A decrementally-improved algorithm for Boolean MQ

Charles Bouillaguet[1 [0000−0001−9416−6244]]
and Julia Sauvage[1 [0009−0004−0239−1241]]

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
{charles.bouillaguet,julia.sauvage}@lip6.fr

Abstract. The MQOM signature scheme is currently a third-round candidate in the NIST competition for additional signatures. It is based on the "MPC-in-the-Head" paradigm and relies on the hardness of the MQ problem. Some of its parameter sets expose a Boolean quadratic system in the public key. While the situation for MQ over larger fields has been relatively quiescent over the last decade, Boolean MQ has seen active progress, culminating with Dinur's algorithms at SODA 2021 and Eurocrypt 2021.

We propose yet another algorithm for Boolean MQ. It is a hybrid between the "polynomial-method" of Lokshtanov, Paturi, Tamaki, Williams and Yu from SODA 2017 and Dinur's "second algorithm" from Eurocrypt 2021. We remove some machinery from the latter to obtain a modest improvement of 1–4 bits in performance for MQOM parameters ("decremental improvement").

MQOM optionally uses the "correlated GGM trees" technique to shorten signatures; in that case, its security also relies on the hardness of the "Partial-Guessing One-Wayness" problem for MQ (PGOW-MQ): given an MQ system supposed to offer λ bits of security, the adversary has to find the first λ bits of a solution, and they have access to an oracle that enables them to check candidate prefixes. The designers of MQOM implicitly assumed that PGOW-MQ is as hard as MQ itself. Our algorithm can exploit the availability of the solution-testing oracle to solve PGOW-MQ 2 to 4 times faster than it solves MQ, thus showing that the two problems are marginally different. This yields attacks against MQOM that are 3–4 bits below the expected security level, but that suffer from huge memory complexities.

Lastly, we survey old and new techniques to find an invertible linear change of variables that puts a few arbitrary polynomials in UOV shape. This leads to a small acceleration of our algorithm, and also incidentally improves upon the Thomae-Wolf and Furue-Nakamura-Takagi algorithms to solve underdetermined Boolean systems. A new idea based on matrix pencils was used to solve the largest underdetermined Boolean Fukuoka MQ challenges and may be of independent interest.

Keywords: Boolean MQ, MQOM, polynomial method, Partial-Guessing One-Way problem, underdetermined systems

1 Introduction

This article presents an algorithm to solve Boolean quadratic systems and discusses its consequences on the security level offered by the parameter sets chosen by the designers of MQOM [4]. MQOM is presently a third-round candidate in the NIST competition for additional signature schemes. It is based on (a variant of) the "MPC-in-the-head" paradigm. An MQOM public key is a random quadratic system of n equations in n variables over a finite field, and a signature is essentially a proof of knowledge of a solution to this system, with random challenges derived from the message to sign. As such, the security of MQOM relies on the hardness of solving random quadratic systems. This is the Multivariate Quadratic (MQ) problem and it is well-known to be NP-complete.

MQOM proposes several parameter sets over several finite fields, and in particular over the Boolean field $\mathbb{F}_2$, since the second round of the competition. Before MQOM, the GeMSS signature scheme [13] was an alternative candidate in the third round of the first NIST post-quantum competition, and it also exposed a public key consisting Boolean quadratic polynomials.

Algorithms for Boolean MQ. There is a wealth of algorithms dedicated to solving Boolean quadratic systems [32,10,2,23,27,8,17,16,12], along with successful implementations [30], even in hardware [11]. Discussing all of them goes beyond the scope of this paper, so we will content ourselves with a few remarks. Exhaustive search solves Boolean MQ in time $\tilde{\mathcal{O}}\left(2^n\right)$, and it has been known for more than 20 years that the problem could be solved in $2^{\alpha n}$ operations, with $\alpha < 1$, by combining exhaustive search and algebraic technique, under algebraic assumptions on the genericity of the input system (it has to be "sufficiently unstructured") [32,1]. The best embodiment of this line of work is the BooleanSolve algorithm of Bardet, Faugère and Spaenlehauer [2], with expected running time $\tilde{\mathcal{O}}\left(2^{0.792n}\right)$. However, none of the sophisticated algorithms could compete with efficiently implemented exhaustive search [10] in practice, and the crossover seemed quite far ([2] estimated it at $n = 200$, without providing evidence). The "Crossbred" algorithm of Joux and Vitse [23] became the first algorithm to "beat brute force" in practice, for sizes as small as $n = 40$. Its asymptotic complexity has not been precisely established, but its practical efficiency is spectacular. It holds the current computational records for both overdetermined (type-I) and underdetermined (type-IV) Boolean systems in the Fukuoka MQ challenges collection [33] (`mqchallenge.com`). A completely different family of algorithms emerged when Lokshtanov, Paturi, Tamaki, Williams and Yu [27] presented a randomized algorithm of complexity $\tilde{\mathcal{O}}\left(2^{0.8765n}\right)$ based on the "polynomial method". In strong contrast with almost all the previous ones, it does not require any assumption on the input polynomials, which is a theoretical breakthrough. The algorithm works by assembling a high-degree polynomial that evaluates to 1 on partial solutions, then approximates it by lower-degree polynomials. The technique was later improved by Björklund, Kaski and Williams [8], reaching $\tilde{\mathcal{O}}\left(2^{0.804n}\right)$, then again by Dinur [17], reaching $\tilde{\mathcal{O}}\left(2^{0.6943n}\right)$. What has

been known as "Dinur's first algorithm" (e.g. [17]) presently offers the best known asymptotic complexity. However, noting that the self-reduction that results in this low asymptotic complexity only kicks in for very large values of n, Dinur proposed a simpler, trimmed down version of his algorithm for the crypto community with complexity $\mathcal{O}\left(n^2 2^{0.815n}\right)$ using $n^2 2^{0.63n}$ bits of memory [16]. This is "Dinur's second algorithm".

However, the algorithms with the best asymptotic complexity are not always the most efficient in practice, and the actual number of operations required to run them cannot be read transparently from their asymptotic complexity. Designers of public-key schemes have to make sure that the instances they expose are concretely hard enough. There is in particular a tension between the low asymptotic complexity of Dinur's first algorithm and the fact that it almost always require a higher number of actual binary operations than all the other one for relevant sizes. For this reason, several software tools provide estimates of the concrete number of bit operations required to execute these algorithms, notably the `MQEstimator` [3] that is available in the `CryptographicEstimators` software library [18].

Concretely, this means that the parameter sets of MQOM over the Boolean field have been chosen to offer the desired security levels against all known algorithms. NIST's requires that the best known attack requires at least $2^{143}, 2^{207}$ and 2^{272} "classical gates" for security levels I, III and V respectively, which correspond to the estimated cost of a brute-force attack against AES-128, AES-192 and AES-256.

Correlated GGM trees and partial guessing one-way problems. MQOM uses the technique of "correlated GGM trees" introduced by Guo, Yang, Wang, Zhang, Xie, Zhang and Liu [22] to yield shorter signatures. In [19], Feneuil and Rivain show that the security of the resulting signature scheme relies both on the hardness of MQ and of an additional problem, "Partial-Guessing One-Way MQ" (PGOW-MQ).

In short, a PGOW-MQ adversary receives a quadratic system supposed to offer λ bits of security, and they have to find the first λ bits of a solution (i.e. a prefix of an actual solution). For any $\hat{y} \in \{0,1\}^\lambda$ we say that $\hat{y}$ is a *partial solution* of the system if there is a $\hat{z} \in \{0,1\}^{n-\lambda}$ such that the concatenation $(\hat{y}, \hat{z})$ is a solution of the system. So, a PGOW-MQ adversary has to find a λ-bit partial solution. They also have access to an oracle that tests partial solution.

More precisely, the (Boolean) PGOW-MQ problem is the following:

- The adversary receives a Boolean quadratic system E composed of m equations in n variables, and an integer λ.
- The adversary has access to an oracle. The adversary may submit candidates $\hat{y} \in \{0,1\}^\lambda$ to the oracle. The oracle returns 1 if the submitted candidate is partial solution of E.
- The adversary wins if it eventually returns a λ-bit partial solution of E.

– The adversary (T, Q, ϵ)-solves the PQOW-MQ problem if it runs in time T, submits at most q queries to the oracle and returns a partial solution of the system with probability at least ϵ.

In the context of MQOM, when a single signature made using correlated trees is available, testing a candidate prefix (i.e. implementing the functionality offered by the oracle) can be done for the cost of a single evaluation of the AES. At security levels I, III and V, where $\lambda = 128, 192$ and 256, NIST estimates that evaluating the AES requires $2^{15}, 2^{15}$ and 2^{16} bit operations, respectively. It follows that an adversary requiring significantly less than $2^{\lambda+15}$ bit operations and making significantly less than 2^{λ} queries to the oracle would break MQOM.

In the second round of the competition, the designers of MQOM implicitly assumed that PGOW-MQ is as hard as MQ. And indeed, when the first λ bit of a solution are provided, finding the remaining unknowns is *practical*: it requires solving a Boolean system of m equations in $(n - \lambda)$ unknowns, where $(m, n - \lambda)$ is $(160, 32), (240, 48)$ or $(320, 64)$ for security levels I, III and V, respectively. Random systems of these sizes are easy to solve: the `hpXbred`[1] software, an open-source "high-performance" implementation of the Crossbred algorithm, solves them in 0.1s, 1s and 750s, respectively, using a single core of the author's laptop. The estimated workload is $2^{37}, 2^{40}$ and 2^{53} bit operations, respectively.

So, could there be an algorithm that solves PGOW-MQ faster than MQ itself can be solved? This is not obvious, for at least two reasons:

1. The oracle is only useful if the adversary finds a candidate *partial* solution $\hat{y}$. Indeed, if the adversary has a candidate *full* solution $\hat{x}$, they could just check if $E(\hat{x}) = 0$ by evaluating the polynomials, without using the oracle. Finding candidate partial solutions better than by random chance seems hard.
2. Given a candidate partial solution, the adversary could try to "implement the oracle" themselves, namely testing if $\hat{y}$ is a partial solution by solving $E(\hat{y}, z) = 0$ (for instance using `hpXbred` as outlined above). So, if the adversary is capable of isolating a *small* number of candidate partial solution, then they can certainly afford this extra cost.

