Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2)

Itai Dinur

Department of Computer Science, Ben-Gurion University, Israel

Abstract. At SODA 2017 Lokshtanov et al. presented the first worst-case algorithms with exponential speedup over exhaustive search for solving polynomial equation systems of degree d in n variables over finite fields. These algorithms were based on the polynomial method in circuit complexity which is a technique for proving circuit lower bounds that has recently been applied in algorithm design. Subsequent works further improved the asymptotic complexity of polynomial method-based algorithms for solving equations over the field $\mathbb{F}_2$. However, the asymptotic complexity formulas of these algorithms hide significant low-order terms, and hence they outperform exhaustive search only for very large values of n.

In this paper, we devise a concretely efficient polynomial method-based algorithm for solving multivariate equation systems over $\mathbb{F}_2$. We analyze our algorithm's performance for solving random equation systems, and bound its complexity by about $n^2 \cdot 2^{0.815n}$ bit operations for $d = 2$ and $n^2 \cdot 2^{(1-1/2.7d)n}$ for any $d \geq 2$.

We apply our algorithm in cryptanalysis of recently proposed instances of the Picnic signature scheme (an alternate third-round candidate in NIST's post-quantum standardization project) that are based on the security of the LowMC block cipher. Consequently, we show that 2 out of 3 new instances do not achieve their claimed security level. As a secondary application, we also improve the best-known preimage attacks on several round-reduced variants of the Keccak hash function.

Our algorithm combines various techniques used in previous polynomial method-based algorithms with new optimizations, some of which exploit randomness assumptions about the system of equations. In its cryptanalytic application to Picnic, we demonstrate how to further optimize the algorithm for solving structured equation systems that are constructed from specific cryptosystems.

1 Introduction

The security of many cryptographic schemes is based on the conjectured hardness of solving systems of polynomial equations over a finite field. This problem is known to be NP-hard even for systems of quadratic equations over $\mathbb{F}_2$.

The input to the problem consists of m polynomials in n variables over a finite field $\mathbb{F}$, denoted by $E = \{P_j(x_1, \ldots, x_n)\}_{j=1}^{m}$, where each polynomial is given

as a sum of monomials. The algebraic degree of each polynomial is bounded by a small constant d. The goal is to find a solution of the system, namely $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n) \in \mathbb{F}^n$ such that $P_j(\hat{x}) = 0$ for every $j \in \{1, \ldots, m\}$, or to determine that a solution does not exist.[1]

In this paper, we will be interested in the concrete (rather than asymptotic) complexity of algorithms for solving polynomial systems over the field $\mathbb{F}_2$ and in their applications to cryptanalysis.

1.1 Previous Algorithms for Solving Polynomial Equation Systems

The problem of solving polynomial equation systems over finite fields is widely studied. We give a brief overview of the main algorithms that were applied in cryptanalysis.

Classical techniques developed to solve polynomial systems attempt to find a reduced representation of the ideal generated by the polynomials in the form of a Gröbner basis (e.g., the F4 [17] and F5 [18] algorithms). These methods have had success in solving some very structured polynomial systems that arise from certain cryptosystems (e.g., see [19]), but it is difficult to estimate their complexity in solving arbitrary systems. Related methods such as XL [11] and its variants typically work well only for largely over-defined systems in which $m \gg n$.

In [3] Bardet et al. analyzed the problem of solving quadratic equations over $\mathbb{F}_2$ and devised an algorithm that combines exhaustive search and sparse linear algebra. The authors estimated the asymptotic complexity of their randomized algorithm for $m = n$ by $O(2^{0.792n})$, under some algebraic assumptions that were empirically found to hold for random systems. However, due to a large overhead hidden in the asymptotic formula, the authors expect their algorithm to beat exhaustive search only when the number of variables is at least 200. Another algorithm based on a different hybrid approach was published by Joux and Vitse, who gave experimental evidence that it outperforms in practice previous algorithms for a wide range of parameters [22]. Analyzing the complexity of the algorithm is non-trivial, but according to the recent work of Duarte [16], the algorithm for solving quadratic systems over $\mathbb{F}_2$ with $m = n + 1$ does not beat existing algorithms (such as the one of [3]) asymptotically.

Finally, we mention the work of Bouillaguet et al. [8], which devised an optimized exhaustive search algorithm for solving polynomial systems of degree d over $\mathbb{F}_2$ whose complexity is $2d \log n \cdot 2^n$ bit operations.

1.2 The Polynomial Method

In [27] Lokshtanov et al. presented the first worst-case algorithms for solving polynomial equations over finite fields that have exponential speedup over exhaustive search. These algorithms were based on a technique known as the *polynomial method*. It was borrowed from circuit complexity [4] and recently applied

[1] We denote an assignment to the formal variable vector x in the polynomial $P_j(x)$ by $\hat{x}$ and the value of $P_j(x)$ on this assignment by $P_j(\hat{x})$.

in algorithm design (see [35] for a survey). The randomized algorithm of Lokshtanov et al. for solving equations over $\mathbb{F}_2$ has runtime of $O(2^{0.8765n})$ for quadratic systems and $O(2^{(1-1/(5d))n})$ in general. Following [27], Björklund, Kaski and Williams [7] reduced the complexity of these algorithms to $O(2^{0.804n})$ for $d = 2$ and $O(2^{(1-1/2.7d)n})$ in general. More recently, these complexities were further improved in [12] by Dinur to $O(2^{0.6943n})$ for $d = 2$ and $O(2^{(1-1/2d)n})$ for $d > 2$.

Although these recent algorithms have better asymptotic complexity than exhaustive search, a close examination reveals that their concrete (non-asymptotic) complexity is above 2^n for parameter ranges that are relevant to cryptography.

1.3 Our Results

We introduce the polynomial method for solving multivariate equation systems over $\mathbb{F}_2$ as a tool in cryptanalysis. For this purpose, we devise a concretely efficient algorithm for solving such systems.

Our algorithm is relatively simple and its analysis assumes the degree d polynomials are selected uniformly at random. Up to small constants, we bound the complexity of our algorithm by $n^2 \cdot 2^{0.815n}$ bit operations for $d = 2$, and $n^2 \cdot 2^{(1-1/2.7d)n}$ for $d \geq 2$. The analysis we present here is heuristic, but is it formally established in the full version of this paper.

In a straightforward implementation of our algorithm, its memory complexity is significant and only about n times lower than its time complexity. In fact, this is the case for all previous polynomial method-based algorithms. We address this issue by presenting a memory-optimized variant of the algorithm which maintains roughly the same time complexity, but whose memory complexity is reduced to about $n^2 \cdot 2^{0.63n}$ bits for $d = 2$ and $n^2 \cdot 2^{(1-1/1.35d)n}$ in general.[2]

Potential fast implementation. Even after the substantial reduction in memory complexity, it remains high and would present a challenge for obtaining a fast practical implementation of the algorithm. To address this challenge, future works may present an additional reduction in memory complexity or utilize time-memory tradeoffs. Taking this optimistic viewpoint, our work may be viewed as a step towards a practically efficient implementation of a polynomial method-based algorithm for solving multivariate equation systems over $\mathbb{F}_2$.

We stress, however, that the main goal of the paper is to give a good *analytical* concrete estimate of the complexity of polynomial method algorithms for problem sizes that are too large to be solved in practice. Consequently, it can be used in the security analysis of cryptosystems and serve as a starting point for additional optimizations.

Asymptotic complexity. Our algorithm can actually be viewed as a concretely efficient variant of the algorithm of [12]. The only reason that [12] seems to

[2] Asymptotically, the polynomial factor in the memory complexity formula is between n^2 and n^3, but it is close to n^2 for relevant parameters.

have better asymptotic complexity is that it uses a self-reduction to a smaller multivariate system. Essentially, each recursive call reduces the exponent in the complexity formula, where the gain diminishes with the number of recursive calls. On the other hand, each such call increases the lower order terms.

An estimated calculation suggests that a self-reduction is profitable for $d = 2$ starting from about $n = 100$. Beyond $n = 200$ for $d = 2$ the advantage in concrete complexity is made substantial using several recursive calls. On the other hand, for $d = 4$ the reduction seems profitable only beyond $n = 200$. We chose not to augment our algorithm with any self-reduction and simply replace it with exhaustive search for two reasons. First, as described below, the applications we present in this paper require solving multivariate systems with $d > 2$, for which the benefit of the self-reduction seems marginal for relevant parameters. Second, this self-reduction significantly complicates the concrete analysis, whereas we aim for simplicity. Yet, this estimation suggests that the full potential of the algorithm is still to be discovered.

Cryptanalytic applications. We estimate the concrete complexities of our algorithm for solving quadratic systems in $80, 128$ and 256 variables by $2^{77}, 2^{117}$ and 2^{223} bit operations, respectively. In terms of cryptanalysis, the main targets of our algorithm for $d = 2$ are multivariate public-key cryptosystems (e.g., HFE by Patarin [30] and UOV by Kipnis, Patarin and Goubin [24]), whose security is directly based on the hardness of solving quadratic systems. However, recent multivariate cryptosystems such as GeMSS [9] (an alternate third-round candidate signature scheme in NIST's post-quantum standardization project [29]) were designed with a sufficiently large security margin and resist our attack. Nevertheless, the security margin for some of these cryptosystems seems to be reduced by our algorithm.

Interestingly, the main application of our algorithm is for solving multivariate systems of degree $d > 2$ which have generally received less attention in the literature compared to quadratic systems. In particular, we apply it to cryptanalyze recently proposed instances [23] of the Picnic signature scheme [10] (an alternate third-round candidate in NIST's post-quantum standardization project) that is based on the security of the LowMC block cipher [1].

We focus on the three Picnic instances where the LowMC block cipher has a full Sbox layer and 4 internal rounds. These instances have claimed security levels of $S \in \{128, 192, 255\}$ bits. The best-known attacks on these instances were recently published by Banik et al. [2], but they are only applicable to weakened variants where the number of LowMC rounds is reduced from 4 to 2. On the other hand, our attacks on the full 4-round instances with $S \in \{128, 192, 255\}$ have complexities of $2^{130}, 2^{188}$ and 2^{245} bit operations, respectively. Thus, 2 out of the 3 instances do not achieve their claimed security level, while the security of the instance with $S = 128$ is somewhat marginal. When optimized for time complexity, the attacks for $S \in \{128, 192, 255\}$ require about 2^{112}, 2^{164} and 2^{219} bits of memory, respectively. However, there is no consensus among researchers

on a model that takes memory complexity into account and the formal security claims of the Picnic (and LowMC) designers only involve time complexity.[3]

The authors of [23] also proposed conservative instantiations where the number of LowMC rounds is increased by 1. The attack complexities for these instances with $S \in \{128, 192, 255\}$ bits become $2^{133}, 2^{192}$ and 2^{251}, respectively. Hence the instance with $S = 255$ still does not achieve the desired security level, while security remains very marginal for the strengthened instance with $S = 192$.

We also analyze round-reduced variants of the Keccak hash function [5], which was selected by NIST in 2015 as the SHA-3 standard. In particular, we describe the best-known preimage attacks on 4 rounds of Keccak-k for all $k \in \{224, 256, 384, 512\}$. These have complexities of $2^{217}, 2^{246}, 2^{374}$ and 2^{502} bit operations, respectively. We further describe the first collision attack on 4-round Keccak-512 that is (slightly) faster than the birthday bound. We consider the cryptanalysis of round-reduced Keccak as a secondary application since our attacks are very far from threatening Keccak's security.

