delta n) &\leq O^\star\left(\binom{n+5kp\delta n}{n}^{1-\delta}\right) \leq O^\star\left(\binom{n+n\frac{p^{1/7}}{60}}{n}^{1-\delta}\right) \\ &\leq O^\star\left(\left(\left(1+\frac{p^{1/7}}{60}\right)\cdot\left(1+\frac{60}{p^{1/7}}\right)^{\frac{p^{1/7}}{60}}\right)^{(1-\delta)n}\right) \leq O^\star\left(\left(p^{1/7}\right)^{(1-\delta)n}\right). \end{aligned}$$

The last transition was verified by explicitely comparing the two sides for every integer p between 2 and $(2e)^7$. This completes the proof.

□

Lemma B.5 only works for prime fields, however, by combining Lemma B.5 with the reduction of Lemma 3.4, we obtain an algorithm that works for all fields. More concretely, we are now in position to prove Theorem 1.2.

Theorem 1.2 (restated). *Let $q = p^d$ for a prime p and integer $d \geq 1$. There is a deterministic algorithm that, given an instance of SysPolyEqs(q) with m polynomial equations of degree at most k in n variables, runs in time $q^{n\left(1-\frac{1}{300kq^{6/7d}}\right)+o(n)} \cdot m^{O(qk)}$ and counts the number of satisfying assignments for the system.*

Proof. Given as input a system of m degree d polynomial equations over n variables in $\mathbb{F}_q$, find in time $p^{O(d)} = q^{O(1)}$ an irreducible polynomial $P(X)$ in $\mathbb{F}_p[X]$ of degree d. This can be done by going over all of the at most p^d choices for the coefficients of $P(X)$, and then testing irreducibility by dividing $P(X)$ by each polynomial in $\mathbb{F}_p[X]$ of degree at most $d/2$. Then transform the input system using Lemma 3.4 to an equivalent system of km degree d polynomial equations over kn variables in $\mathbb{F}_p$, and use Lemma B.5 to solve this system. □

C Proofs Omitted from Section 4

Lemma 4.1 (restated) *Let $U_1, U_2, \ldots, U_t$ be subsets of $[n]$ and $L_1, L_2, \ldots, L_{t'}$ be linear forms such that $\{L_i\}_{i=1}^{t'}$ is linearly independent and the rank of $\{U_i\}_{i=1}^{t}$ relative to $\{L_i\}_{i=1}^{t'}$ is d. Then, for all $a_1, a_2, \ldots, a_{t'}, b_1, b_2, \ldots, b_t \in \{0,1\}$, there exists a polynomial p of degree at most d such that $p(x) = \prod_{k=1}^{t}\left(b_k + \sum_{l\in U_k} x_l\right)$ holds for all $x \in \{L_i = a_i\}_{i=1}^{t'}$.*

Proof. Let V be a linearly independent subset $\{S_1, S_2, \ldots, S_d\}$ of $\{U_i\}_{i=1}^{t}$ such that the rank of V relative to $\{L_i\}_{i=1}^{t'}$ is d. By the definition of the rank of a set relative to a set, each U_i can be written as a linear combination of vectors in $V \cup \{L_i\}_{i=1}^{t'}$. This implies that we can write $\prod_{k=1}^{t} \left(b_k + \sum_{l \in U_k} x_l\right)$ as a function of $S_1, S_2, \ldots, S_d, L_1, L_2, \ldots, L_{t'}$. By setting $L_1 = a_1, L_2 = a_2, \ldots, L_{t'} = a_{t'}$ and using the fact that every function in d variables can be written as a degree-d polynomial, we complete the proof. $\square$