So, at first glance, it seems that in most cases, what the oracles does, the adversary could do by themselves. Only if the adversary produces a *large* list of candidate partial solutions is the oracle useful to sieve them.

Our contributions. We present a new algorithm for Boolean MQ, that improve only marginally upon the state of the art. However, minor modifications enable it to solve PGOW-MQ by actually exploiting the availability of the oracle. In the context of MQOM, where an oracle query costs one AES evaluation, then it requires a bit less operations to retrieve the secret key from one signature by solving PGOW-MQ than by solving the instance of MQ exposed by the public key. This provides the first evidence that the two problems do not have the exact

[1] `https://gitlab.lip6.fr/almasty/hpXbred`

Security level	I	III	V
λ	128	192	256
n	160	240	320
Expected hardness	143	207	272
Crossbred	144	213	282
Dinur 2nd	142	208	274
This work (MQ)			
$\log_2 T$	141	205	270
$\log_2 M$	108	162	213
This work (PGOW-MQ)			
$\log_2 T$	140	203	269
$\log_2 M$	106	159	210

Table 1. Estimates of number of bit operations required to break the Boolean instances exposed by MQOM, either by attacking the public key alone (MQ), or by exploiting the availability of a single signature (PGOW-MQ). Numbers have been rounded to the nearest integer. Estimates for the cost of the Crossbred are taken from [4]. Estimates for the cost of Dinur's second algorithm do not match those given in [4] because the cost model is different. T and M give the number of "classical gates" and bits of memory required to run the new algorithm, respectively.

same hardness (even though the difference is quite small). Table 1 summarizes the results.

The proposed algorithm is a hybrid between the algorithm of Lokshtanov et al. [27] and Dinur's second algorithm [16]. While the latter asymptotically improves upon the former, we in fact revert some of the modifications. We suggest to call our modest proposal "Dinur's 1.9-th algorithm", to highlight our decremental contribution in terms of sophistication level. The proposed modifications nevertheless also decrease a bit the concrete number of operations required by the algorithm. It results in, to the best of our knowledge, to the smallest number of bit operations required to solve quadratic Boolean systems of relevant cryptographic sizes. However, just like the two algorithms it is based on, its huge memory requirements make it complement impractical and there is little point in attempting an implementation.

We do not give an asymptotic expression of the complexity of the algorithm, and we only analyze it under (plausible) heuristic assumptions. However, we try to derive concrete estimates of the required number of operations to run it, with a view towards assessing the hardness of the chosen MQOM parameter sets.

In passing, we discuss techniques to perform a linear change of variables that puts some arbitrary quadratic polynomials in "UOV shape". Besides improving a bit the performance of our algorithm, they are potentially of more broadly applicable, and of independent interest. In particular, they improve upon the Thomae-Wolf algorithm [31] and the Furue-Nakamura-Takagi algorithm [20] to

solve mildly underdetermined Boolean quadratic systems. As an independent contribution, we discuss their application to solve systems with $n = 1.5m$ variables (as in the Fukuoka MQ challenges) or $n = 3m$ (as in the original UOV scheme [24]).

Consequences for MQOM. Even though the estimated number of operations to break MQOM drops 3 to 4 bits below NIST's expectations, we believe that the proposed algorithms have no impact on the security of MQOM, for two reasons (beyond the fact that 3 bits may fall within the error margin of any estimate).

Large memory requirements. There is no consensus in the cryptologic community regarding the cost of accessing a large memory, which may further depend on the actual access pattern of the algorithm. This debate likely cannot be settled by experiments: it seems difficult to say anything about the "practical cost" (on machines that do not exist) of a computation whose parameters were chosen precisely to make it impossible. However, assuming that a machine large enough to execute our algorithms (e.g. capable of holding 2^{100} bits of storage) could be built seems akin to assuming that a machine large enough to break the AES by exhaustive search in a reasonable amount of time could also be built: replace the hardware that stores bits by hardware that evaluates the AES (this point was also made by Bernstein [5]). Further note that, at levels III and V, the proposed algorithm needs space comparable to or larger than one bit per atom of the Earth (including the core).

Simplified non-uniform computational model. Because the proposed algorithms are unimplementable, we analyze their number of operations in a simple computational model that ignores all the complications of the real world. Following [16], we consider a Boolean circuit model represented by straight-line programs:

> *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).*

In other terms, a statement such as:

$$A[2015494782137237151] \leftarrow A[8910305899308506505] \oplus A[14034715819129815024]$$

counts as a single bit operation.

Clearly, just *producing* the straight-line program in the first place requires more work than simply *executing* it. Switching to a computational model where the adversary executes a compact program written in a usual programming language would likely drive the computational cost up by an unknown factor, because the adversary would then have to generate the circuit "on the fly". Bouillaguet observed in [9] that when implementing Dinur's memory-efficient Möbius transform (a crucial component of his second algorithm), one "bit operation"

required an average of 20 CPU cycles to actually compute the indices of the "memory addresses" in the straight-line program. On the basis of this observation, the designers of MQOM applied a $20n$ multiplicative penalty to the cost of this specific operation when estimating the cost of Dinur's second algorithm.

In the straight-line program model, the program may also contain an "advice" (of size even greater than its running time) in the hard-coded memory addresses. The computational model is thus non-uniform, which gives extra artificial power to the adversaries. Bernstein and Lange [6] have shown that the AES could be generically broken in such a model if the envelope is pushed too far.

Here, for the sake of simpler comparison with [16], we stick to the simplified and unrealistic computational model. This is why Table 1 shows an estimate for Dinur's second algorithm that is below the NIST expected security level: we do not apply the penalty incurred by the more realistic model as the designers of MQOM did.

It follows that, as in [16], we give an estimate of the number of Boolean operations performed by an exponentially large circuit. This may not reflect accurately the actual difficulty of running an implementation of the algorithm on existing hardware. But it does not really matter, because it is impossible to actually run it (with non-toy inputs) on existing hardware in the first place.

2 Preliminaries and notations

The ring of Boolean polynomials in n variables $x = (x_1, \ldots, x_n)$ is the quotient of the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_n]$ by the ideal spanned by the "field equations" $\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$. In other terms, in a Boolean polynomial, variables appear with exponent at most one. We borrow from [16] the convenient notation $\binom{n}{\downarrow d}$ to represent the partial sum $\sum_{k=0}^{d} \binom{n}{k}$. A Boolean polynomial of degree d in n variables has at most $\binom{n}{\downarrow d}$ terms. Indeed, there are $\binom{n}{k}$ Boolean monomials of degree k: they are the product of k distinct variables.

All arithmetic takes place in the field with two elements. We often write $\hat{x}$ to emphasize that $\hat{x}$ is a vector with coordinates in $\{0, 1\}$, as opposed to a vector of formal variables. We write $(e_1, \ldots, e_n)$ the canonical basis of the vector space $\mathbb{F}_2^n$.

A polynomial system E is simply a vector of Boolean polynomials. Given a system $E = (f_1, \ldots, f_m)$, we write $E(\hat{x})$ to denote the parallel evaluation of the polynomials of E on $\hat{x}$, namely the vector $(f_1(\hat{x}), \ldots, f_m(\hat{x})) \in \{0, 1\}^m$. We often split vectors in two parts: if $\hat{y} \in \{0, 1\}^{n-k}$ and $\hat{z} \in \{0, 1\}^k$, then we write $(\hat{y}, \hat{z}) \in \{0, 1\}^n$ for their concatenation. A vector $\hat{x} \in \{0, 1\}^n$ is a solution of a polynomial system E if $E(\hat{x}) = (0, \ldots, 0)$. If a system admits a solution $(\hat{y}, \hat{z})$ we say that $\hat{y}$ a *partial solution*, and that $\hat{z}$ is a *completion* of $\hat{y}$ (it "completes" the partial solution).

A quadratic polynomial can be represented as $f(x) = x^\mathsf{T} Q x + c$, where Q is an $n \times n$ Boolean matrix and Q_{st} $(s \neq t)$ is the coefficient of $x_s x_t$ and the diagonal Q_{ss} holds the coefficient of the linear term x_s (this works because $x_s^2 = x_s$). A linear change of variables $x = Sx'$ acts by congruence: $f(Sx') = x'^\mathsf{T}(S^\mathsf{T} Q S)x' + c_i$, the diagonal of $S^\mathsf{T} Q_i S$ being reinterpreted as linear terms.

The *polar form* of f is the symmetric bilinear form

$$b(x, x') = f(x + x') + f(x) + f(x') + f(0),$$

whose matrix is $B = Q + Q^{\mathsf{T}}$: linear and constant parts cancel. Its diagonal is zero, so b is *alternating*: $b(x, x) = 0$ for every x. It is *nondegenerate* when B is invertible, which requires n to be even, the rank of an alternating form being even. The *orthogonal* of a subspace W with respect to b is

$$W^{\perp} = \{x : b(x, w) = 0 \text{ for all } w \in W\},$$

and $\dim W^{\perp} = n - \dim W$ when b is nondegenerate. Finally, a subspace W is *isotropic* for b when $b(x, x') = 0$ for all $x, x' \in W$ (equivalently, if $W \subseteq W^{\perp}$).

3 The new algorithm

The input is a system $E = (f_1, \ldots, f_m)$ of m Boolean quadratic polynomials in n variables, that we assume are drawn uniformly at random. The goal is to find a solution of E. The pseudo-code is given by Algorithm 1 and the text of this section explains some of its details. The algorithm has four parameters: k (size of the suffix), ℓ (size of the compressed subsystems), τ (number of repetitions), and σ (detection threshold).

Split the n variables into a prefix $y = (y_1, \ldots, y_{n-k})$ and a suffix $z = (z_1, \ldots, z_k)$. As discussed in the introduction, given a partial solution $\hat{y}$ of E, a complete solution can be found by solving the overdetermined system $E(\hat{y}, z) = 0$ for z, as this is much easier than the original problem. The algorithm works by finding candidate partial solutions $\hat{y}$, and checking if each one can be extended into a complete solution $(\hat{y}, \hat{z})$.