Complexity evaluation. It is important to emphasize that the complexities of our attacks are measured in *bit operations*. On the other hand, the complexity of exhaustive search for the cryptanalytic problems we consider on a space of size 2^n is larger than 2^n bit operations. Hence the improvement we obtain over exhaustive search is more significant than it may first appear.

In particular, the encryption algorithms of the LowMC instances we cryptanalyze (that are used in Picnic) have complexities of at least 2^{17} bit operations. However, evaluating an attack in terms of the complexity of the LowMC encryption algorithm is misleading, as naive exhaustive search is not the most efficient generic attack on the 4-round LowMC instances. Indeed, breaking these instances is easily reduced to solving a multivariate system in (about) n variables with $d = 4$, for which the best-known generic attack is the optimized exhaustive search algorithm of Bouillaguet et al. [8], whose complexity is about $2d \log n \cdot 2^n = 8 \log n \cdot 2^n$ bit operations (also see [15] for an alternative algorithm). Overall, in terms of bit operations, our algorithm is more efficient than the one of [8] by a factor which is between 32 and $2^{16} = 65536$ (depending on the LowMC instance considered).

1.4 Comparison to Previous Works

The analysis of our algorithm for random equation systems over $\mathbb{F}_2$ is simple. In contrast, the analysis of previous cryptanalytic algorithms that beat brute force for random equation systems over $\mathbb{F}_2$ (particularly, the one by Bardet et al. [3]) is based on heuristic algebraic assumptions that are difficult to analyze.

The algorithm of [3] (applied to systems with $m = n$) may seem to have a slightly better asymptotic complexity than ours, but this is a misleading comparison, since (as noted above) our algorithm can be extended to have better

[3] The Picnic designers have confirmed our findings and plan to update the parameter sets accordingly.

asymptotic complexity. In terms of concrete complexity for relevant parameters, [3] only beats exhaustive search for $d = 2$ beyond $n = 200$, whereas the complexity of our algorithm for $d = 2, n = 200$ is about 2^{177}.

The algorithm of Joux and Vitse [22] performs well in practice. However, its concrete complexity has not been established analytically and it is not clear how to use it in the security analysis of cryptosystems.

Finally, previous polynomial method-based algorithms were only analyzed asymptotically. While it is difficult to calculate their exact concrete complexity, our optimizations reduce complexity by many orders of magnitude for relevant parameters.

1.5 Technical Contribution

Let $E = \{P_j(x_1, \ldots, x_n)\}_{j=1}^m$ be a polynomial system of degree d over $\mathbb{F}_2$. Algorithms based on the polynomial method consider the polynomial

$$F(x) = (1 + P_1(x))(1 + P_2(x)) \ldots (1 + P_m(x))$$

(operations are over $\mathbb{F}_2$). Note that $F(\hat{x}) = 1$ if and only if $\hat{x}$ is a solution of E. However, the degree of $F(x)$ can be as high as $d \cdot m$ and it generally contains too many monomials to manipulate efficiently. It is thus replaced by a *probabilistic polynomial* $\tilde{F}(x)$ with a lower degree that agrees with $F(x)$ on most assignments. Taking advantage of the low degree of $\tilde{F}(x)$ by using fast polynomial interpolation and evaluation algorithms allows to solve E faster than brute force.

Our main algorithm includes various concrete optimizations and simplifications to previous polynomial method-based algorithms. These are described in detail in Section 3. For example, we reduce the number of polynomials which need to be interpolated and evaluated and show how to jointly interpolate several polynomials with improved amortized complexity.

Then, we show how to reduce the memory complexity of the algorithm by an exponential factor with essentially no penalty in time complexity. The optimization is based on a memory-reduced variant of the Möbius transform over $\mathbb{F}_2$ which is a fast polynomial interpolation and evaluation algorithm. This variant allows to evaluate a low-degree polynomial on its entire domain with memory complexity proportional to the memory required to store the input polynomial itself (and time complexity proportional to the domain size). Although the Möbius transform is widely used and the variant we describe is simple, it seems not to be well-known. The way this variant is used in our algorithm is, however, slightly more involved.

As an additional technical contribution, we show how to optimize our algorithm for solving *structured* equation systems that are constructed from specific cryptosystems. In particular, we observe that in cryptographic settings, a probabilistic polynomial can be replaced with a deterministic construction of a polynomial that preserves the structure of the polynomials of E. We show that in some cases (e.g., in the analysis of Picnic in Section 5.2) this alternative polynomial has reduced degree, optimizing the attack. We view this optimization as

one of the main contributions of this paper, as it may serve as a starting point for future works in cryptanalysis.

Paper structure. Next, we describe some preliminaries. We overview our algorithm in Section 3 and give its details in Section 4. Applications are described in Section 5.

2 Preliminaries

2.1 Boolean Algebra

Given a finite ordered set S, denote by $|S|$ its size and by $S[i]$ its i'th element. For a positive integer n, let $[n] = \{1, 2, \ldots, n\}$.

It is well-known that any Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely described as a multilinear polynomial, whose algebraic normal form (ANF) is given by $F(x_1, \ldots, x_n) = \sum_{u \in \{0,1\}^n} \alpha_u(F) M_u(x)$, where $\alpha_u(F) \in \{0, 1\}$ is the coefficient of the monomial $M_u(x) = \prod_{i=1}^n x_i^{u_i}$ (operations are over $\mathbb{F}_2$).

We denote by $\mathrm{HW}(x)$ the Hamming weight of a vector $x \in \{0,1\}^n$. The algebraic degree of a function F is defined as $\max\{\mathrm{HW}(u) \mid \alpha_u(F) \neq 0\}$. Let W_w^n be the set $\{x \in \{0,1\}^n \mid \mathrm{HW}(x) \leq w\}$. Thus, a function F of degree $d \leq n$ can be described using $|W_d^n| = \sum_{i=0}^d \binom{n}{i}$ coefficients. We simplify notation by denoting $\binom{n}{\downarrow w} = \sum_{i=0}^w \binom{n}{i}$.

For $i \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$, define the function $F_{x_i \leftarrow b} : \mathbb{F}_2^{n-1} \to \mathbb{F}_2$ by

$$F_{x_i \leftarrow b}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = F(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n).$$

Interpolation. Any ANF coefficient $\alpha_u(F)$ can be interpolated by summing (over $\mathbb{F}_2$) over $2^{\mathrm{HW}(u)}$ evaluations of F: for $u \in \{0,1\}^n$, define $I_u = \{i \in \{1, \ldots, n\} \mid u_i = 1\}$ and let $S_u = \{x \in \{0,1\}^n \mid I_x \subseteq I_u\}$. Then,

$$\alpha_u(F) = \sum_{\hat{x} \in S_u} F(\hat{x}). \tag{1}$$

Indeed, among all monomials only $M_u(\hat{x})$ attains a value of 1 an odd number of times in the expression $\sum_{\hat{x} \in S_u} F(\hat{x}) = \sum_{\hat{x} \in S_u} \sum_{v \in \{0,1\}^n} \alpha_v(F) M_v(\hat{x})$.

Proposition 2.1. *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. For some $1 \leq n_1 \leq n$, partition its n variables into two sets $y_1, \ldots, y_{n-n_1}, z_1, \ldots, z_{n_1}$. Given the* ANF *of F, write it as $F(y, z) = (z_1 \ldots z_{n_1}) F_1(y) + F_2(y, z)$ by factoring out all the monomials that are multiplied with $z_1 \ldots z_{n_1}$. Then, $F_1(y) = \sum_{\hat{z} \in \{0,1\}^{n_1}} F(y, \hat{z})$.*

The proposition follows from (1) by considering the polynomial $F_1(y)$ as the symbolic coefficient of the monomial $z_1 \ldots z_{n_1}$. Observe that if $F(y, z)$ is of degree d then $F_1(y)$ is of degree at most $\max(d - n_1, 0)$.

Remark 2.1. Proposition 2.1 is also at the basis of cube attacks [14]. However, in cube attacks, the z variables are public bits controlled by the attacker (e.g., plaintext bits) while the y variables are secret key bits. In our case, we will apply Proposition 2.1 in a setting where all variables are secret key bits.

2.2 Model of Computation

We estimate the complexity of a straight-line implementation of our algorithm by counting the number of bit operations (e.g., AND, OR, XOR) on pairs of bits. This ignores bookkeeping operations such as moving a bit from one position to another (which merely requires renaming of variables in straight-line programs).

2.3 Basic Algorithms

We describe the basic algorithms that our main algorithm uses as sub-procedures.

Möbius transform. Given the truth table of an arbitrary function F (as a bit vector of 2^n entries), the ANF of F can be represented as a bit vector of 2^n entries, corresponding to its 2^n coefficients. This ANF representation can be computed from the truth table of F via the *Möbius transform* over $\mathbb{F}_2^n$.

A fast algorithm for computing this transform is based on the decomposition

$$F(x_1, \ldots, x_n) = x_1 \cdot F_1(x_2, \ldots, x_n) + F_2(x_2, \ldots, x_n). \tag{2}$$

Thus, one recursively computes the ANF of $F_1(x_2, \ldots, x_n)$ and $F_2(x_2, \ldots, x_n)$. Given the evaluations of F, for every $(\hat{x}_2, \ldots, \hat{x}_n) \in \{0,1\}^{n-1}$,

$$F_2(\hat{x}_2, \ldots, \hat{x}_n) = F(0, \hat{x}_2, \ldots, \hat{x}_n)$$

and

$$F_1(\hat{x}_2, \ldots, \hat{x}_n) = F(0, \hat{x}_2, \ldots, \hat{x}_n) + F(1, \hat{x}_2, \ldots, \hat{x}_n).$$

Therefore, computing the evaluations of F_1 requires 2^{n-1} bit operations. Denoting the time complexity by $T(n)$, we have $T(n) = 2T(n-1) + 2^{n-1}$, and hence $T(n) \leq n \cdot 2^{n-1} < n \cdot 2^n$.

By (1), a function F of degree bounded by $d \leq n$ can be interpolated from its evaluations on the set W_d^n. Adapting the Möbius transform for such a function F using the decomposition above gives an algorithm with complexity $T(n,d) \leq T(n-1,d) + T(n-1,d-1) + \binom{n-1}{\downarrow d}$ and $T(n,n) \leq n \cdot 2^n$. It can be shown by induction that $T(n,d) \leq n \cdot \binom{n}{\downarrow d}$ bit operations.

The Möbius transform over $\mathbb{F}_2^n$ coincides with its inverse which corresponds to evaluating the ANF representation of F (i.e., computing its truth table). More details about the Möbius transform over $\mathbb{F}_2^n$ and its applications in crypography can be found in [21, p.285].

Memory complexity. A standard in-place implementation of the Möbius transform performs n iterations on its input vector, where in each iteration, half of the array entries are XORed to the other half. This requires 2^n bits of memory.

Fast exhaustive search for polynomial systems over $\mathbb{F}_2$ [8]. At CHES 2010 Bouillaguet et al. presented an optimized exhaustive search algorithm for enumerating over all solutions of polynomials system over $\mathbb{F}_2$. For a polynomial system of degree d with n variables, the complexity of their algorithm is $2d \cdot \log n \cdot 2^n$. The algorithm also requires a preprocessing phase that has complexity of n^{2d}, which is negligible when d is much smaller than n. We note that the analysis of the algorithm makes some randomness assumptions about the polynomial system, and requires that the expected number of solutions to a system with m equation over n variables is about 2^{n-m}.