Recursion. To solve E, the algorithm in fact solves an exponentially large number of smaller quadratic systems. This can be done by any sub-algorithm that returns all solutions of the given system — this seems to exclude the present one, as it only return a given solution with some probability. The "base case" solver can be either exhaustive search (e.g. the FES algorithm [10,9]) or the Crossbred algorithm [23]. The latter is more efficient for large systems.

Input randomization. The algorithm randomizes the input system τ times as follows. To sample a random "compressed system" $\tilde{E}^{(i)} = (\tilde{f}_1^{(i)}, \ldots, \tilde{f}_\ell^{(i)})$ for $0 \leq i < \tau$, pick ℓ random linear combinations of the original polynomials. In other terms, pick a uniformly random matrix $A^{(i)}$ of dimension $m \times \ell$ and set $\tilde{E}^{(i)} \leftarrow E \times A^{(i)}$. All solutions of E are solutions of $\tilde{E}^{(i)}$, but the converse is not true. Because ℓ will always be much smaller than n, the resulting compressed systems are underdetermined and have $2^{n-\ell}$ solutions on average, whereas the input system is expected to have only a constant number of solutions. So $\tilde{E}^{(i)}$ has a lot of "parasitic" solutions in addition to the original ones.

Algorithm 1 The proposed algorithm.

procedure DINUR-1.9(E, k, ℓ, τ, σ)
  Set $w \leftarrow 2\ell - k$ and $N \leftarrow \binom{n-k}{\downarrow w}$
  Allocate a bi-dimensional array G of dimension $\tau \times N$
  for $0 \leq i < \tau$ **do**
    Sample a random matrix $A^{(i)}$ of dimension $m \times \ell$ and set $\tilde{E}^{(i)} \leftarrow E \times A^{(i)}$
    for $\hat{y} \in \{0,1\}^{n-k}$ of Hamming weight at most w **do**
      Solve $\tilde{E}_i(\hat{y}, z) = 0$ ▷ *ℓ equations, k variables*
      Set $\mathtt{G}[i][\hat{y}]$ to the parity of the number of solutions
    INTERPOLATE($n-k, w, \mathtt{G}[i]$)
  for $\hat{y} \in \{0,1\}^{n-k}$ **do**
    $c_{\hat{y}} \leftarrow 0$ ▷ *integer counter*
    for $0 \leq i < \tau$ **do**
      if EVALUATE($n-k, w, \mathtt{G}[i], \hat{y}) = 1$ **then**
        Increment $c_{\hat{y}}$
    if $c_{\hat{y}} \geq \sigma$ **then**
      Solve $E(\hat{y}, z) = 0$ ▷ *m equations, k variables*
      Emit any solution $(\hat{y}, \hat{z})$ that has been found

Define:

$$F^{(i)}(x) = (1 + \tilde{f}_1^{(i)}(x))(1 + \tilde{f}_2^{(i)}(x)) \cdots (1 + \tilde{f}_\ell^{(i)}(x)),$$
$$G^{(i)}(y) = \sum_{\hat{z} \in \{0,1\}^k} F^{(i)}(y, \hat{z}),$$

so that $F^{(i)}(\hat{x}) = 1$ if and only if $\hat{x}$ is a solution of $\tilde{E}^{(i)}$, and $G^{(i)}(\hat{y})$ is the parity of the number of solutions of $\tilde{E}^{(i)}$ with prefix $\hat{y}$. In particular $G^{(i)}(\hat{y}) = 1$ implies that $\hat{y}$ is a partial solution of $\tilde{E}^{(i)}$.

The main idea of the algorithm is to use the $G^{(i)}$'s as approximate oracles to detect partial solutions of the original system E. However, this approximation has two-sided error, and this is why the process is repeated τ times to amplify precision: if $G^{(i)}(\hat{y}) = 1$ for at least σ out of τ iterations, then $\hat{y}$ is a candidate partial solution of the original system E with sufficiently high probability, and we try to solve $E(\hat{y}, z) = 0$. The values of τ and σ are chosen such that an actual solution is reported with probability at least $1/2$, and the number of tested candidates is not too high.

Computing the coefficients of $G^{(i)}$. The polynomial $F^{(i)}$ has n variables and degree 2ℓ, whereas $G^{(i)}$ has $n-k$ variables and degree at most $2\ell - k$, under the assumption that this quantity is non-negative (this is a standard result, a proof of which is given in [16]). This implies that ℓ cannot be too small, but in our case, the optimal value of ℓ is generally greater than or equal to k, and this condition is always satisfied.

Write $w = 2\ell - k$ for the degree of $G^{(i)}$ and $N = \binom{n-k}{\downarrow w}$ for the number of monomials of degree at most w in $n-k$ variables. Note that $G^{(i)}$ has N

potentially non-zero coefficients and can be represented by an array of N bits. The algorithm computes the coefficients of $G^{(i)}$ as follows:

- For each $\hat{y} \in \{0,1\}^{n-k}$ of Hamming weight at most w, solve the quadratic system $\tilde{E}^{(i)}(\hat{y}, z) = 0$ and record the parity of its number of solutions. This gives the value of $G^{(i)}(\hat{y})$.
- Once the values of G (a degree-w polynomial) are known on all input vectors of weight at most w, then the coefficients of its algebraic normal form can be interpolated. Such a procedure, inspired by the Möbius transform, is described in [16, §2.3]. It runs in time $\mathcal{O}((n-k)N)$. We denote this procedure by INTERPOLATE.

Evaluating $G^{(i)}$ on all possible inputs. Once the coefficients of $G^{(i)}$ have been obtained, then $G^{(i)}$ can be evaluated on arbitrary input points. We denote by EVALUATE the corresponding procedure that takes as arguments the coefficients of a polynomial P, an evaluation point $\hat{x}$, and returns $P(\hat{x})$. In Algorithm 1, $G^{(i)}$ is in fact evaluated on all $\hat{y} \in \{0,1\}^{n-k}$, and this makes it possible to use efficient evaluation strategies. To simplify the analysis, we consider that this is done thanks to the memory-efficient Möbius transform described by Dinur in [16], that runs in $w2^{n-k}$ bit operations (hence w per evaluation on average) and uses negligible memory in addition to the size of its input.

Memory requirements. The algorithm needs τN bits of storage for the $G^{(i)}$'s, plus any additional space required by the "base-case" quadratic solver and by the memory-efficient Möbius transform, which we neglect.

Time complexity. When a given partial solution $\hat{y} \in \{0,1\}^{n-k}$ is "suggested" by at least σ out of τ iterations, the algorithm tries to solve the original system $E(\hat{y}, z) = 0$ with the proper substitution. Denote by C the number of tested candidates. Then the total number of bit operations required to run the algorithm is:

$$\tau \cdot \left(N \cdot T_{MQ}(\ell, k) + (n-k)N + w2^{n-k}\right) + C \cdot T_{MQ}(m, k), \tag{1}$$

where $T_{MQ}(\ell, k)$ denote the running time of the "base case" solver that finds all solutions of a typical system of ℓ quadratic Boolean equations in k variables. The four terms correspond respectively to solving the subsystems (step 7), interpolating the $G^{(i)}$'s (step 9), evaluating the $G^{(i)}$'s on all possible inputs (step 13) and testing candidate solutions (step 16).

It remains to determine the expected value of C and the success probability of the algorithm to complete its analysis. This is the object of section 4.

Comparison with Dinur's second algorithm. The main difference with [16] is that the latter, in addition to $G^{(i)}$, computes k extra polynomials in $n-k$ variables of degree $w+1$. These polynomials directly reveal a completion of the candidate partial solution $\hat{y}$ when $G^{(i)}(\hat{y}) = 1$. The candidates are therefore

not partial, but full solutions, and it suffices to evaluate the input polynomials on them to check their validity. The algorithm of [16] does only 4 iterations on average, and tests candidate solutions when they have been suggested by at least two iterations.

Compared to Algorithm 1, this has two main effects:

- N increases from $\binom{n-k}{\downarrow w}$ in Algorithm 1 to $\binom{n-k}{\downarrow w+1}$ in [16].
- Instead of evaluating a single degree-w polynomial on all $\hat{y} \in \{0,1\}^{n-k}$, [16] has to evaluate k of them (and they have degree $w+1$) in each iteration.
- The time spent testing partial solutions (in step 16) does not exist in [16].
- The number of iterations of [16] is 4 on expectation, and τ (often of order $k/2$) in Algorithm 1.

Assuming that $\ell \approx k$, we have $\binom{n-k}{\downarrow w+1} \approx \left(\frac{n}{k} - 2\right)\binom{n-k}{\downarrow w}$. So with $k \approx n/5$ as is typical, this is an increase of about $\times 2.5$. Evaluating all the high-degree polynomials on all $\hat{y}$ will cost about k times more in [16]. However, the downside of Algorithm 1 is that it has to do more iterations to obtain a sufficiently precise "classifier" in step 15. Anticipating on the sequel, the optimal value of τ is often close to $k/2$. Assuming $\tau \approx k/2$, this implies that, compared to [16], in Algorithm 1:

- The time spent in step 7 is multiplied by $\approx k/20$.
- The time spent in step 13 is divided by ≈ 8.

So the two algorithm reach slightly different compromises.

4 Heuristic analysis of Algorithm 1

Let us fix a vector $(\hat{y}^\star, \hat{z}^\star)$ and let E be a random system such that $E(\hat{y}^\star, \hat{z}^\star) = 0$. Further assume that all the $A^{(i)}$ matrices sampled in step 5 of the algorithm have full column rank (this fails to happen with probability less than $\tau 2^{\ell-m}$). We analyze the probability that the algorithm returns the "planted" solution $(\hat{y}^\star, \hat{z}^\star)$ and the time it needs to do so.

On step 15, the algorithm classifies candidate prefixes $\hat{y}$ as "likely" when the counter $c_{\hat{y}}$ is above σ, a fixed threshold. The intuition behind this procedure is that $G^{(i)}(\hat{y}) = 1$ should happen more often (over the τ iterations) when $\hat{y}$ is a partial solution of E, and in particular for $\hat{y}^\star$.

Our goal is therefore to estimate $\Pr[c_{\hat{y}^\star} \geq \sigma]$ (the probability to find the planted solution) and $\Pr[c_{\hat{y}} \geq \sigma \mid \hat{y} \neq \hat{y}^\star]$ (probability to test an arbitrary prefix). All of this section therefore revolves around the distribution of the counters $c_{\hat{y}}$.

Effect of the randomization. In each iteration, a randomized compressed system $\tilde{E}^{(i)}$ is drawn. When the input system is random, these systems are also random. This follows Lemma 2 below, with $A = A^{(i)}$.

Lemma 1 ([16, Propositions A.6 and A.7]). *Let A be a fixed $m \times \ell$ matrix that has full column rank, and let $\tilde{E} = E \times A$. If E is a random quadratic system of m polynomials in n variables, then the quadratic system $\tilde{E}(\hat{y}^\star, z)$ is a uniformly random system of ℓ quadratic equations in the k variables z, vanishing on $\hat{z}^\star$. If $\hat{y} \neq \hat{y}^\star$, then $\tilde{E}(\hat{y}, z)$ is a uniform quadratic system.*

Proof. Let $\mathcal{H}_j$ denote the $\mathbb{F}_2$-vector space of Boolean polynomials of degree at most 2 in j variables, and let $\mathcal{H}_n^0 = \{f \in \mathcal{H}_n : f(\hat{y}^\star, \hat{z}^\star) = 0\}$ and $\mathcal{H}_k^0 = \{g \in \mathcal{H}_k : g(\hat{z}^\star) = 0\}$. These are hyperplanes, since evaluation at a point is a nonzero linear functional (it maps the constant polynomial 1 to 1). The planted input system E is uniformly distributed over $(\mathcal{H}_n^0)^m$. We use the following elementary fact: if X is uniformly distributed over a finite $\mathbb{F}_2$-vector space V and $\phi : V \to W$ is a surjective linear map, then $\phi(X)$ is uniformly distributed over W, because all the fibers of ϕ have the same cardinality $|\ker \phi|$.

For any prefix $\hat{y}$, substituting $y \leftarrow \hat{y}$ commutes with taking linear combinations of polynomials, so $\tilde{E}(\hat{y}, z) = E(\hat{y}, z) \times A$. In other terms, $\tilde{E}(\hat{y}, z) = \Phi_{\hat{y}}(E)$, where $\Phi_{\hat{y}}$ is the linear map

$$\Phi_{\hat{y}} : \mathcal{H}_n^m \longrightarrow \mathcal{H}_k^\ell, \qquad (f_1, \ldots, f_m) \longmapsto \big(f_1(\hat{y}, \cdot), \ldots, f_m(\hat{y}, \cdot)\big) \times A,$$

namely the coordinate-wise substitution $\rho_{\hat{y}} : f \mapsto f(\hat{y}, \cdot)$ followed by right-multiplication by A. Because A has full column rank, there is a matrix $B \in \mathbb{F}_2^{\ell \times m}$ such that $BA = I_\ell$, and any target $T \in \mathcal{H}_k^\ell$ is the image of $TB \in \mathcal{H}_k^m$ under right-multiplication by A. Note that the entries of TB are linear combinations of the entries of T, so they belong to any subspace of $\mathcal{H}_k$ that contains the entries of T.

First take $\hat{y} = \hat{y}^\star$. The restriction of $\Phi_{\hat{y}^\star}$ to $(\mathcal{H}_n^0)^m$ takes its values in $(\mathcal{H}_k^0)^\ell$, since each $f_s(\hat{y}^\star, \cdot)$ vanishes at $\hat{z}^\star$ and vanishing at $\hat{z}^\star$ is preserved by linear combinations. This restriction is surjective onto $(\mathcal{H}_k^0)^\ell$. Indeed, $\rho_{\hat{y}^\star}$ maps $\mathcal{H}_n^0$ onto $\mathcal{H}_k^0$: any $g \in \mathcal{H}_k^0$, seen as a polynomial in (y, z) that does not involve y, vanishes at $(\hat{y}^\star, \hat{z}^\star)$ and is its own preimage; then, given a target $T \in (\mathcal{H}_k^0)^\ell$, its preimage TB lies in $(\mathcal{H}_k^0)^m$. It follows that $\tilde{E}(\hat{y}^\star, z)$ is uniform over $(\mathcal{H}_k^0)^\ell$, i.e., it is a uniformly random system of ℓ quadratic equations in k variables vanishing at $\hat{z}^\star$.

Now suppose $\hat{y} \neq \hat{y}^\star$. This time, $\rho_{\hat{y}}$ maps $\mathcal{H}_n^0$ onto the whole of $\mathcal{H}_k$: pick a coordinate j such that $\hat{y}_j \neq \hat{y}_j^\star$; then any $g \in \mathcal{H}_k$ is the image of $f = g + g(\hat{z}^\star) \cdot (y_j + \hat{y}_j)$, which has degree at most 2, satisfies $f(\hat{y}, \cdot) = g$, and vanishes at $(\hat{y}^\star, \hat{z}^\star)$. Together with the matrix B as before, the restriction of $\Phi_{\hat{y}}$ to $(\mathcal{H}_n^0)^m$ is surjective onto $\mathcal{H}_k^\ell$, and $\tilde{E}(\hat{y}, z)$ is uniform over $\mathcal{H}_k^\ell$. □

Isolation. Completions of $\hat{y}$ in $\tilde{E}^{(i)}$ come in two kinds. Either $(\hat{y}, \hat{z})$ is a solution of the original system E and we call the completion *genuine*, or it is not and we call it *parasitic*: it was introduced by the compression process. The distinction matters because genuine completions survive compression in every single iteration (solutions of E are solutions of all the $\tilde{E}^{(i)}$'s), whereas parasitic ones are resampled with each $A^{(i)}$.

Write $s(E, \hat{y})$ for the number of genuine completions of $\hat{y}$; this quantity is frozen once E is drawn (it does not depend on the random coins of the algorithm), and it is shared by all τ iterations. For a fixed input system E, write P_i for the number of parasitic completions of $\hat{y}$ in $\tilde{E}^{(i)}$. This is a random variable over the choice of $A^{(i)}$, and $G^{(i)}(\hat{y})$ is the parity of $s(E, \hat{y}) + P_i$. The τ repetitions are meant to amplify the "signal" $s(E, \hat{y})$ while the "noise" P_i will be smoothed out — this works because P_i is only mildly dependent on the actual choice of E. However, this can only work if there is a signal to amplify in the first place, namely if $s(E, \hat{y})$ is odd.

In other terms, the classifier of step 15 does not detect partial solutions of E *per se*: it only detects prefixes with an odd number of genuine completions. A prefix with exactly two completions in E is statistically indistinguishable from a prefix with none just by looking at the parities given by $G^{(i)}$. Fortunately, for the planted prefix, the favorable situation is overwhelmingly likely.

Lemma 2 (Given informally in [16]). *The probability that there is $\hat{z} \neq \hat{z}^\star$ such that $E(\hat{y}^\star, \hat{z}) = 0$ is at most 2^{k-m}.*

Proof. By Lemma 1 applied with $\ell = m$ and $A = I_m$, the system $E(\hat{y}^\star, z)$ is a uniformly random system of m quadratic equations in the k variables z vanishing at $\hat{z}^\star$. For any fixed $\hat{z} \neq \hat{z}^\star$, evaluation at $\hat{z}$ is a surjective linear functional on the space of quadratic polynomials vanishing at $\hat{z}^\star$ (there are affine polynomials vanishing at $\hat{z}^\star$ but not at $\hat{z}$), hence $\Pr[E(\hat{y}^\star, \hat{z}) = 0] = 2^{-m}$. By linearity of expectation, the expected number of such $\hat{z}$ is $(2^k - 1)\, 2^{-m}$, and the conclusion follows from Markov's inequality. □

When $\hat{z}^\star$ is the only suffix that completes $\hat{y}^\star$, in other terms when $s(E, \hat{y}^\star) = 1$, we say (following Dinur) that the partial solution $\hat{y}^\star$ is *isolated*.

Law of the counters. Fix the input system E and a prefix $\hat{y}$. The random coins of the algorithm, namely the choice of the $A^{(i)}$ matrices, are the only remaining randomness. They are independent and identically distributed. Define

$$p(E, \hat{y}) := \Pr_{A^{(i)}}\left[G^{(i)}(\hat{y}) = 1\right]$$

and note that it does not actually depend on i because the $A^{(i)}$'s are identically distributed. The events "$G^{(i)}(\hat{y}) = 1$" are determined by the choice of the $A^{(i)}$'s, and therefore they are independent Bernoulli trials. As a consequence, $c_{\hat{y}}$ follows a binomial law of parameters $(\tau, p(E, \hat{y}))$.

For the algorithm to work, $p(E, \hat{y})$ has to be higher when $\hat{y}$ is a partial solution of E than when it is not, and we need this to hold for almost all input systems. If E is randomly sampled, then $p(E, \hat{y})$ is itself a random variable and $c_{\hat{y}}$ is distributed like a *mixture of binomials* (i.e. a binomial distribution where the number of trials is fixed but the common success probability of the trials has itself been randomly chosen). The marginal tail of the counter, averaged over all possible inputs, which is what we really need, is:

$$\Pr_{E, A^{(i)}}\left[c_{\hat{y}} \geq \sigma\right] = \mathop{\mathbb{E}}_{E}\left[\Pr\left[\mathrm{Bin}\big(\tau, p(E, \hat{y})\big) \geq \sigma\right]\right]. \tag{2}$$

Average behavior of the counters. We first focus on the average behavior of the counters for a typical input, across a single iteration. Define:

$$q(\hat{y}) := \Pr_{E,A^{(i)}}\left[G^{(i)}(\hat{y}) = 1\right] = \mathop{\mathbb{E}}_{E}[p(E,\hat{y})].$$

It follows from Lemma 1 that q has a relatively tame behavior, and takes only two values: $q(\hat{y}^\star)$ on the one hand and $q(\hat{y})$ for $\hat{y} \neq \hat{y}^\star$ on the other, regardless of the actual choice of $\hat{y}$. Indeed, let us consider a uniformly random system of ℓ quadratic equations in k variables $R = (R_1, \ldots, R_\ell)$ and let Z be the random variable that counts its solutions in $\{0,1\}^k$. It follows from Lemma 1 that

$$\begin{aligned} \alpha := q(\hat{y}^\star) &= \Pr\left[Z \text{ odd} \mid R(\hat{z}^\star) = 0\right] \\ \beta := q(\hat{y}) \ &= \Pr\left[Z \text{ odd}\right] \qquad\qquad (\hat{y} \neq \hat{y}^\star) \end{aligned}$$

Both α and β are functions of k and ℓ only. To obtain simple closed-form expressions, we make the following heuristic assumption about the number of solutions of overdetermined systems:

Assumption 1 *For $k \leq \ell$, the number Z of solutions of a uniformly random system of ℓ quadratic equations in k variables is $Z \sim \text{Poisson}(2^{k-\ell})$. For systems conditioned to vanish on a fixed point, $Z \sim 1 + \text{Poisson}(2^{k-\ell})$.*

Bach and Fusco have shown that for random systems, the first part of this assumption is in fact a theorem when taken asymptotically (i.e. when $n \to +\infty$) [21]. They report on experiments showing that this is additionally fairly precise even for small values of n.