In this paper we will use this algorithm to find solutions insides sets of the special form $W_w^{n-n_1} \times \{0,1\}^{n_1}$ for some values of w and n_1 in time

$$2d \cdot \log n \cdot |W_w^{n-n_1} \times \{0,1\}^{n_1}| = 2d \cdot \log n \cdot 2^{n_1} \cdot \binom{n-n_1}{\downarrow w}.$$

For an arbitrary value of n_1, obtaining this complexity may not be trivial because the algorithm of [8] iterates over the search space using a Gray code, and hence it cannot be implemented on the set of low Hamming weight vectors $W_w^{n-n_1}$ (for $w < n - n_1$) in a straightforward manner.

On the other hand, the least significant bits of the vectors in the set $W_w^{n-n_1} \times \{0,1\}^{n_1}$ can be traversed using a standard Gray code, paying a penalty only every 2^{n_1} iterations. Since we can naively enumerate any set of n-bit vectors by flipping at most n bits at a time, we can conservatively estimate the multiplicative penalty by about n. Thus, the amortized penalty over [8] is about $2^{-n_1} \cdot n$. In our setting, $2^{n_1} \gg n$, and the overhead is negligible.

Remark 2.2. Asymptotically, we will set $n_1 = \Theta(n)$, hence $2^{n_1} = \omega(n)$. Concretely, the advantage of the algorithm over exhaustive search will be roughly $\frac{2^{n_1}}{n^2}$. Thus, we may assume that $2^{n_1} \gg n$ holds without loss of generality, as otherwise, the algorithm would not obtain any advantage over exhaustive search.

2.4 Probabilistic Polynomials

Previous algorithms for solving multivariate equation systems based on the polynomial method (starting with [27]) make use of probabilistic polynomials. In particular, these works use the following construction (credited to Razborov [31] and Smolensky [32]). Given m polynomial equations of degree d in the n Boolean variables $x_1, \ldots, x_n$, $E = \{P_j(x)\}_{j=1}^m$, consider the polynomial $F(x) = (1 + P_1(x))(1 + P_2(x)) \ldots (1 + P_m(x))$. Note that $\hat{x}$ is a solution of E if and only if $F(\hat{x}) = 1$. Thus, we call the polynomial F the *identifying polynomial* of E.

The degree of $F(x)$ is generally too high and we work with a probabilistic polynomial with a lower degree, defined as follows. Let $\ell < m$ be a parameter. Pick a uniformly random matrix of full rank ℓ, $A \in \mathbb{F}_2^{\ell \times m}$ and define ℓ degree d polynomials as

$$R_i(x) = \sum_{j=1}^{m} A_{i,j} \cdot P_j(x). \tag{3}$$

We note that previous works [7,12,27] did not restrict the rank of A. In our case, this restriction will slightly simplify the analysis. Let

$$\tilde{F}(x) = (1 + R_1(x))(1 + R_2(x)) \dots (1 + R_\ell(x)) \tag{4}$$

be the identifying polynomial of the system $\tilde{E} = \{R_i(x)\}_{i=1}^{\ell}$. Note that the degree of $\tilde{F}(x)$ (denoted by $d_{\tilde{F}}$) is at most $d \cdot \ell$.

Proposition 2.2. *For any $\hat{x} \in \{0,1\}^n$, if $F(\hat{x}) = 1$, then $\tilde{F}(\hat{x}) = 1$. Otherwise, $F(\hat{x}) = 0$ and then $\Pr[\tilde{F}(\hat{x}) = 0] \geq 1 - 2^{-\ell}$.*

Proof. If $F(\hat{x}) = 1$, then $P_j(\hat{x}) = 0$ for all $j \in [m]$ and therefore $R_i(\hat{x}) = 0$ for all $i \in [\ell]$. Hence, $\tilde{F}(\hat{x}) = 1$.

Otherwise, $F(\hat{x}) = 0$. Let $v \in \mathbb{F}_2^m$ be a vector such that $v_j = P_j(\hat{x})$ and $u \in \mathbb{F}_2^\ell$, a vector such that $u_i = R_i(\hat{x})$. Note that $u = A \cdot v$. Since $F(\hat{x}) = 0$, there exists $j \in [m]$ such that $P_j(\hat{x}) = 1$ and thus $v \neq 0$. On the other hand, if $\tilde{F}(\hat{x}) = 1$, then $R_i(\hat{x}) = 0$ for all $i \in [\ell]$, implying that $u = 0$. Therefore, v is a non-zero vector in the kernel of A. Since A is a uniform matrix of full rank ℓ, any fixed non-zero vector (including v) belongs its kernel with probability $2^{-\ell} - 2^{-m} < 2^{-\ell}$. ■

2.5 Previous Polynomial Method Algorithms for Solving Equation Systems over $\mathbb{F}_2$

In this section we give a short description the previous polynomial method-based algorithms of [7,12]. We focus of the parts which are most relevant to this work.

The Björklund et al. algorithm [7]. In the algorithm of [7], the search problem of finding a solution to the system E is reduced to the parity-counting problem, where the goal is to compute the parity of the number of solutions.

The first step reduces the search problem to the problem of deciding whether a solution exists. The reduction iteratively fixes one variable of the solution at a time using $\Theta(n)$ calls to the decision algorithm. Then, the decision problem is reduced to the parity-counting problem. This reduction uses the Valiant-Vazirani affine hashing [34], adding random linear equations to the system with the goal of *isolating* some solution of E (assuming a solution exists), such that it is the only solution to the new system. In this case, the output of the parity-counting algorithm is 1. The number of linear equations to add that ensures isolation with high probability depends on the logarithm of the number of solutions to E, which is unknown. Hence, the algorithm exhausts all its possible n values.

We now overview the Björklund et al. parity-counting algorithm. Algebraically, the parity of solutions of $E = \{P_j(x)\}_{j=1}^m$ is computed by $\sum_{\hat{x} \in \{0,1\}^n} F(\hat{x})$, where $F(x) = (1+P_1(x)) \dots (1+P_m(x))$. This sum (parity) is computed in parts by partitioning the n variables into 2 sets $y = y_1, \dots, y_{n-n_1}$ and $z = z_1, \dots z_{n_1}$, where $n_1 < n$ is a parameter. Thus, $\sum_{\hat{x} \in \{0,1\}^n} F(\hat{x}) = \sum_{\hat{y} \in \{0,1\}^{n-n_1}} \sum_{\hat{z} \in \{0,1\}^{n_1}} F(\hat{y}, \hat{z})$.

Replace the polynomial $F(y,z)$ with the probabilistic polynomial $\tilde{F}(y,z) = (1+R_1(y,z))\dots(1+R_\ell(y,z))$ for $\ell = n_2+2$, similarly to (4). By Proposition 2.2 and a union bound over all $\hat{z} \in \{0,1\}^{n_1}$, for each $\hat{y} \in \{0,1\}^{n-n_1}$, $\Pr\left[\sum_{\hat{z}\in\{0,1\}^{n_1}} \tilde{F}(\hat{y},\hat{z}) = \sum_{\hat{z}\in\{0,1\}^{n_1}} F(\hat{y},\hat{z})\right] \geq 1 - 2^{n_1-\ell} = \frac{3}{4}$. In order to compute the sum for each $\hat{y} \in \{0,1\}^{n-n_1}$ efficiently, let $G(y) = \sum_{\hat{z}\in\{0,1\}^{n_1}} \tilde{F}(y,\hat{z})$. It follows from Proposition 2.1 that the degree of $G(y)$ is at most $d_G = d\cdot\ell - n_1$. Proceed by interpolating $G(y)$ by first computing its values on the set $W_{d_G}^{n-n_1}$. For this purpose, find all solutions of the equation system $\{R_i(y,z)\}_{i=1}^{\ell}$ in $W_{d_G}^{n-n_1} \times \{0,1\}^{n_1}$ via brute force, giving $\tilde{F}(\hat{y},\hat{z})$ for $(\hat{y},\hat{z}) \in W_{d_G}^{n-n_1} \times \{0,1\}^{n_1}$. Then, compute $G(\hat{y}) = \sum_{\hat{z}\in\{0,1\}^{n_1}} \tilde{F}(\hat{y},\hat{z})$ for $\hat{y} \in W_{d_G}^{n-n_1}$. Given these values of $G(y)$, interpolate it using the Möbius transform.

Next, evaluate $G(y)$ on all $\hat{y} \in \{0,1\}^{n-n_1}$ (using the Möbius transform) to obtain the partial parity of each part $\sum_{\hat{z}\in\{0,1\}^{n_1}} \tilde{F}(\hat{y},\hat{z})$. However, each partial parity is correct only with probability $\frac{3}{4}$. Thus, repeat the above steps with $t = 48n+1$ independent probabilistic polynomials. Obtain t suggestions for each part and take their majority to obtain the true partial parity, except with exponentially small probability. Assuming that all partial parities are computed correctly, return $\sum_{\hat{x}\in\{0,1\}^n} F(\hat{x}) = \sum_{\hat{y}\in\{0,1\}^{n-n_1}} \sum_{\hat{z}\in\{0,1\}^{n_1}} F(\hat{y},\hat{z})$.

Optimizing complexity by self-reduction. Ignoring low-order (but concretely substantial) terms, the complexity of the above algorithm is dominated by brute force on sets $W_{d_G}^{n-n_1} \times \{0,1\}^{n_1}$ and polynomial evaluations on the sets $\{0,1\}^{n-n_1}$. The complexity is thus

$$O^*\left(\tbinom{n-n_1}{\downarrow d_G}\cdot 2^{n_1} + 2^{n-n_1}\right) \tag{5}$$

(O^* hides polynomial factors in n). The parameter n_1 can be set to balance these terms and optimize complexity.

In [7], the asymptotic complexity is optimized. Instead of brute force, for each $\hat{y} \in W_{d_G}^{n-n_1}$, compute $G(\hat{y})$ by a self-reduction to a parity-counting problem with input $\{R_i(\hat{y},z)\}_{i=1}^{\ell}$, which is a system with n_1 variables $z_1,\dots,z_{n_1}$.

Dinur's algorithm for enumerating solutions [12]. The main result of the followup paper [12] is an asymptotically more efficient algorithm. Essentially, it defines a multiple parity-counting problem (which computes many parities at once), and shows that all the $\binom{n-n_1}{\downarrow d_G}$ parities returned by recursive calls in [7] can be computed more efficiently by a single recursive call to a multiple parity-counting problem. As noted in Section 1.3, we do not use this reduction.

Instead, we focus on the secondary result of [12], which is an algorithm for enumerating *all* solutions of a polynomial system. In our context, we will use a related technique to eliminate the initial reduction of [7] from solving E to parity-counting (which has a high concrete overhead) and replace it with a more direct way of recovering solutions from parity computations.

Isolating solutions. The first observation is that we can isolate many solutions of E at once using a variable partition $x = (y, z) = (y_1, \ldots, y_{n-n_1}, z_1, \ldots, z_{n_1})$.

Definition 2.1 (Isolated solutions). *A solution $\hat{x} = (\hat{y}, \hat{z})$ to $E = \{P_j(y, z)\}_{j=1}^m$ is called isolated (with respect to the variable partition (y, z)), if for any $\hat{z}' \neq \hat{z}$, $(\hat{y}, \hat{z}')$ is not to a solution to E.*

The goal is to "evenly spread" solutions across the different $\hat{y}$ values. Thus, for many solutions $(\hat{y}, \hat{z})$, there is no additional solution that shares the same $\hat{y}$ value (namely, $(\hat{y}, \hat{z})$ is isolated). This is made possible by a random linear change of variables applied to the polynomials of E and a careful choice of n_1.