Parities are conveniently extracted from the Poisson model. If $Z = s + P$ where s is a fixed integer and $P \sim \text{Poisson}(\lambda)$, then on the one hand $\mathbb{E}\left[(-1)^Z\right] = (-1)^s e^{-\lambda} \sum_i (-\lambda)^i / i! = (-1)^s e^{-2\lambda}$, while on the other hand $\mathbb{E}\left[(-1)^Z\right] = \Pr[Z \text{ even}] - \Pr[Z \text{ odd}]$; since the two probabilities sum to 1, we obtain:

$$\Pr[Z \text{ odd}] = \frac{1 - (-1)^s e^{-2\lambda}}{2}.$$

Assumption 1 corresponds to the cases $s = 0$ (unconditioned systems) and $s = 1$ (systems vanishing at a fixed point) with $\lambda = 2^{k-\ell}$, and therefore yields simple closed forms for α and β:

$$\alpha = \frac{1 + e^{-2^{1+k-\ell}}}{2}, \qquad \beta = \frac{1 - e^{-2^{1+k-\ell}}}{2}. \tag{3}$$

This reveals the expected values of the counters by linearity of expectation: $\mathbb{E}[c_{\hat{y}^\star}] = \tau\alpha$ and $\mathbb{E}[c_{\hat{y}}] = \tau\beta$ when $\hat{y} \neq \hat{y}^\star$ (the expectations are taken both over the randomness of E and the random coins of the algorithm). This is not yet enough to control the success probability of the "classifier" that decides if a prefix is likely a partial solution. Indeed, the right-hand side of 2 is determined by the whole distribution of $p(E, \hat{y})$, not by its mean alone.

Approximate distribution of the counters. We therefore make another heuristic assumption that replaces the mixture of binomials governing the counters by its two dominant branches:

Assumption 2 *If $\hat{y}^\star$ is isolated, then $c_{\hat{y}^\star} \sim \mathrm{Bin}(\tau, \alpha)$. If $\hat{y}$ has no genuine completion, then $c_{\hat{y}} \sim \mathrm{Bin}(\tau, \beta)$.*

This amounts to say that $p(E, \hat{y})$ is entirely determined by the parity of $s(E, \hat{y})$ and does not otherwise fluctuate with E. Alternatively, we assume that $p(E, \hat{y})$ is so concentrated around its mean $q(\hat{y})$, over the random choice of E, that we can replace the actual value by its expectation in (2), which then degenerates into a usual binomial tail bound. Of course, in reality $p(E, \hat{y})$ actually fluctuates a little when E ranges over all possible input systems. The assumption says that this fluctuation is small enough so that it only introduces negligible error in the tail bound.

In [16], Dinur proves *pairwise* independence of the events "$G^{(i)}(\hat{y}) = 1$", which is enough to compute the variance of the counters and bound the error probability of the classifier using Chebyshev's inequality (but the resulting bound is loose). Our model yields the correct mean and, up to negligible error, the correct variance; Assumption 2 is that the agreement extends to the tail behavior, which second-moment information alone cannot certify.

Conclusion. We now have everything we need to complete the analysis of the algorithm.

Lemma 3. *Under Assumption 2, the algorithm emits the planted solution $(\hat{y}^\star, \hat{z}^\star)$ with probability at least $\Pr[\mathrm{Bin}(\tau, \alpha) \geq \sigma] - 2^{k-m} - \tau 2^{\ell-m}$ (taken over both the random choice of the input and the random coins of the algorithm).*

Proof. By Lemma 2, the planted solution fails to be isolated with probability at most 2^{k-m} over the random choice of E. When the planted solution is isolated, then by Assumption 2,$c_{\hat{y}^\star} \sim \mathrm{Bin}(\tau, \alpha)$.

The analysis requires that $A^{(i)}$ has full rank ℓ, and this happens with probability $\prod_{i<\ell}(1 - 2^{i-m}) \geq 1 - 2^{\ell-m}$. A union bound over the τ iterations therefore leads to a worst-case drop in success probability of at most $\tau 2^{\ell-m}$.

Lemma 4. *Under Assumption 2, the expected number of tested candidate partial solutions is bounded by:*

$$\mathbb{E}[C] \leq 2^{n-k} \Pr[\mathrm{Bin}(\tau, \beta) \geq \sigma] + 2^{n-m} + 1.$$

Proof. By linearity of expectation, $\mathbb{E}[C] = \sum_{\hat{y}} \Pr[c_{\hat{y}} \geq \sigma]$. By Assumption 2, each of the (at most 2^{n-k}) prefixes $\hat{y}$ with no genuine completion contributes $\Pr[\mathrm{Bin}(\tau, \beta) \geq \sigma]$ to the sum. Every remaining prefix admits at least one genuine completion, so their number is at most the number of solutions of E, whose expectation is $1 + (2^n - 1)2^{-m}$ (the planted solution counts, and each of the $2^n - 1$ other candidates are solutions with probability 2^{-m}). Each contributes at most 1 to the sum.

So the success probability and expected running time have been completely determined under Assumptions 1 and 2.

5 Parameter search and concrete estimates

With n and m given, the four parameters k, ℓ, τ, σ can be chosen by exhaustive search to minimize the cost given in (1) (where C is given by Lemma 4), under the condition that the success probability given by Lemma 3 is at least $1/2$. Our estimator is provided as supplementary material.

To obtain actual numbers, this requires estimations of the number of operations in the "base-case" quadratic solvers used in steps 7 and 16. We use the following heuristic, which is well-matched by our own experiment: if the number of variables is greater than 40, or greater than 32 when the systems are overdetermined, we use the Crossbred algorithm. Otherwise, we use the FES algorithm. In both case, we use our own estimators (but off-the-shelf ones, such as those available in the `CryptographicEstimators` library could be used too).

Several observations may help speed up the search. First, σ need not be searched at all. The number of tested candidates is a decreasing function of σ, so the highest possible value should be used to minimize the cost. The success probability is also a decreasing function of σ, and this fixes the highest admissible value.

Second, it is more convenient to explore the "strictness" of the filter than the number of iterations itself. Suppose we want $C \leq 2^{n-k-x}$ for some $x \geq 0$. In practice, we search $x = 0, 0.2, 0.4, \ldots$ and we find the $(\tau; \sigma)$ pair that achieve this goal while offering the target success probability. The values of τ obtained are increasing with x, and we stop the process when this makes the cost blow-up past the best known parameter set.

Third, τ can be determined without enumerating all possible values. We always have $\mathbb{E}[C] \leq 2^{n-k}\beta^{\tau}$ (attained at $\sigma = \tau$), so $\tau \geq x/\log_2(1/\beta)$ is a floor; from there τ is doubled until feasible and the last interval bisected.

Finally, if not imposed by the outside, we search k and ℓ exhaustively. The outer loops run k from $n/7$ to $n/2$ and ℓ from $k+1$ to $k+10$, the lower limit encoding both $k \leq 2\ell - 2$ and $k \leq \ell$.

Table 2 shows concrete parameters and bit-operation estimates for Algorithm 1 and Dinur's second algorithm from [16] on random systems of n equations in n variables, with sizes corresponding to MQOM parameters. We used our own estimator for Dinur's algorithm, and checked that it was consistent with the figures given in [16]. As argued above, the improvement offered by the new algorithm is modest. However, Algorithm 1 is somewhat flexible, and allows a greater range of tradeoffs, as we will now illustrate by applying it to the PGOW-MQ problem.

n	160	240	320
Algorithm 1			
k	28	45	61
ℓ	30	47	63
τ	18	21	31
σ	15	17	25
$\log_2 N$	102.4	155.2	207
$\log_2 T_{MQ}(\ell, k)$	32.8	44.7	58.4
$\log_2 E[C]$	105.6	166.5	218
$\log_2 T_{MQ}(m, k)$	32.8	37.3	50.8
$\log_2 M$	106.6	159.6	211.9
$\log_2 T$	141.7	206.0	271.4
Algorithm of [16]			
n_1	30	45	50
$\log_2 T$	142.3	208.4	274.1

Table 2. Estimates of number of bit operations required to run Algorithm 1 and Dinur's "second algorithm" from [16] on random systems of n equations in n variables. M denote the space complexity, in bits, and T the total number of "classical gates".

6 Application to Partial-Guessing One-Way MQ

Algorithm 1 can exploit the oracle to test candidate partial solutions in step 16. To obtain a PGOW-MQ solver, set $k = n - \lambda$ (this is imposed by the PGOW-MQ setting) and modify step 16 to query the oracle on $\hat{y}$ instead of solving $E(\hat{y}, z) = 0$, and return $\hat{y}$ if the oracle answers 1.

In the context of MQOM, "invoking the oracle" costs one AES, which is cheaper than solving the subsystem. This leads to different tradeoffs and parameters than presented above to solve MQ. Table 3 shows the parameters for Algorithm 1 used as a PGOW-MQ solver for the instances exhibited by MQOM.

In the PGOW-MQ setting, we are forced to use a sub-optimal value of k. Also, compared to the MQ setting, τ and σ change: the "classifier" is indeed much more permissive, and checks a fraction comprised between 1/4 and 1/8 of all possible partial solutions.

Just doing an exhaustive search on 1/8 of all possible $\hat{y}$ would result in an algorithm with a comparable running time, but success probability 1/8. Using Algorithm 1 boosts the success probability to 1/2. This provides concrete evidence that PGOW-MQ can be marginally easier than MQ, by a factor of about 2, at least in the Boolean case.

7 Solving multiple correlated MQ instances

Dinur's second algorithm [16] and Algorithm 1 have a common feature: they both need to solve $\tilde{E}^{(i)}(\hat{y}, z) = 0$ repeatedly, for a large number of different

n	160	240	320
λ	128	192	256
$\log_2 T_{AES}$	15	15	16
ℓ	32	48	64
τ	8	15	15
σ	5	9	9
$\log_2 N$	100.8	152.4	204.1
$\log_2 T_{MQ}(\ell, k)$	36.9	47.2	61.2
$\log_2 E[C]$	125.9	189.2	253.2
$\log_2 M$	103.8	156.3	208
$\log_2 T$	141.8	205.1	270.3

Table 3. Estimates of number of bit operations required to solve PGOW-MQ using Algorithm 1.

$\hat{y} \in \{0,1\}^{n-k}$. In addition, Algorithm 1 also needs to solve $E(\hat{y}, z) = 0$ for many $\hat{y}$'s. Recognizing this situation leads to several opportunities to make the process more efficient.

We show how to find an invertible linear change of variables S such that after setting $E'(x) \leftarrow E(Sx)$, some of the resulting polynomials $E'(y, z)$ contain no quadratic monomial in the z variables (i.e. no $z_s z_t$). Polynomials of this specific shape notably appear in the secret key of the UOV signature scheme [24], and therefore we will say that these polynomial without quadratic monomials in the z's are in "UOV shape". Our goal is to find a change of variables that puts the largest number of arbitrary polynomials in UOV shape simultaneously. *A contrario*, the security of UOV relies of the assumption that applying a random invertible linear change of variables to a collection of polynomials in UOV shape effectively obfuscates their structure and make them appear random.

The interest of this maneuver is that if we can find a linear invertible change of variable that puts r polynomials in UOV shape, then $E'(\hat{y}, z)$ contains r linear equations in z. This allows (except with negligible probability) to eliminate r variables from $E'(\hat{y}, z)$. This can only make the resulting system easier to solve.

We present two techniques to achieve this goal. They were in fact readily available in the existing literature, but were presented with a different goal. These algorithms rest on the following observation: making polynomials affine in z is the same thing as exhibiting a common isotropic subspace.

Lemma 5. *Let S be an invertible matrix and let W be the subspace spanned by its last k columns. Let f denote a quadratic polynomial, b the corresponding polar form, and set $f' \leftarrow f(Sx)$. Then $f'(y, z)$ contains no quadratic monomial $z_s z_t$ if and only if W is isotropic for b.*

Proof. Write $f(x) = x^\mathsf{T} Q x + c$, and let $B = Q + Q^\mathsf{T}$ denote the matrix representation of its polar form. Let $w_1, \ldots, w_k$ denote the last k columns of S, so that $W = \langle w_1, \ldots, w_k \rangle$; under the split $x = (y, z)$, the variable z_s multiplies the column w_s in the substitution $x \leftarrow Sx$.

We have $f'(x) = x^\mathsf{T}(S^\mathsf{T}QS)\,x + c$. For $s \neq t$, the monomial $z_s z_t$ collects the two entries of $S^\mathsf{T}QS$ indexed by the (unordered) pair $\{n-k+s,\ n-k+t\}$, so its coefficient in f' is

$$w_s^\mathsf{T} Q\, w_t + w_t^\mathsf{T} Q\, w_s = w_s^\mathsf{T}(Q + Q^\mathsf{T})\, w_t = b(w_s, w_t),$$

where the first equality holds because $w_t^\mathsf{T} Q\, w_s$, being a 1×1 matrix, equals its transpose $w_s^\mathsf{T} Q^\mathsf{T} w_t$. (The diagonal entries $w_s^\mathsf{T} Q\, w_s$ of the bottom-right block only contribute to the linear terms, through $z_s^2 = z_s$.)

Therefore $f'(y, z)$ contains no quadratic monomial $z_s z_t$ if and only if $b(w_s, w_t) = 0$ for all $1 \leq s < t \leq k$. Since b is alternating, $b(w_s, w_s) = 0$ holds unconditionally, so this is equivalent to the vanishing of b on all pairs of basis vectors of W, hence, by bilinearity, to W being isotropic for b. □

7.1 The greedy method

This is a straightforward adaptation of the technique introduced by Courtois, Goubin, Meier and Tacier to solve very underdetermined quadratic systems in polynomial time [14]. The applicability of the core idea was later expanded by Thomae and Wolf [31] as well as Furue, Nakamura and Takagi [20]. The result is summarized as:

Theorem 1 (derived from [14]). *A change of variables putting $r = \left\lfloor \frac{n}{k} \right\rfloor - 1$ arbitrary polynomials in UOV shape can be computed with $\mathcal{O}\left(n^3\right)$ bit operations.*

Algorithm 2 Greedy construction of a common isotropic subspace.

Require: alternating forms $B_1, \ldots, B_r$ over $\mathbb{F}_2^n$
Ensure: a basis $w_1, \ldots, w_d$ of a subspace isotropic for all of them, with $d = \lceil n/(r+1) \rceil$

$V \leftarrow \{e_1, \ldots, e_n\}$
for $i \leftarrow 1$ **to** d **do**
  $w_i \leftarrow$ any element of V
  $V \leftarrow V \setminus \{w_i\}$
  for $j \leftarrow 1$ **to** r **do**
    $c \leftarrow B_j w_i$
    if $c^\mathsf{T} v = 1$ for at least one $v \in V$ **then** ▷ *(Gaussian elimination)*
      $v_0 \leftarrow$ any such v
      $v \leftarrow v + v_0$ for every other $v \in V$ such that $c^\mathsf{T} v = 1$
      $V \leftarrow V \setminus \{v_0\}$
return $(w_1, \ldots, w_d)$

The theorem follows from the correctness of Algorithm 2: it returns a basis of a common isotropic subspace for r arbitrary polar forms, of dimension $\lceil n/(r+1) \rceil$. Because $r = \lfloor n/k \rfloor - 1$, we have $r + 1 \leq n/k$ and obviously $k \leq n/(r+1)$. Therefore any subset of k vectors returned by the algorithm span a subspace of dimension k that is still isotropic, and Lemma 5 concludes.

The following invariant is maintained throughout Algorithm 2: at the beginning of iteration i of the outer loop, the family $w_1, \ldots, w_{i-1}$ together with V is linearly independent, and

$$b_j(w_\ell, v) = 0 \qquad \text{for every } j \leq r, \text{ every } \ell < i \text{ and every } v \in V.$$

Both parts of the invariant hold vacuously at the beginning of the first iteration when $i = 1$. The transfer of an element from V to $\{w_1, \ldots, w_i\}$ on steps 3–4 does not change linear independence; step 9 adds one element of V to others and step 10 remove elements: such operations preserve the linear independence of the whole family, and they preserve every orthogonality relation already acquired. It remains to see that the inner loop installs the relations involving the newly selected w_i: once the constraint $c = B_j w_i$ has been processed, every v still in V satisfies $b_j(w_i, v) = c^\mathsf{T} v = 0$, and the later steps, being of the two kinds just described, leave that untouched.

When the algorithm terminates, the w_ℓ are linearly independent by the first part of the invariant. For $\ell < i$ we have $b_j(w_\ell, w_i) = 0$ by the second part, since w_i belonged to V when it was chosen; and $b_j(w_i, w_i) = 0$ because the forms are alternating. The subspace $\langle w_1, \ldots, w_d \rangle$ is therefore isotropic for every b_j.

Each iteration of the outer loop removes at most $r+1$ elements, so V is never empty and step 3 cannot fail.

One execution of steps 6–10 computes a matrix-vector product, evaluates $|V|$ scalar products and updates at most $|V|$ vectors, for $\mathcal{O}\left(n^2\right)$ bit operations. It is executed $dr \leq n + r$ times, whence the bound $\mathcal{O}\left(n^3\right)$.

7.2 The pencil method

The greedy method works well when k is small compared to n: it allows to control more polynomials simultaneously. In the limit case where $k = \frac{n}{2}$, Theorem 1 essentially restates the classical result that any Boolean non-degenerate quadratic form can be put into one of the following shapes:

$$\begin{aligned}
&x_1x_2 + x_3x_4 + \cdots + x_{n-2}x_{n_1} + x_n && n \text{ odd} \\
&x_1x_2 + x_3x_4 + \cdots + x_{n-1}x_n && n \text{ even} \\
&x_1x_2 + x_3x_4 + \cdots + x_{n-1}x_n + x_{n-1} + x_n && n \text{ even}
\end{aligned}$$

after which fixing the variables with odd index gives a linear form. See [26] for more details.

However, it is possible to do better in this case, using techniques reminiscent of the simultaneous diagonalization of two quadratic forms. The specific technique exploited here can be traced to the cryptanalysis of the original Oil and Vinegar signature scheme by Kipnis and Shamir [25]. Similar ideas were later used by Macario-Rat, Plût and Gilbert to give an algorithm provably solving the Isomorphism of Polynomial with One secret (IP1S) problem in polynomial time [28]. Building on the theory of matrices pencils (i.e., the family of matrices

Algorithm 3 Pencil-based construction of a common Lagrangian.