Enumerating isolated solutions. The second observation is that *all* solutions isolated by the variable partition (y, z) can be recovered bit-by-bit by computing $n_1 + 1$ sums (parities) for each $y \in \{0,1\}^{n-n_1}$. Let

$$V_0(y) = \sum_{\hat{z} \in \{0,1\}^{n_1}} F(y, \hat{z}), \text{and let } V_i(y) = \sum_{\hat{z} \in \{0,1\}^{n_1 - 1}} F_{z_i \leftarrow 0}(y, \hat{z})$$

for $i \in \{1, \ldots, n_1\}$, where $F(y, z) = (1 + P_1(y, z)) \ldots (1 + P_m(y, z))$.

Proposition 2.3. *Assume that $(\hat{y}, \hat{z})$ is an isolated solution of E with respect to (y, z). Then, $V_0(\hat{y}) = 1$ and $V_i(\hat{y}) = \hat{z}_i + 1$ for all $i \in \{1, \ldots, n_1\}$.*

As a result, in order to recover $(\hat{y}, \hat{z})$ (assuming $V_0(\hat{y}) = 1$), it is sufficient to compute $V_i(\hat{y})$ for each $i \in \{1, \ldots, n_1\}$. The formal polynomials $V_i(y)$ cannot be directly interpolated (they are derived from F and hence of high-degree). In [12], these sums are computed using its first algorithm.

Proof. First, since F is the identifying polynomial of E, then $V_0(\hat{y})$ counts the parity of solutions of the system $\{P_j(\hat{y}, z)\}_{i=1}^m$ (in which y is fixed to $\hat{y}$). Therefore, if $(\hat{y}, \hat{z})$ is an isolated solution of E with respect to (y, z), then $V_0(\hat{y}) = 1$.

Next, if $\hat{z}_i = 0$, then the assignment $(\hat{y}, \hat{z})$ continues to be an isolated solution of E with respect to (y, z) after setting $z_i = 0$ and hence $V_i(\hat{y}) = 1$. Otherwise, $\hat{z}_i = 1$, and E has no solutions after setting $y = \hat{y}$ and $z_i = 0$, implying that $V_i(\hat{y}) = 0$. In both cases, $V_i(\hat{y}) = \hat{z}_i + 1$ for all $i \in \{1, \ldots, n_1\}$ as required. ■

Testing solutions. For each $\hat{y} \in \{0,1\}^{n-n_1}$, the algorithm computes $n_1 + 1$ sums. Those with $V_0(\hat{y}) = 1$ suggest some solution $(\hat{y}, \hat{z})$. The suggestion is correct if $(\hat{y}, \hat{z})$ is an isolated solution. Otherwise, the suggestion may be a "false alarm". Consequently, all suggested solutions are tested on E.

3 Overview of the New Algorithm

The starting point of our algorithm is the solution enumeration algorithm of [12]. We first notice that in order to find a solution to E, it is, in fact, sufficient to enumerate isolated solutions of $\tilde{E}$, (as they form a superset of the solution set of E) and test each one on E. This has a significant advantage in terms of concrete complexity, as detailed below.

We now describe how we isolate solutions of $\tilde{E}$ with respect to a variable partition $x = (y, z)$ according to a parameter n_1 and then overview our algorithm that outputs all isolated solutions of $\tilde{E}$ and tests them.

Isolating solutions. We use the following proposition.

Proposition 3.1. *Let E and $\tilde{E}$ be polynomials systems with identifying polynomials $F(x)$ and $\tilde{F}(x)$, respectively. For $n_1 = \ell - 1$, define a variable partition $x = (y, z) = (y_1, \ldots, y_{n-n_1}, z_1, \ldots, z_{n_1})$. Assume that $(\hat{y}, \hat{z})$ is an isolated solution of E. Then, $\Pr[(\hat{y}, \hat{z})$ is an isolated solution of $\tilde{E}] \geq 1 - 2^{n_1-\ell} = \frac{1}{2}$.*

Proof. The proposition follows by Proposition 2.2 by a union bound over the set $\{(\hat{y}, \hat{z}') \mid \hat{z}' \neq \hat{z}\}$, whose size is $2^{n_1} - 1$. ■

Hence, assuming that E has an isolated solution, setting $\ell = n_1 + 1$ ensures that this solution is also isolated in $\tilde{E}$ with probability at least $\frac{1}{2}$. Consequently, the algorithm has to be repeated a few times (with independent probabilistic polynomials) until it is output.

We also need to argue that E has an isolated solution with high probability. The (y, z) variable partition groups a solution to E together with $2^{n_1} - 1$ different assignments. In a cryptographic setting, we may assume that each such assignment satisfies E (containing m equations) with probability 2^{-m}. Thus, a solution to E is isolated with probability at least $1 - 2^{n_1-m}$, which is typically very close to 1, as in our case we set $n_1 < n/5$ (to optimize complexity) and we usually have $m \gg n/5$.

Enumerating isolated solutions. Similarly to [12], isolated solutions are recovered bit-by-bit by computing n_1 sums, but we enumerate isolated solutions of $\tilde{E}$ rather than E. Define the polynomials

$$U_0(y) = \sum_{\hat{z} \in \{0,1\}^{n_1}} \tilde{F}(y, \hat{z}), \text{and } U_i(y) = \sum_{\hat{z} \in \{0,1\}^{n_1-1}} \tilde{F}_{z_i \leftarrow 0}(y, \hat{z})$$

for $i \in \{1, \ldots, n_1\}$.

Proposition 3.2. *Assume that $(\hat{y}, \hat{z})$ is an isolated solution of $\tilde{E}$ with respect to (y, z). Then, $U_0(\hat{y}) = 1$ and $U_i(\hat{y}) = \hat{z}_i + 1$ for all $i \in \{1, \ldots, n_1\}$.*

Proof. The proposition follows from Proposition 2.3, applied with $\tilde{E}$ and $\tilde{F}$, instead of E and F. ■

Exploiting the low degree of $\tilde{F}$, our algorithm interpolates all $n_1 + 1$ polynomials $U_i(y)$ for $i \in \{0, \ldots, n_1\}$ and then evaluates each one on all $\hat{y} \in \{0, 1\}^{n-n_1}$ to recover isolated solutions. [4] We optimize the interpolation of the polynomials $U_i(y)$, exploiting the following proposition.

[4] We never explicitly interpolate the probabilistic polynomial $\tilde{F}$ itself, but only the polynomials $U_i(y)$ derived from it.

Proposition 3.3. *Let $\tilde{E} = \{R_i(y,z)\}_{i=1}^{\ell}$ and let $\tilde{F}$ by its identifying polynomial. Denote by $d_{\tilde{F}}$ the degree of $\tilde{F}$ and let $w = d_{\tilde{F}} - n_1$. The polynomial $U_0(y)$ can be interpolated from the solutions of $\tilde{E}$ on the set $W_w^{n-n_1} \times \{0,1\}^{n_1}$, while $U_i(y)$ for $i \in \{1,\ldots,n_1\}$ can be interpolated from the solutions of $\tilde{E}$ on the set $W_{w+1}^{n-n_1} \times \{0,1\}^{i-1} \times \{0\} \times \{0,1\}^{n_1-i}$. Hence, all the $n+1$ polynomials can be interpolated from the solutions of $\tilde{E}$ on the set $W_{w+1}^{n-n_1} \times \{0,1\}^{n_1}$.*

Proposition 3.3 shows that the domains of the system $\tilde{E}$ solved for interpolating $U_i(y)$ for $i \in \{1,\ldots,n_1\}$ overlap. Instead of naively solving $\tilde{E}$ on $n+1$ overlapping domains, we solve $\tilde{E}$ on one (slightly bigger) domain. Specifically, we use the exhaustive search algorithm of [8] for this purpose. We note that the analysis of [8] requires randomness assumptions about the input system, yet the other optimizations described here for enumerating isolation solutions do not.

Proof. By Proposition 2.1, the algebraic degree of $U_0(y) = \sum_{\hat{z} \in \{0,1\}^{n_1}} \tilde{F}(y,\hat{z})$ is at most $d_{\tilde{F}} - n_1 = w$, while the algebraic degree of each $U_i(y)$ for $i \in \{1,\ldots,n_1\}$ is at most $d_{\tilde{F}} - n_1 + 1 = w + 1$.

Therefore, $U_0(y)$ can be interpolated from its values on the set $W_w^{n-n_1}$, where the computation of each such value requires 2^{n_1} evaluations of $\tilde{F}(y,z)$. Thus, $U_0(y)$ can be interpolated from the values of $\tilde{F}(y,z)$ on the set $W_w^{n-n_1} \times \{0,1\}^{n_1}$. Similarly, $U_i(y)$ for $i \in \{1,\ldots,n\}$ can be interpolated given the values of $\tilde{F}(y,z)$ on the set $W_{w+1}^{n-n_1} \times \{0,1\}^{i-1} \times \{0\} \times \{0,1\}^{n_1-i}$. The proposition follows since $\tilde{F}$ is the identifying polynomial of $\tilde{E}$, and hence $\tilde{F}(\hat{y},\hat{z}) = 1$ if and only if $(\hat{y},\hat{z})$ is a solution to $\tilde{E}$. ■

Testing solutions. Similarly to [12], for each $\hat{y} \in \{0,1\}^{n-n_1}$, the algorithm computes $n_1 + 1$ sums. In our case, those with $U_0(\hat{y}) = 1$ suggest some solution $(\hat{y},\hat{z})$ to $\tilde{E}$ (and hence to E), and we need to test them. However, these tests make expensive evaluations of polynomials, which generally require about $\binom{n}{d}$ bit operations. This may lead to a large overhead, particularly for $d > 2$. In order to reduce this overhead, we repeat the algorithm a small number of times (using independent probabilistic polynomials) and test only candidate solutions that are output more than once. This is an additional concrete optimization over the second algorithm of [12], and makes use of assumptions about the input system to argue that it is unlikely for an incorrect candidate solution to be suggested more than once.

Comparison to the previous works [7,12]. Our algorithm differs from previous works in each of the three elements mentioned above.

First, unlike the worst-case setting of [7,12], isolating solutions in a cryptographic setting is essentially trivial. In particular, there is no need for a random change of variables, and the parameter n_1 will simply be chosen to optimize the complexity. In addition, the procedure of testing solutions is more efficient that the one of [12] as explained above.

Technically, a more interesting modification is that we enumerate isolated solutions of $\tilde{E}$ instead of E as in [12]. As a result, we only need to compute sums

of the form $\sum_{\hat{z}\in\{0,1\}^{n_1}} \tilde{F}(\hat{y},\hat{z})$. This is a significant concrete optimization, as accurate sums of the form $\sum_{\hat{z}\in\{0,1\}^{n_1}} F(\hat{y},\hat{z})$ are too expensive to compute directly due to the high degree of F. In previous algorithms of [7,12], such sums are computed by majority voting across $48\cdot n+1$ evaluations of different polynomials derived from E, which have to be interpolated and evaluated. Consequently, the complexity of our algorithm is reduced by a factor of $\Omega(n)$ while additional savings are obtained via Proposition 3.3. Thus, our algorithm eliminates majority voting altogether and uses probabilistic polynomials in a different and a more direct way to solve E.

The asymptotical complexity of the algorithm is determined by two terms, similarly to (5). It could be improved using the techniques of [12], essentially by recursively solving the multiple parity-counting problem on $\tilde{E}$ for sets of the form $W_w^{n-n_1}\times\{0,1\}^{n_1}$ (rather than applying brute force). Yet, these recursive calls have a significant concrete overhead (they require working with many more probabilistic polynomials) and we do not use them, as noted in Section 1.3.

4 Details and Analysis of the New Algorithm

The pseudo-code of our main algorithm in given in Algorithm 1. It uses procedures 1 and 2. We now describe it in detail and then analyze its complexity.

Details of algorithm 1. The main loop of Algorithm 1 runs until we find a solution of E. In each iteration, we define a new probabilistic set of equations $\tilde{E}$ from E and call Procedure 1 to output all candidate solutions of $\tilde{E}$. The output of Procedure 1 is a 2-dimensional $2^{n-n_1}\times(n_1+1)$ array that contains for each $\hat{y}\in\{0,1\}^{n-n_1}$, the evaluations $U_0(\hat{y})$ and $U_i(\hat{y})+1$ for $i\in\{1,\ldots,n_1\}$. Hence, by Proposition 3.2, assuming that $(\hat{y},\hat{z})$ is an isolated solution of $\tilde{E}$, then $U_0(\hat{y})=1$ and $U_i(\hat{y})+1=\hat{z}_i$ for $i\in\{1,\ldots,n_1\}$.

We store candidate solutions of $\tilde{E}$ in an array and check whether a potential solution has been output before (for a previous probabilistic set of equations). Such a potential solution is tested against the full system E.

Details of procedures 1 and 2. Procedures 1 and 2 output the potential solutions of a given system $\tilde{E}$ by interpolating the polynomials $U_i(y)$ for $i\in\{0,\ldots,n_1\}$ and evaluating them on all $\hat{y}\in\{0,1\}^{n-n_1}$.

Recall from Proposition 3.3 that $U_i(y)$ for $i\in\{0,\ldots,n_1\}$ can be interpolated by solving $\tilde{E}$ on the set $W_{w+1}^{n-n_1}\times\{0,1\}^{n_1}$, where $w=d_{\tilde{F}}-n_1$ and $d_{\tilde{F}}$ the degree of $\tilde{F}$. In procedure 2, these solutions are output by the fast exhaustive search algorithm of [8].

We denote by L the number of solutions and store them in memory. Next, we need to compute the values of each $U_i(y)$ on the sets described in Proposition 3.3 (with the aim of interpolating it).[5] These values are computed by summing the

[5] In practice, we do not need to store all the L solutions in memory at once, but we can interleave the exhaustive search with the computation of the $U_i(y)$ values.

evaluations of $\tilde{F}(y,z)$ on the corresponding subset of $\hat{z} \in \{0,1\}^{n_1}$. The values of all the polynomials $U_i(y)$ for $i \in \{0,\ldots,n_1\}$ are computed in parallel by iterating over the solutions of the system. For each solution, we calculate its contribution to each of the relevant polynomials. Having calculated the required values of each of the polynomials, we return them.

Then, in Procedure 1, we interpolate $U_i(y)$ for $i \in \{0,\ldots,n_1\}$ using the Möbius transform. Finally, we evaluate all the n_1+1 polynomials on the full range $\hat{y} \in \{0,1\}^{n-n_1}$ using the Möbius transform and output the potential solutions.

Parameters: $n_1, d_{\tilde{F}}$
Initialization: $\ell \leftarrow n_1+1, w \leftarrow d_{\tilde{F}} - n_1$
1: $PotentialSolutionsList[0\ldots] \leftarrow$ NewList()
2: **for all** $k = 0,1,\ldots$ **do**
3: Pick a uniformly random matrix of full rank ℓ, $A^{(k)} \in \mathbb{F}_2^{\ell \times m}$. Compute $\tilde{E}^{(k)} = \{R_i^{(k)}(x)\}_{i=1}^{\ell} = \{\sum_{j=1}^{m} A_{i,j}^{(k)} \cdot P_j(x)\}_{i=1}^{\ell}$
4: $CurrPotentialSolutions \leftarrow$ OutputPotentialSolutions$(\{R_i^{(k)}(x)\}_{i=1}^{\ell}, n_1, w)$
5: $PotentialSolutionsList[k] \leftarrow CurrPotentialSolutions$
6: **for all** $\hat{y} \in \{0,1\}^{n-n_1}$ **do**
7: **if** $CurrPotentialSolutions[\hat{y}][0] = 1$ ▷test whether $U_0(\hat{y}) = 1$, i.e., entry is valid **then**
8: **for all** $k_1 \in \{0,\ldots,k-1\}$ **do**
9: **if** $CurrPotentialSolutions[\hat{y}] = PotentialSolutionsList[k_1][\hat{y}]$ ▷check whether solution appears twice **then**
10: $sol \leftarrow \hat{y} \| CurrPotentialSolutions[\hat{y}][1\ldots n_1]$ ▷concatenate n_1 least significant bits
11: **if** TestSolution$(\{P_j(x)\}_{j=1}^{m}, sol) =$ TRUE **then**
12: **return** sol
13: **break** ▷continue with next $\hat{y}$

Algorithm 1: Solve$(\{P_j(x)\}_{j=1}^{m})$

4.1 Time Complexity Analysis

We now analyze the expected time complexity of the algorithm, denoted by $T = T_{n_1,d_{\tilde{F}}}(n,m,d)$, in terms of bit operations. For this purpose we define the following notation:

- $\mathcal{E}$ is the event that E has an isolated solution.
- N_k is the number main loop iterations of Algorithm 1.

1: $(V, ZV[1 \dots n_1]) \leftarrow \text{ComputeUValues}(\{R_i(y,z)\}_{i=1}^{\ell}, n_1, w)$
▷obtain values for interpolating each $U_i(y)$
2: Interpolate $U_0(y)$: apply Möbius transform to $V[1 \dots |W_w^{n-n_1}|]$
3: **for all** $i \in \{1, \dots, n_1\}$ **do**
4: Interpolate $U_i(y)$: apply Möbius transform to $ZV[i][1 \dots |W_{w+1}^{n-n_1}|]$
5: $Evals[0 \dots n_1][0 \dots 2^{n-n_1}-1] \leftarrow \vec{0}$ ▷init evaluation array
6: **for all** $i \in \{0, 1, \dots, n_1\}$ **do**
7: Evaluate $U_i(y)$ on $\{0,1\}^{n-n_1}$ by Möbius transform. Store result in $Evals[i][0 \dots 2^{n-n_1}-1]$
8: $Out[0 \dots 2^{n-n_1}-1][0 \dots n_1] \leftarrow \vec{0}$ ▷init output
9: **for all** $\hat{y} \in \{0,1\}^{n-n_1}$ **do**
10: **if** $Evals[0][\hat{y}] = 1$ **then**
11: $Out[\hat{y}][0] \leftarrow 1$ ▷indicate $U_0(\hat{y}) = 1$, i.e., entry is valid
12: **for all** $i \in \{1, \dots, n_1\}$ **do**
13: $Out[\hat{y}][i] \leftarrow Evals[i][\hat{y}] + 1$
▷copy potential solution by flipping evaluation bit
14: **return** Out

Procedure 1: OutputPotentialSolutions$(\{R_i(x)\}_{i=1}^{\ell}, n_1, w)$

- T_1 is the average complexity of an iteration of Algorithm 1, not including the complexity of testing solutions.
- N_s is the total number of solutions tested by Algorithm 1.
- T_s is the average complexity of testing a solution.

We have

$$T \leq N_k \cdot T_1 + N_s \cdot T_s. \tag{6}$$

Probabilistic setting. We assume that the polynomials of E are chosen independently and uniformly at random from all degree d polynomials, conditioned on having a pre-fixed solution chosen initially (e.g., a cryptographic key). A formal analysis of Algorithm 1 is given in the full version of the paper.

Below, we give a simple heuristical analysis, assuming each assignment $\hat{x}$ that is not the pre-fixed solution satisfies any polynomial equation in E with probability $1/2$ independently of the other assignments and equations.

Theorem 4.1 (heuristic). *For a random equation system, the success probability of Algorithm 1 is at least* $1 - 2^{n_1-m}$. *Given that* $m \geq 2 \cdot (n_1+1) + 2$ *and* $T_s \ll n_1 \cdot n \cdot 2^{n_1}$, *ignoring negligible factors, its expected running time is at most*

$$4\left(2d \cdot \log n \cdot 2^{n_1} \cdot \binom{n-n_1}{\downarrow d_{\bar{F}} - n_1 + 1} + n_1 \cdot n \cdot 2^{n-n_1}\right) \leq \tag{7}$$

$$4\left(2d \cdot \log n \cdot 2^{n_1} \cdot \binom{n-n_1}{\downarrow n_1 \cdot (d-1) + d + 1} + n_1 \cdot n \cdot 2^{n-n_1}\right). \tag{8}$$

1: $Sols[1 \dots L] \leftarrow \text{BruteForceSystem}(\{R_i(y,z)\}_{i=1}^{\ell}, n-n_1, w+1)$
▷brute force on space $W_{w+1}^{n-n_1} \times \{0,1\}^{n_1}$
2: $V[1 \dots |W_w^{n-n_1}|] \leftarrow \vec{0}$, $ZV[1 \dots n_1][1 \dots |W_{w+1}^{n-n_1}|] \leftarrow \vec{0}$ ▷init values for each $U_i(y)$
3: **for all** $(\hat{y}, \hat{z}) \in Sols[1 \dots L]$ **do**
4: **if** $\text{HW}(\hat{y}) \leq w$
▷values of HW more than w do not contribute to $U_0(y)$
then
5: $index \leftarrow \text{IndexOf}(\hat{y}, n-n_1, w)$ ▷get index of $\hat{y}$ in $W_w^{n-n_1}$
6: $V[index] = V[index] + 1$ ▷sum is over $\mathbb{F}_2$
7: **for all** $i \in \{1, \dots, n_1\}$ **do**
8: **if** $\hat{z}_i = 0$
▷only values with $z_i = 0$ contribute to $U_i(y)$ for $i > 1$
then
9: $index \leftarrow \text{IndexOf}(\hat{y}, n-n_1, w+1)$
10: $ZV[i][index] = ZV[i][index] + 1$
11: **return** $V, ZV[1 \dots n_1]$

Procedure 2: ComputeUValues$(\{R_i(y,z)\}_{i=1}^{\ell}, n_1, w)$

The "negligible factors" ignored are negligible both asymptotically and for relevant concrete parameter choices. Several terms will be neglected based on assumptions that n_1 is sufficiently large (such as $2^{n_1} \gg n$). We have already used and justified such assumptions (see Remark 2.2).