Require: two polar forms b_i, b_j of E, the matrix B_j being invertible
Ensure: a subspace of dimension $n/2$ isotropic for b_i and b_j

$M \leftarrow B_j^{-1} B_i$
$W \leftarrow 0$
$P \leftarrow \mathbb{F}_2^n$ ▷ $P = W^\perp$
while $W \neq P$ **do**
  $w \leftarrow$ any element of $P \setminus W$
  repeat
    $W \leftarrow W + \langle w \rangle$
    $P \leftarrow P \cap \langle w \rangle^\perp$
    $w \leftarrow Mw$
  until $w \in W$
return W

$A + \lambda B$, indexed by the scalar λ), they prove that every regular pencil of alternating forms over a binary field is congruent to a pencil of block-anti-diagonal matrices. This is in fact stronger than what we need: it shows that is possible to compute in polynomial time a linear invertible change of variable that makes two quadratic forms simultaneously bilinear over a binary field. In our context, this can be simplified a little to yield:

Theorem 2 (derived from [25,28]). *For even n and $k \leq \frac{n}{2}$, a change of variables putting $r = 2$ polynomials in UOV shape can be computed with $\mathcal{O}\left(n^3\right)$ bit operations, when one of the two has a non-degenerate quadratic component.*

Algorithm 3 shows the promised pseudo-code. Let b_i and b_j denote two polar forms, with respective matrices B_i and B_j, and B_j is invertible (this requires n to be even). We let $M = B_j^{-1} B_i$. For a vector x we write $\mathbb{F}_2[M]\, x = \langle x, Mx, M^2x, \dots \rangle$ for the *Krylov subspace* generated by x under M, that is, the smallest M-stable subspace containing x. The inner loop of Algorithm 3 is the usual Krylov iteration: it extends the sequence $x, Mx, M^2x, \dots$ for as long as the next iterate is linearly independent from the previous ones. We begin with two simple results.

Lemma 6. *$M^\mathsf{T} B_j = B_j M = B_i$, and $M^\mathsf{T} B_i = B_i M$. Consequently $p(M)^\mathsf{T} B = B\, p(M)$ for every polynomial $p \in \mathbb{F}_2[X]$ and both $B \in \{B_i, B_j\}$.*

Proof. $B_j M = B_i$ by definition, and $M^\mathsf{T} B_j = B_i^\mathsf{T} B_j^{-\mathsf{T}} B_j = B_i^\mathsf{T} = B_i$ since B_i is symmetric. Then $M^\mathsf{T} B_i = M^\mathsf{T} B_j M = B_i M$. The statement for $p(M)$ follows by additivity and induction. □

Lemma 7. *For every polynomial $p \in \mathbb{F}_2[X]$, every $x \in \mathbb{F}_2^n$, and both $B \in \{B_i, B_j\}$, we have $x^\mathsf{T} B p(M) x = 0$. Consequently, every Krylov subspace $\mathbb{F}_2[M]\, x$ is isotropic for b_i and for b_j simultaneously.*

Proof. By additivity it suffices to treat $p = X^e$, and by Lemma 6 half the powers of M can be moved to the left argument. If $e = 2s$ then $x^\mathsf{T} B_j M^{2s} x =$

$(M^s x)^\mathsf{T} B_j (M^s x) = 0$ because b_j is alternating. If $e = 2s{+}1$ then $x^\mathsf{T} B_j M^{2s+1} x = (M^s x)^\mathsf{T} B_j M (M^s x) = (M^s x)^\mathsf{T} B_i (M^s x) = 0$ because b_i is alternating. The case $B = B_i$ reduces to the previous one, as $x^\mathsf{T} B_i M^e x = x^\mathsf{T} B_j M^{e+1} x$.

For the second claim, let $p, q \in \mathbb{F}_2[X]$; using Lemma 6 again, $b(p(M)x, q(M)x) = x^\mathsf{T} p(M)^\mathsf{T} B\, q(M) x = x^\mathsf{T} B\, (pq)(M) x = 0$. □

In the sequel, all the orthogonals are taken for b_j (the non-degenerate form). The algorithm maintains the invariant $P = W^\perp$, which steps 7 and 8 preserve jointly, since $\big(W + \langle w \rangle\big)^\perp = W^\perp \cap \langle w \rangle^\perp$. We argue below that W stays isotropic, that is $W \subseteq P$; since $\dim P = n - \dim W$, the outer loop stops exactly when W has dimension $n/2$, and while it runs $P \setminus W$ is not empty so step 5 draws w from a non-empty set. We now argue that Algorithm 3 terminates and returns a subspace of dimension $n/2$ that is isotropic for b_i and for b_j.

Two properties are maintained by the outer loop: W is stable under M, and W is isotropic for b_i and for b_j (so that $W \subseteq P$).

For stability, let w be the vector chosen at step 5 during a turn. The vectors added to W during that turn are $w, Mw, \ldots, M^{e-1}w$, with $M^e w \in W$; the subspace acquired is therefore the Krylov subspace $\mathbb{F}_2[M]\, w$, which is M-stable, and so W is a sum of M-stable subspaces.

Isotropy is where the choice made at step 5 is used. That Krylov subspace is isotropic for both forms by Lemma 7, so only the cross terms between the hold W and the newly acquired Krylov space have to be examined. Let u belong to W as it was at the beginning of the turn, and let $p \in \mathbb{F}_2[X]$. By Lemma 6, $b_j\big(u, p(M)w\big) = b_j\big(p(M)u, w\big)$, which vanishes because $p(M)u \in W$ by stability and because w was taken in $P = W^\perp$; and $b_i\big(u, p(M)w\big) = b_j\big(Mu, p(M)w\big)$ vanishes for the same reason.

We finish by explaining how to implement the algorithm with the announced complexity. The subspace W is stored as an echelon basis, namely a family of vectors with, for each of them, a distinguished coordinate that the others have equal to 0. The subspace P is stored as a basis split in two: that of W, and that of a complement of W in P. Both dimensions are kept as counters.

Step 4 is then a comparison of two integers and step 5 returns any vector of the second part, both in constant time. Step 10 reduces w against the echelon basis of W and step 7 inserts the result into it, each in $\mathcal{O}(n)$ operations on vectors of $\mathbb{F}_2^n$. Step 8 forms $B_j w$, evaluates the linear form $x \mapsto (B_j w)^\mathsf{T} x$ on the basis of P and eliminates one of the vectors on which it does not vanish, exactly as in steps 8–10 of Algorithm 2; one further elimination, this time of the coordinate form at the pivot of w, restores the split of the basis of P. This is again $\mathcal{O}(n)$ vector operations.

Each vector added to W therefore costs $\mathcal{O}\left(n^2\right)$ bit operations, and there are at most $n/2$ of them; inverting B_j at step 1 costs $\mathcal{O}\left(n^3\right)$, which dominates.

8 Application of the greedy method to Algorithm 1

We show how to improve algorithm 1 using linear change of variables that put polynomials in UOV shape. The same applies, *mutatis mutandis*, to Dinur's second algorithm.

- First, using the greedy method (Theorem 1), compute the linear change of variable S that puts as many polynomials in UOV shape as possible (because $k \approx n/5$, then $r = 4$ UOV-shaped polynomials should generally be within reach). Compute the modified system $E'(x) = E(Sx)$. Denote by E'_{UOV} the r polynomials in UOV-shape.
- Run Algorithm 1 on E', with the following two alterations:
  - Forming a compressed system $\tilde{E}^{(i)}$ works by taking $\ell - r$ random linear combinations of polynomials in E' and adding (without modifications) the r polynomials in E'_{UOV}.
  - When solving a subsystem in steps 7 and 16, exploit the r linear equations to eliminate r variables before solving the resulting system
- At the end, if Algorithm 1 returns a solution $\hat{x}$, then return $S^{-1}\hat{x}$.

Because of the modification in the sampling of the compressed systems $\tilde{E}^{(i)}$, the previous analysis no longer applies as-is. For instance, $\tilde{E}^{(i)}(\hat{y}, z)$ cannot be uniformly random, because it has r UOV-shaped polynomials. However, it is plausible that, after the r linear equations have been used to eliminate r variables, the result appears uniformly random.

Note that Dinur proposed a heuristic modification of a comparable nature, by suggesting to take random subsets of E to form the $\tilde{E}^{(i)}$, instead of random linear combinations.

Tables 4 and 5 show the revised estimates and parameter choice when applied to the MQ and PGOW-MQ instances exposed by MQOM, respectively. For MQ, it yields a speed-up of about $\times 2$, while for PGOW-MQ it yields an acceleration of $\times 4$. It queries the AES on a fraction roughly $1/64$ of the whole codebook.

9 Application of the pencil method to underdetermined MQ systems

In order to solve an underdetermined system of m equations in $n > m$ variables, the Thomae-Wolf algorithm [31] applies Theorem 1 with $k = m$ to obtain $r := \lfloor \frac{n}{m} \rfloor - 1$ linear equations after $n - m$ variables are fixed to arbitrary values. The r linear equations can then be used to eliminate r variables, leading to a subsystem of $m - r$ equations in $m - r$ variables. Therefore, if all systems were solved by exhaustive search, this reduce the hardness of the initial problem by r bits.

Fixing $n-m$ variables to arbitrary values makes the initially underdetermined system "square", with one solution on average. The Furue-Nakamura-Takagi algorithm [20] goes a step further by fixing $n - m + g$ variables, with $g \geq 0$. So,

n	160	240	320
Algorithm 1			
k	29	47	62
ℓ	31	49	64
τ	16	18	28
σ	13	15	23
$\log_2 N$	103.6	157.5	208.1
$\log_2 T_{MQ}(\ell - r, k - r)$	29.6	42.7	56.1
$\log_2 E[C]$	108.8	166.6	218
$\log_2 T_{MQ}(m - r, k - r)$	29.6	35.3	47.8
$\log_2 M$	107.6	161.6	212.9
$\log_2 T$	140.6	204.9	270.1

Table 4. Estimates of number of bit operations required to run Algorithm 1 to solve MQ combined with putting 4 polynomials in UOV shape. T and M denote the number of "classical gates" and the number of bits of memory required to run the computation, respectively.

n	160	240	320
λ	128	192	256
$\log_2 T_{AES}$	15	15	16
ℓ	33	49	65
τ	8	8	8
σ	6	6	6
$\log_2 N$	100.8	155.5	207.3
$\log_2 T_{MQ}(\ell - r, k - r)$	32.8	43.7	57.9
$\log_2 E[C]$	121.9	185.9	249.9
$\log_2 M$	106.0	158.5	210.3
$\log_2 T$	140.0	203.0	268.5

Table 5. Estimates of number of bit operations required to break MQOM by solving a PGOW-MQ using Algorithm 1 combined with putting 4 polynomials in UOV shape. T and M denote the number of "classical gates" and the number of bits of memory required to run the computation, respectively.

in addition to making the system square, g extra variables are "guessed". This yields an overdetermined system of m equations in $m-g$ variables, that has a solution with probability 2^{-g} (heuristically). Therefore the process will have to be repeated 2^g times on average before a solution is found. The point is that Theorem 1 can be applied with $k = m-g$, leading to $r := \lfloor \frac{n}{m-g} \rfloor - 1$ linear equations. This again enables to eliminate r variables, and reduce the total workload by 2^r. Increasing g increases r, and thus decreases the total work factor.

Pushing to the limit, and guessing $g = m - \sqrt{n}$ variables leaves a system of m equations, of which $\approx \sqrt{n}$ are linear, in $\sqrt{n}$ variables. This leads to a simple algorithm that solves instances of MQ in time about $n^{1.5}2^{m-\sqrt{n}}$. This is a simple case of the "just guess" algorithm from May, Ostuzzi and Ressler [29]. In its full form, it reaches a complexity of $\tilde{\mathcal{O}}\left(2^{m-\sqrt{2n-m}}\right)$. This nicely complements another comparably simple idea given by Bouillaguet, Delaplace and Trimoska in [12], that runs in time $n^{1.5}2^{n-\sqrt{2m}}$ for overdetermined systems.

Simple as it is, this is asymptotically inferior to the algorithms that run in time $\mathcal{O}(2^{\alpha n})$ with $\alpha < 1$. However, the Furue-Nakamura-Takagi can be combined with an arbitrary algorithm with running time $T(m,n)$. The number g of "guessed" variables then becomes an additional parameter, and the problem consists in finding the minimum of

$$2^g \cdot T\left(m - \left\lfloor \frac{n}{m-g} \right\rfloor + 1, m - g - \left\lfloor \frac{n}{m-g} \right\rfloor + 1\right).$$

Increasing g means many small overdetermined systems instead of a few big ones.

We know show how the pencil method (Theorem 2) can lead to improvements. We discuss two relevant use-cases: underdetermined Boolean Fukuoka MQ challenges and the original UOV signature scheme over $\mathbb{F}_2$.

9.1 Fukuoka MQ challenges with $n = 1.5m$

The *Fukuoka MQ challenge* [33] (`mqchallenge.org`) is a collection of random MQ instances of various sizes over different field. Type-I and Type-IV challenges are Boolean, and type-IV are underdetermined, consisting of m equations and $n = 1.5m$ variables. This matches the public-key sizes exposed by the Rainbow signature scheme [15]. Rainbow has since been broken by Beulens [7]. The largest solved type-IV instance has 76 equations in 114 variables. It was solved by the `hpXbred` software, and the computation made use of the pencil method, as described below.

Starting from the input system E, use Theorem 2 with $k = n/2$ to find a change of variable S putting two polynomials in UOV shape, resulting in $E'(x) := E(Sx)$. Then, repeat these steps until success:

1. Pick a random $\hat{y} \in \{0,1\}^{0.75m}$, yielding the subsystem $E'(\hat{y}, z) = 0$
2. Use the two linear linear polynomials in z to eliminate two variables the other $m-2$ equations.

3. Solve the resulting (overdetermined) system of $m-2$ equations in $0.75m-2$ variables using the Crossbred algorithm.
4. If it has a solution $\hat{z}$, return $S^{-1}(\hat{y}, \hat{z})$ — this is a solution of the input system E. Otherwise, go back to step 1.

The subsystem produced in each iteration is overdetermined, and is expected to have a solution with probability $2^{-m/4}$, so about $2^{m/4}$ iterations can be expected. They are independent and can be carried out in parallel, making the computation "embarrassingly parallel". In the actual computation with $m = 76$, 512K subsystems of 74 equations in 55 variables were solved in parallel using a single Intel Xeon Gold 6248 core per system. Each trial required 16640s on expectation, for an expected running time of 2.4M CPU·hours. This was estimated to be less than fixing 38 variables randomly and attacking the remaining "square" system of 76 equations in 76 variables using the Crossbred algorithm directly.

Here, $g = 0.25m$ variables have been "guessed" beyond what is necessary to make the system "square", which means that $2^{0.25m}$ iterations are necessary on average. These extra guesses enables the use of the pencil method and leads to subsystems of $m-2$ equations in $0.75m-2$ variables.

In the same setting, the Thomae-Wolf algorithm simply does not apply. The Furue-Nakamura-Takagi algorithm would need to guess $0.5m$ variables to obtain the same two linear equations. This would have resulted in 256G subsystems of 74 equations in 38 variables, a fairly less practical scenario.

9.2 The UOV signature scheme over $\mathbb{F}_2$ with $n = 3m$

In the case of the original UOV scheme [24], the public key is a system of m quadratic equations in $n = 3m$ variables. In this case, it is possible to apply the pencil method twice: applying it once with $k = 1.5m$, then fixing the first $1.5m$ variables to random values and exploiting the two resulting linear equations yields a system of $m-2$ equations in $1.5m-2$ variables. Applying it again with $k = 0.75m-1$, fixing k additional variable to random values and exploiting the two new linear equations yields $m-4$ quadratic equations in $0.75m-3$ variables. The resulting system is overdetermined, therefore $2^{0.25m-1}$ iterations are expected before a solution can be found. If all the subsystems were solved by brute-force, the workload would be an exhaustive search over 2^{m-4} candidates.

In the same setting, the Thomae-Wolf algorithm produces a system of $m-2$ equations in $m-2$ variables after fixing $2m$ variables to a random value, so the "brute-force workload" would be 2^{m-2}. The Furue-Nakamura-Takagi algorithm would need to guess $g = 0.4m$ variables to obtain 4 linear equations, leading to $2^{0.4m}$ systems of $m-4$ equations in $0.6m-4$ variables.

Once again, the pencil method enables dealing with less bigger subproblems, which is always easier.

References

1. Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Pierre and Marie Curie University, Paris, France, 2004.
2. Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *J. Complexity*, 29(1):53–75, 2013.
3. Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, and Javier A. Verbel. An estimator for the hardness of the MQ problem. In Lejla Batina and Joan Daemen, editors, *Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings*, volume 13503 of *Lecture Notes in Computer Science*, pages 323–347. Springer Nature Switzerland, 2022.
4. Ryad Benadjila, Charles Bouillaguet, Thibauld Feneuil, and Matthieu Rivain. Mqom: Mq on my mind. Algorithm Specifications and Supporting Documentation (Version 2.1), 2025.
5. Daniel J. Bernstein. Understanding brute force, 2005. Available online: `http://cr.yp.to/papers.html#bruteforce`.
6. Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2013.
7. Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2022.
8. Andreas Björklund, Petteri Kaski, and Ryan Williams. Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
9. Charles Bouillaguet. Algorithm 1052: Evaluating a boolean polynomial on all possible inputs. *ACM Trans. Math. Softw.*, 50(4), December 2024.
10. Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.
11. Charles Bouillaguet, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. Fast Exhaustive Search for Quadratic Systems in $\mathbb{F}_2$ on FPGAs. In *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2013. `https://eprint.iacr.org/2013/436.pdf`.
12. Charles Bouillaguet, Claire Delaplace, and Monika Trimoska. A simple deterministic algorithm for systems of quadratic polynomials over f_2. In Karl Bringmann and Timothy M. Chan, editors, *5th Symposium on Simplicity in Algorithms,*

SOSA@SODA 2022, Virtual Conference, January 10-11, 2022, pages 285–296. SIAM, 2022.

13. Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A Great Multivariate Short Signature. Research report, UPMC - Paris 6 Sorbonne Universités ; INRIA Paris Research Centre, MAMBA Team, F-75012, Paris, France ; LIP6 - Laboratoire d'Informatique de Paris 6, December 2017.
14. Nicolas T. Courtois, Louis Goubin, Willi Meier, and Jean-Daniel Tacier. Solving underdefined systems of multivariate quadratic equations. In *Public Key Cryptography, PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 211–227. Springer, 2002.
15. Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175, 2005.
16. Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021.
17. Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021.
18. Andre Esser, Javier A. Verbel, Floyd Zweydinger, and Emanuele Bellini. Sok: Cryptographicestimators - a software library for cryptographic hardness estimation. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*. ACM, 2024.
19. Thibauld Feneuil and Matthieu Rivain. On the security of MPC-in-the-head signatures with correlated GGM trees. Cryptology ePrint Archive, Paper 2026/615, 2026.
20. Hiroki Furue, Shuhei Nakamura, and Tsuyoshi Takagi. Improving thomae-wolf algorithm for solving underdetermined multivariate quadratic polynomial problem. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2021.
21. Giordano Fusco and Eric Bach. Phase transition of multivariate polynomial systems. *Math. Struct. Comput. Sci.*, 19(1):9–23, 2009.
22. Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 330–362. Springer, 2023.

23. Antoine Joux and Vanessa Vitse. A Crossbred Algorithm for Solving Boolean Polynomial Systems. In *NuTMiC*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017. `https://eprint.iacr.org/2017/372.pdf`.
24. Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999.
25. Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil & vinegar signature scheme. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 257–266. Springer, 1998.
26. Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 2 edition, 1994.
27. Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017.
28. Gilles Macario-Rat, Jérôme Plut, and Henri Gilbert. New insight into the isomorphism of polynomial problem ip1s and its use in cryptography. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 117–133, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
29. Alexander May, Massimo Ostuzzi, and Henrik Ressler. Just guess: Improved (quantum) algorithm for the underdetermined MQ problem. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part IV*, volume 16544 of *Lecture Notes in Computer Science*, pages 334–362. Springer, 2026.
30. Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang. Implementing joux-vitse's crossbred algorithm for solving MQ systems over GF(2) on gpus. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 121–141. Springer, 2018.
31. Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2012.
32. Bo-Yin Yang and Jiun-Ming Chen. Theoretical analysis of XL over small fields. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2004.
33. Takanori Yasuda, Xavier Dahan, Yun-Ju Huang, Tsuyoshi Takagi, and Kouichi Sakurai. A multivariate quadratic challenge toward post-quantum generation cryptography. *ACM Commun. Comput. Algebra*, 49(3):105–107, 2015.