The total complexity of Algorithm 1. The complexity formula (8) establishes a tradeoff between two terms. First, the term

$$2d \cdot \log n \cdot 2^{n_1} \cdot \binom{n-n_1}{\downarrow w+1} \tag{9}$$

accounts for the brute force on the space $W_{w+1}^{n-n_1} \times \{0,1\}^{n_1}$ in Procedure 2 (based on the analysis of Section 2.3 for random systems). The second term accounts for the evaluation of the polynomials $U_i(y)$ on $\{0,1\}^{n-n_1}$ in Procedure 1 and is

$$(n_1+1) \cdot (n-n_1) \cdot 2^{n-n_1} \leq n_1 \cdot n \cdot 2^{n-n_1} \tag{10}$$

(given that $n_1^2 + n_1 \geq n$). The free parameter n_1 is set to balance these terms and optimize the complexity. Assuming the terms are equal, based on the second term, the gain in complexity over 2^n bit operations is roughly $\frac{2^{n_1}}{8 \cdot n_1 \cdot n}$. In the full version of the paper we estimate the total complexity of Algorithm 1 by about $n^2 \cdot 2^{0.815n}$ bit operations for $d = 2$ and $n^2 \cdot 2^{(1-1/2.7d)n}$ in general. This is obtained by setting $n_1 \approx \frac{n}{2.7d}$. Table 1 (at the end of this section) shows that $n^2 \cdot 2^{(1-1/2.7d)n}$ slightly overestimates the complexity for relevant parameters.

Next, we establish Theorem 4.1 and argue that the condition $T_s \ll n_1 \cdot n \cdot 2^{n_1}$ holds both asymptotically and for relevant concrete parameter choices.

Success probability analysis. To analyze $\mathcal{E}$ (the event that E has an isolated solution), examine a specific solution to E. It is placed in a group with $2^{n_1}-1$ additional potential solutions. We thus estimate the probability that another solution exists in this group by 2^{n_1-m}, and $\mathcal{E}$ holds with probability $1-2^{n_1-m}$.

Time complexity analysis. The algorithm succeeds once a solution to E is isolated twice. Given that $\mathcal{E}$ holds, by Proposition 3.1, every iteration isolates a particular solution with probability at least $\frac{1}{2}$. Hence the expected number of iterations is at most $N_k \leq 2 \cdot 2 = 4$.

We analyze the most expensive operations of procedures 2 and 1, showing that the terms (9) and (10) indeed dominate. We then analyze the cost of testing solutions.

Procedure 2. As noted above, using the analysis of Section 2.3, the brute force complexity is given by (9). In addition, we estimate

$$L = 2^{-\ell} \cdot |W_{w+1}^{n-n_1} \times \{0,1\}^{n_1}| = \tfrac{1}{2} \cdot \tbinom{n-n_1}{\downarrow w+1}, \tag{11}$$

as the equation systems we solve by brute force have $\ell = n_1 + 1$ equations.

The complexity of computing the values of the arrays V and ZV is slightly more[6] than $(n_1+1) \cdot L$, which is negligible compared to (9), given that $2d \cdot \log n \cdot 2^{n_1} \gg \frac{1}{2}(n+1)$.

Procedure 1. The complexity of interpolating $(U_0(y), U_1(y), \ldots, U_{n_1}(y))$ from their evaluations is $n \cdot \binom{n-n_1}{\downarrow w} + n_1 \cdot n \cdot \binom{n-n_1}{\downarrow w+1} < (n_1+1) \cdot n \cdot \binom{n-n_1}{\downarrow w+1}$, which is negligible compared to (9) given that $2d \cdot \log n \cdot 2^{n_1} \gg (n_1+1) \cdot n$.

The complexity of evaluating these polynomials on $\{0,1\}^{n-n_1}$ is given in (10).

Estimating N_s. Fix an iteration pair $0 \leq i < j < N_k$. Given that we have at least $m \geq 2 \cdot (n_1+1) + 2 = 2 \cdot \ell + 2$ equations, the 2ℓ rows of $A^{(i)}$ and $A^{(j)}$ are linearly independent vectors over $\{0,1\}^m$ with high probability[7] in which case $\tilde{E}^{(i)}$ and $\tilde{E}^{(j)}$ are independent equation systems with ℓ equations. Hence, restricting these systems to $\hat{y} \in \{0,1\}^{n-n_1}$, the pair suggests the same n_1-bit solution suffix $\hat{z}$ with probability 2^{-n_1}. Considering all $\hat{y} \in \{0,1\}^{n-n_1}$, the expected number of suggested solutions is about 2^{n-2n_1}. As the number of iteration pairs is small and many systems restricted to $\hat{y}$ do no suggest any solution since $U_0(\hat{y}) = 0$, we estimate

$$N_s = 2^{n-2n_1}. \tag{12}$$

Since (9) and (10) dominate T_1, up to now we estimate

$$T \leq N_k \cdot T_1 + N_s \cdot T_s \lessapprox 4 \cdot (2d \cdot \log n \cdot 2^{n_1} \cdot \tbinom{n-n_1}{\downarrow w+1} + n_1 \cdot n \cdot 2^{n-n_1}) + 2^{n-2n_1} \cdot T_s.$$

[6] We note that the operations of the IndexOf functions can be implemented with small overhead because solutions are output by the brute force algorithm in fixed order.

[7] Even if the rows of $A^{(i)}$ and $A^{(j)}$ have a few linear dependencies, it does not substantially affect the analysis.

In order to establish Theorem 4.1, we need to show that

$$T_s \ll n_1 \cdot n \cdot 2^{n_1}, \tag{13}$$

so the final term that corresponds to the total complexity of testing candidate solutions may be neglected compared to the term $4 \cdot n_1 \cdot n \cdot 2^{n-n_1}$.

Testing solutions. Naively testing a candidate solution requires evaluating two polynomials in E on average, which has complexity of about $2 \cdot \binom{n}{\downarrow d}$ bit operations. Asymptotically, for constant d, this complexity is negligible compared to $2^{n_1} = 2^{\Omega(n)}$, hence (13) holds.

Concretely, for $d = 2$, since $n \ll 2^{n_1}$, then $n^2 \ll n_1 \cdot n \cdot 2^{n_1}$ and (13) holds. However, when $d > 2$ is relatively large compared to n, then (13) may no longer hold for relevant parameter choices (e.g., for $d = 4$ and $n = 128$).

On the other hand, we can reduce this complexity such that it becomes negligible for relevant parameter choices by tweaking Algorithm 1. The main idea is to test the potential solutions in batches, reducing the amortized complexity as described in the full version of this paper.

In practice, E is constructed from a cryptosystem, and for relatively large d one may simply test candidate solutions by directly evaluating the cryptosystem.

Experimental validation. The most important probabilistic quantities analyzed are L (11) and N_s (12) (the bound on the expected value of N_k is rigourous). As detailed in the full version of this paper, we experimentally calculated their values (for random equation systems with $m = n$), and conclude that our estimate of L is accurate, while the estimate of N_s is somewhat conservative.

4.2 Optimizing Memory Complexity

The expected memory complexity of the algorithm is about $4 \cdot (n_1+1) \cdot 2^{n-n_1}$ bits, dominated by storing the potential solutions output by the different executions of Procedure 1. A simple way to obtain a time-memory tradeoff is to guess several bits of x and repeat the algorithm for each guess. However, we can improve the memory complexity with essentially no penalty by making use of a memory-efficient implementation of the Möbius transform.

Memory-efficient Möbius transform. We deal with the problem of evaluating a polynomial $F(x_1, \ldots, x_n)$ of degree d on the space $\{0,1\}^n$ using the Möbius transform. Assume that d is not too large and the polynomial is represented by a bit array of size $\binom{n}{\downarrow d} \ll 2^n$. Moreover, assume that the application does not need to store the evaluation of the polynomial on the full space, but can work even if the space is partitioned into smaller spaces on which the polynomial is evaluated on the fly. Then, the memory complexity can be reduced as follows. Instead of allocating an array of size 2^n, we work directly with the recursive

formula (2), $F(x_1,\ldots,x_n) = x_1 \cdot F_1(x_2,\ldots,x_n) + F_2(x_2,\ldots,x_n)$, by first evaluating $F_2(x_2,\ldots,x_n)$ (i.e., $F(x)$ for $x_1 = 0$), and then calculating and evaluating $F_1(x_2,\ldots,x_n) + F_2(x_2,\ldots,x_n)$ (i.e., $F(x)$ for $x_1 = 1$).

This algorithm does not work in-place, but only keeps in memory the recursion stack. The memory complexity is bounded by the formulas $M(n,d) = M(n-1,d) + \binom{n}{\downarrow d}$ and $M(n,n) = 2^n$. Thus, the total memory complexity is less than $n \cdot \binom{n}{\downarrow d}$. The time complexity in bit operations is bounded by $\binom{n}{\downarrow d} + 2 \cdot \binom{n-1}{\downarrow d} + \ldots + 2^{n-d-1} \cdot \binom{d+1}{\downarrow d} + 2^{n-d} \cdot d \cdot 2^d$.

Remark 4.1. A more precise evaluation reveals that the total number of bit operations is about $d \cdot 2^n$. The exhaustive search algorithm of [8] for enumerating all zeroes of a polynomial of degree d also requires $d \cdot 2^n$ bit operations. It would be interesting to investigate whether the recursive Möbius transform can compete with [8] in practice. We note that it was already observed in [15] that the Möbius transform on degree d polynomials requires $d \cdot 2^n$ bit operations, but the algorithm used a standard implementation with memory complexity of 2^n.

In our context, we will exploit the lower complexity of the top level recursive calls to further reduce the memory complexity, while keeping the time complexity below $n \cdot 2^n$. Specifically, for a parameter $k \approx n - \log \binom{n}{\downarrow d}$, we perform the top k levels of the recursion independently without saving the recursion stack (i.e., we recursively evaluate the input polynomial on all 2^k values of $x_1,\ldots,x_k$ independently). At the bottom levels, we switch to the in-place implementation of the Möbius transform to evaluate the polynomial on all values of $x_{k+1},\ldots,x_n$. In order to perform the independent evaluations, we only need to allocate two additional arrays, one for the input to the recursive call and one for its output. The roles of these arrays are interchanged on every recursive call. The memory required for each such additional array is bounded by $\binom{n}{\downarrow d}$. The memory required for the in-place Möbius transform is $2^{n-k} \approx \binom{n}{\downarrow d}$. Therefore, the in-place transform does not require additional memory and the total memory complexity is bounded by $3 \cdot \binom{n}{\downarrow d}$. The time complexity of the procedure is bounded by $2^k \cdot \left(\binom{n}{\downarrow d} + \ldots + \binom{n-k}{\downarrow d}\right) + (n-k) \cdot 2^n < n \cdot 2^n$.

We note that the algorithm of [8] could also be used for the same purpose. However, it requires a preprocessing phase of complexity n^{2d}. On the other hand, we will use the Möbius transform variant with relatively large d (e.g., $d = n/3$), for which such preprocessing is too expensive.

Improving the memory complexity of algorithm 1. In order to reduce the memory complexity we first interpolate the polynomials $(U_0(y),\ldots,U_{n_1}(y))$ for several executions (e.g., 4 or a bit more) of Procedure 1 in advance. Using the recursive version of the Möbius transform (as described in Section 2.3), the additional memory required is negligible.

The main idea that allows to save memory is to interleave the tasks of evaluating all the polynomials (in parallel) with testing solutions that are suggested at least twice. The parallel evaluation is performed using the memory-optimized

implementation of the Möbius transform. This reduces the memory complexity to about 3 times the memory required to store all the polynomials. In fact, for the purpose of testing solutions, we only need to keep the evaluations of each such polynomial on a space proportional to its size, used by the in-place transform. Thus, when sequentially calculating several transforms, we reuse one of the two additional allocated arrays. In total, we require 2 times the memory used for storing all the polynomials, namely

$$8 \cdot (n_1 + 1) \cdot \binom{n-n_1}{\downarrow d_{\bar{F}} - n_1 + 1}. \tag{14}$$

Choosing n_1 that minimizes time complexity (balancing the two terms of (7)), gives $\binom{n-n_1}{\downarrow d_{\bar{F}} - n_1 + 1} \approx \frac{n_1 \cdot n}{2d \cdot \log n} \cdot 2^{n-2n_1}$ and total memory complexity of about $\frac{4 \cdot n_1 \cdot (n_1+1) \cdot n}{d \cdot \log n} \cdot 2^{n-2n_1}$ bits. Compared to Algorithm 1, this saves a multiplicative factor of about $\frac{d \cdot \log n}{n_1 \cdot n} \cdot 2^{n_1}$. Since $n_1 < \frac{n}{5}$, $\frac{8 \cdot n_1 \cdot (n_1+1) \cdot n}{2d \cdot \log n} \approx n^2$ for relevant concrete parameters.

Asymptotically, the memory complexity is $O(n^3 \cdot 2^{n-2n_1})$ bits. A choice of n_1 that minimizes time complexity gives $O\left(n^3 \cdot 2^{0.63n}\right)$ bits for $d = 2$ and $O\left(n^3 \cdot 2^{(1-1/1.35d)n}\right)$ in general.

Concrete parameters. In Table 1 we give concrete complexity estimates for interesting parameter sets after optimizing the free parameter n_1 of (8).

Variables n	Degree d	Internal parameter n_1	Complexity (bit operations)	Memory (bits)	Exhaustive search [8] ($2d \log n \cdot 2^n$)
80	2	16	2^{77}	2^{60}	2^{84}
128	2	25	2^{117}	2^{91}	2^{133}
128	4	12	2^{129}	2^{112}	2^{134}
192	2	37	2^{170}	2^{132}	2^{197}
192	4	18	2^{188}	2^{164}	2^{198}
256	2	49	2^{223}	2^{173}	2^{261}
256	4	25	2^{246}	2^{219}	2^{262}

Table 1. Concrete complexity of (the memory-optimized variant of) Algorithm 1

5 Cryptanalytic Applications

In this section we describe cryptanalytic applications of Algorithm 1. Our main application is in cryptanalysis of Picnic (and LowMC) variants and our secondary application is in cryptanalysis of round-reduced Keccak. We begin by describing the general optimization method we use in cryptanalysis of Picnic.

5.1 Deterministic Replacement of Probabilistic Polynomials

In cryptanalytic applications, E may have some properties that depend on the underlying cryptosystem and could be ruined in $\tilde{E}$ that mixes the equations of E. We observe that in cryptographic settings, we may replace the randomized construction of $\tilde{E}$ (and $\tilde{F}$) by a deterministic construction which simply takes subsets of equations from E, thus preserving the properties of E.

Essentially, the randomness of the probabilistic constructions is already "embedded" in E itself. For example, we still (heuristically) expect a variant of Proposition 3.1 to hold: if $\hat{x}$ is a solution to E, it is also a solution to $\tilde{E}$. On the other hand, since $\tilde{E}$ is an system with $\ell = n_1 + 1$ equations in n_1 variables, we expect it not to have an additional solution with high probability.

In order for the number of tested candidate solutions to remain small, we require the different equations systems $\tilde{E}$ analyzed to be roughly independent. Specifically, the intersections of the equations subsets taken for different "probabilistic" equations systems $\tilde{E}$ should be empty (or small).

5.2 Picnic and LowMC

The Picnic signature scheme [10] is an alternate third-round candidate in NIST's post-quantum standardization project [29]. It uses a zero-knowledge protocol in order to non-interactively prove knowledge of a preimage x to a public value y under a one-way function f, where y is part of the public key and x is the secret signing key. The one-way function is implemented using a block cipher, where the secret signing key is the block cipher's key, while the public key consists of a randomly chosen plaintext and the corresponding ciphertext (the encryption of the plaintext with the secret key). Thus a key-recovery attack on Picnic reduces to finding the block cipher's secret key from one plaintext-ciphertext pair.

Picnic uses the LowMC block cipher family, proposed at EUROCRYPT 2015 by Albrecht et al. [1]. It is optimized for practical instantiations of multi-party computation, fully homomorphic encryption, and zero-knowledge proofs, in which non-linear operations are typically much more expensive than linear ones. Consequently, LowMC uses a relatively small number of multiplications.

LowMC is an SP-network built using several rounds, where in each round, a round-key is added to the state, followed by an application of a linear layer and a non-linear layer (operations are over $\mathbb{F}_2$). Finally, an additional round-key is added to the state. Importantly, the key schedule of LowMC is linear.

Each non-linear layer of LowMC consists of identical Sboxes $\mathcal{S} : \{0,1\}^3 \rightarrow \{0,1\}^3$ of algebraic degree 2. The algebraic normal form of an Sbox is

$$\mathcal{S}(a_1, a_2, a_3) = (a_1 + a_2 a_3, a_1 + a_2 + a_1 a_3, a_1 + a_2 + a_3 + a_1 a_2). \tag{15}$$

We note that the inverse Sbox also has algebraic degree of 2.

In this paper, we focus on LowMC instances that were recently integrated into Picnic variants [33]. These instances have internal state and key sizes of $129, 192$ and 255, claiming security levels of $128, 192$ and 255 bits, respectively

and have a full non-linear layer (as opposed to other instances of LowMC that have a partial non-linear layer). All of these instances have 4 rounds, although in a recent publication by some of the designers [23] additional instances with 5 rounds were proposed in order to provide a larger security margin.

Attacks on LowMC instances. As noted above, we analyze LowMC instances with a full Sbox layer given only a single plaintext-ciphertext pair. The best-known attacks on such instances were recently published by Banik et al. [2] where the authors analyzed instances reduced to 2 rounds. Their techniques are based on linearization and it is not clear how to extend them beyond 2 rounds without exceeding the complexity of (optimized) exhaustive search [8].

We focus on the full 4 and 5-round instances. We fix an arbitrary plaintext-ciphertext pair to a LowMC instance with key and internal state size of n bits. We denote the unknown key by $x = (x_1, \ldots, x_n)$.

Even round number. We begin by considering LowMC instances with an even number of rounds r. We focus on an arbitrary state bit b_i (for some $i \in \{1, \ldots, n\}$) after $r/2$ rounds. Starting from the plaintext, we symbolically evaluate the encryption process and express b_i as a polynomial $p_i(x)$. Similarly, we symbolically evaluate the decryption process starting from the ciphertext and express b_i as a polynomial $q_i(x)$. This standard meet-in-the-middle approach gives rise to the equation $p_i(x) + q_i(x) = 0$. As the algebraic degree of the LowMC round and its inverse is 2 and the key schedule is linear, the algebraic degree of both $p_i(x)$ and $q_i(x)$ is at most $2^{r/2}$. Repeating this process n times for all intermediate state bits, we obtain an equation systems with $m = n$ equations. For the small values of r we consider, the complexity of calculating the equation system is negligible. We can now apply Algorithm 1 and solve for the secret key.

Odd round number. For an odd number of rounds r, the approach above gives rise to equations of degree at least $2^{(r+1)/2}$, as in general, any intermediate state bit has algebraic degree of at least $2^{(r+1)/2}$ from either the encryption or the decryption side. If we apply Algorithm 1 in a straightforward manner, its complexity compared to an attack on an even number of $r-1$ rounds would increase substantially. We now show that by a better choice of the equation system and careful analysis we can reduce the algorithm's complexity.

Consider the first 3 intermediate state bits b_1, b_2, b_3 that are outputs of the first Sbox in round $(r+1)/2$. From the decryption side, we can express them as polynomials of degree $2^{(r-1)/2}$, denoted by $q_1(x), q_2(x), q_3(x)$, respectively. From the encryption side, based on (15), we can express these bits as functions of the bits a_1, a_2, a_3 that are inputs the first Sbox in round $(r+1)/2$,

$$(b_1, b_2, b_3) = \mathcal{S}(a_1, a_2, a_3) = (a_1 + a_2a_3, a_1 + a_2 + a_1a_3, a_1 + a_2 + a_3 + a_1a_2).$$

From the encryption side, we can express each of a_1, a_2, a_3 as a polynomial of degree $2^{(r-1)/2}$ in the key. Equating each bit to its evaluation from the decryption

side, we obtain the 3 equations

$$\begin{aligned} q_1(x) + a_1(x) + a_2(x)a_3(x) &= 0, \\ q_2(x) + a_1(x) + a_2(x) + a_1(x)a_3(x) &= 0, \\ q_3(x) + a_1(x) + a_2(x) + a_3(x) + a_1(x)a_2(x) &= 0. \end{aligned}$$

Each polynomial appearing in these equations is of algebraic degree $2^{(r-1)/2}$.

Recall that the main complexity formula (7) heavily depends on the value of $d_{\tilde{F}}$. This value is upper bounded by $d \cdot \ell$, but it could be lower if we choose $\tilde{F}$ more carefully. Indeed, we will construct the "probabilistic polynomials" deterministically using (non-overlapping) subsets of the original equation system E. The probabilistic analysis is essentially unchanged, as suggested in Section 5.1.

Specifically, in this case, each equation is of degree $2^{(r-1)/2} + 2^{(r-1)/2} = 2^{(r+1)/2}$ due to the multiplication of the a_i's. However, if we multiply the 3 polynomials for the purpose of calculating $\tilde{F}$ (as in (4), but with R_i's replaced by the equations above), the term $a_1(x)a_2(x)a_3(x)$ can only be multiplied with at most one of the $q_i(x)$'s and therefore the degree of multiplication of all the equations is at most $4 \cdot 2^{(r-1)/2} = 2^{(r+3)/2}$, rather than the trivial upper bound of $6 \cdot 2^{(r-1)/2}$, reducing the degree by a factor of $\frac{1}{3}$. For example, for $r = 3$, we get a bound of 8 instead of the general upper bound of 12, whereas for $r = 5$, we get a bound of 16 instead of the general upper bound of 24.

We proceed to collect n equations as before. However, in Algorithm 1, for some integer ℓ' we analyze equations that are computed as above using ℓ' triplets that are outputs of the Sbox layer of round $(r+1)/2$. We now have $\ell = 3\ell'$ equations. The total degree of $\tilde{F}$ in (4) is upper bounded by $d_{\tilde{F}} \leq \ell' \cdot 2^{(r+3)/2} = \ell/3 \cdot 2^{(r+3)/2}$. We choose $n_1 = \ell - 1$ as before. Revisiting the complexity analysis formula of (7), we obtain

$$4\left(2 \cdot 2^{(r+1)/2} \cdot \log n \cdot 2^{n_1} \cdot \binom{n-n_1}{\downarrow \frac{\ell}{3} \cdot 2^{(r+3)/2} - n_1 + 1} + n_1 \cdot n \cdot 2^{n-n_1}\right). \tag{16}$$

If we take ℓ which is not a multiple of 3, then $d_{\tilde{F}}$ increases more sharply due to the last Sbox. Specifically, if $\ell \bmod 3 = 1$, then $d_{\tilde{F}} \leq \frac{\ell-1}{3} \cdot 2^{(r+3)/2} + 2^{(r+1)/2}$, whereas if $\ell \bmod 3 = 2$, then $d_{\tilde{F}} \leq \frac{\ell-2}{3} \cdot 2^{(r+3)/2} + 3 \cdot 2^{(r-1)/2}$. Standard approaches to deal with the middle Sbox layer (e.g., linearization) result in higher complexity.

Results. Table 2 summarizes our attacks on instances of LowMC used in recent Picnic variants (and additional 5-round instances). Solutions can be tested simply by evaluating the LowMC encryption process, whose most expensive procedures are the evaluations of the linear layers, each consisting of a multiplication of an n-bit state with an $n \times n$ matrix. Naively, this has complexity of $2n^2$ bit operations. It can be checked that (13) holds for the parameters of Table 2.

5.3 Keccak

Keccak is a family of cryptographic functions, designed by Bertoni et al. in 2008 [5]. We focus on the Keccak hash function family, selected by NIST in 2015

Security level S	Key length n	Rounds r	Internal parameters $(n_1, d, d_{\tilde{F}})$	Attack complexity (bit operations)
128	129	4	$(12, 4, 52)$	2^{130}
192	192	4	$(18, 4, 76)$	2^{188}
192	192	5	$(14, 8, 80)$	2^{192}
255	255	4	$(25, 4, 104)$	2^{245}
255	255	5	$(18, 8, 104)$	2^{251}

Table 2. Attacks on LowMC instances with 4 rounds (used in Picnic) and 5 rounds

as the SHA-3 standard. It is built using the sponge construction (cf. [6]) using a permutation that operates on a 1600-bit state. The permutation consists of 24 rounds, where each round consists of an application of a non-linear layer, followed by linear operations over $\mathbb{F}_2$. Importantly, the non-linear layer is of algebraic degree 2. We analyze the 4 basic Keccak variants which are parameterized by the output size of k bits and denoted by Keccak-k for $k \in \{224, 256, 384, 512\}$.

Preimage attacks of round-reduced Keccak. We consider messages of length that is smaller than the *rate* of the hash function (so that the output is produced after a single invocation of the permutation). We start by representing the message (preimage) bits as symbolic variables. We then linearize the first round of Keccak by setting some linear constraints on the variables, such that each state bit in the second round of the permutation is a linear polynomial in these variables. Using the linearization technique of [13,20] for Keccak-k (that selects variables that keep the *column parities* constant), this leaves more than 224 and 256 free variables for Keccak-224 and Keccak-256, respectively, and 256 and 128 free variables for Keccak-384 and Keccak-512, respectively (for the SHA-3 versions, the number is slightly smaller).

For Keccak-384 and Keccak-512, we can further partially invert the final non-linear layer (applied to the first 5×64 Sboxes) on the target image to obtain its input values. For Keccak-224 and Keccak-256, not all these 5×64 output bits are fixed by the image. However, we obtain and 192 and 256 linear relations among the input bits of the final Sbox layer, for Keccak-224 and Keccak-256 respectively (e.g., see [20]). Having peeled off 2 out of the 4 non-linear layers, we obtain equations of degree $2^2 = 4$ and solve for the preimage using Algorithm 1.

Results. We begin by considering preimage attacks on 4 rounds of Keccak-224 and Keccak-256. These instances were recently analyzed in by Li and Sun [25], who devised attacks with claimed complexities of 2^{207} and 2^{239}, on Keccak-224 and Keccak-256, respectively. However, the analysis of these attacks ignores the complexity of solving (numerous) linear equation systems over $\mathbb{F}_2$ with hundreds of variables, each requiring many thousands of bit operations.

For Keccak-224 and Keccak-256 we have sufficiently many free variables to obtain systems with 224 and 256 variables (respectively), which we assume to

have a solution. Consequently, our preimage attacks have complexities of 2^{217} and 2^{246} bit operations by choosing $n_1 = 22$ and $n_1 = 25$, respectively. These are lower complexities than those obtained in [25] in terms of bit operations.

For Keccak-384, the number of free variables is only 256, so we need to solve systems of degree 4 with 256 variables an expected number of $2^{384-256} = 2^{128}$ times (with different initial linear constraints on the variables) to obtain a solution. This requires time $2^{256+128} = 2^{374}$ (by choosing $n_1 = 25$). In terms of bit operations, this improves upon the recent result of [26], which estimated the attack complexity by 2^{375} evaluations of the 4-round Keccak-384 function.

For Keccak-512, we do not linearize the first round, but directly solve a system of degree 8 in 512 variables. This requires 2^{502} bit operations (by choosing $n_1 = 26$) and improves the previous attack [28] that requires 2^{506} Keccak calls.

Collision attacks on round-reduced Keccak. The problem of finding a collision can be formulated as a non-linear equation system (where the variables are the bits of two colliding messages). However, the complexity of solving such a system is unlikely to be more efficient than a generic birthday attack on the k-bit hash function which takes time $2^{k/2}$. A better idea is to directly speed up the generic birthday attack. In order to do so, for a parameter ℓ, we fix ℓ bits of the output to an arbitrary value v and try to find $2^{(k-\ell)/2}$ messages whose output value on the ℓ bits is equal to v. With high probability, these messages contain a pair whose outputs also agree of the remaining $k - \ell$ bits that we have not fixed, and therefore constitute a colliding pair. In order to find $2^{(k-\ell)/2}$ such messages, we apply Procedure 1.

Suppose we run Procedure 1 with n variables after fixing $\ell = n_1 + 1$ output bits. We expect the output to contain about $\frac{1}{2} \cdot 2^{n-n_1}$ isolated solutions (messages) that satisfy the n_1 constraints. We evaluate the hash function on each output message, and test whether it is indeed a solution. We store all the true solutions and sort them, trying to find a collision among them. If $\frac{1}{2} \cdot 2^{n-n_1} < 2^{(k-\ell)/2}$ (e.g., we lack degrees of freedom and are forced to choose a small value of n), we repeat the procedure several times (with a different set of message variables), while storing all the produced messages until a collision is found.

Since we test all outputs of Procedure 1, the complexity of this attack also directly depends on the number of bit operations required to evaluate the hash function on some message. We denote this number by τ.

We apply the framework to 4-round Keccak-512, as there are no published attacks better than the birthday bound on this variant. Assuming $\tau = 2^{13}$ (hence the complexity of the birthday attack is $2^{256+13} = 2^{269}$ bit operations), we set $d = 4$ and $n = 128$ (unlike the preimage attack, we linearize the first round) and choose $n_1 = \ell - 1 = 12$. Calculation reveals that the complexity is about 2^{263} bit operations, which is roughly 64 times faster than the birthday attack.[8] If we assume a larger value of τ, the complexity of the attack will increase, but also its relative advantage compared to the birthday attack.

[8] The complexity of sorting the $2^{(512-13)/2} = 2^{249.5}$ images is estimated to be smaller than 2^{262} bit operations.

Acknowledgements. The author was supported by the Israeli Science Foundation through grant No. 573/16 and grant No. 1903/20, and by the European Research Council under the ERC starting grant agreement No. 757731 (LightCrypt).

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015)
2. Banik, S., Barooti, K., Durak, F.B., Vaudenay, S.: Cryptanalysis of LowMC instances using single plaintext/ciphertext pair. IACR Transactions on Symmetric Cryptology 2020(4), 130–146 (2020), https://tosc.iacr.org/index.php/ToSC/article/view/8751
3. Bardet, M., Faugère, J., Salvy, B., Spaenlehauer, P.: On the complexity of solving quadratic Boolean systems. J. Complex. 29(1), 53–75 (2013)
4. Beigel, R.: The Polynomial Method in Circuit Complexity. In: Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993. pp. 82–95. IEEE Computer Society (1993)
5. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference https://keccak.team/files/Keccak-reference-3.0.pdf
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer (2008)
7. Björklund, A., Kaski, P., Williams, R.: Solving Systems of Polynomial Equations over GF(2) by a Parity-Counting Self-Reduction. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece. LIPIcs, vol. 132, pp. 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
8. Bouillaguet, C., Chen, H., Cheng, C., Chou, T., Niederhagen, R., Shamir, A., Yang, B.: Fast Exhaustive Search for Polynomial Systems in F_2. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 203–218. Springer (2010)
9. Casanova, A., Faugère, J.C., Macario-Rat, G., Patarin, J., Perret, L., Ryckeghem, J.: GeMSS: A Great Multivariate Short Signature. Submission to NIST (2017), https://www-polsys.lip6.fr/Links/NIST/GeMSS.html
10. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1825–1842. ACM (2017)

11. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 392–407. Springer (2000)
12. Dinur, I.: Improved Algorithms for Solving Polynomial Systems over GF(2) by Multiple Parity-Counting. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, January 10-13, 2021. pp. 2550–2564. SIAM (2021)
13. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 733–761. Springer (2015)
14. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)
15. Dinur, I., Shamir, A.: An Improved Algebraic Attack on Hamsi-256. In: Joux, A. (ed.) Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6733, pp. 88–106. Springer (2011)
16. Duarte, J.D.: On the Complexity of the Crossbred Algorithm. IACR Cryptol. ePrint Arch. 2020, 1058 (2020), https://eprint.iacr.org/2020/1058
17. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139(1-3), 61–88 (Jun 1999)
18. Faugère, J.C.: A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. pp. 75—-83. ISSAC '02, Association for Computing Machinery, New York, NY, USA (2002)
19. Faugère, J., Joux, A.: Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 44–60. Springer (2003)
20. Guo, J., Liu, M., Song, L.: Linear Structures: Applications to Cryptanalysis of Round-Reduced Keccak. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 249–274 (2016)
21. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC, 1st edn. (2009), pages 285-286
22. Joux, A., Vitse, V.: A Crossbred Algorithm for Solving Boolean Polynomial Systems. In: Kaczorowski, J., Pieprzyk, J., Pomykala, J. (eds.) Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10737, pp. 3–21. Springer (2017)

23. Kales, D., Zaverucha, G.: Improving the Performance of the Picnic Signature Scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020(4), 154–188 (2020)
24. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar Signature Schemes. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 206–222. Springer (1999)
25. Li, T., Sun, Y.: Preimage Attacks on Round-Reduced Keccak-224/256 via an Allocating Approach. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 556–584. Springer (2019)
26. Liu, F., Isobe, T., Meier, W., Yang, Z.: Algebraic Attacks on Round-Reduced Keccak/Xoodoo. IACR Cryptol. ePrint Arch. 2020, 346 (2020), https://eprint.iacr.org/2020/346
27. Lokshtanov, D., Paturi, R., Tamaki, S., Williams, R.R., Yu, H.: Beating Brute Force for Systems of Polynomial Equations over Finite Fields. In: Klein, P.N. (ed.) Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19. pp. 2190–2202. SIAM (2017)
28. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced keccak. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 241–262. Springer (2013)
29. NIST's Post-Quantum Cryptography Project, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
30. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In: Maurer, U.M. (ed.) Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding. Lecture Notes in Computer Science, vol. 1070, pp. 33–48. Springer (1996)
31. Razborov, A.A.: Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. Mathematical Notes of the Academy of Sciences of the USSR 41(4), 333—-338 (1987)
32. Smolensky, R.: Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 77–82. ACM (1987)
33. The Picnic Design Team: The Picnic Signature Algorithm Specification. April 2020. Version 3.0. https://microsoft.github.io/Picnic/
34. Valiant, L.G., Vazirani, V.V.: NP is as Easy as Detecting Unique Solutions. Theor. Comput. Sci. 47(3), 85–93 (1986)
35. Williams, R.R.: The Polynomial Method in Circuit Complexity Applied to Algorithm Design (Invited Talk). In: Raman, V., Suresh, S.P. (eds.) 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India. LIPIcs, vol. 29, pp. 47–60. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2014)