Results on polynomial interpolation with mixed modular operations and unknown moduli

Oscar Garcia-Morchon[1], Ronald Rietman[1], Igor E. Shparlinski[2] and Ludo Tolhuizen[1]

[1] Philips Group Innovation, Research, Eindhoven, The Netherlands
[2] Department of Pure Mathematics, University of New South Wales, Sydney, Australia

Abstract. Motivated by a recently introduced HIMMO key predistribution scheme, we investigate the limits of various attacks on the polynomial interpolation problem with mixed modular operations and hidden moduli. We firstly review the classical attack and consider it in a quantum-setting. Then, we introduce new techniques for finding out the secret moduli and consider quantum speed-ups.

Keywords Key pre-distribution scheme, polynomial interpolation, mixed modular operations, hidden moduli

1 Introduction

In a key pre-distribution scheme [7], a trusted third party (TTP) provides each node in a system with a function that enables it to obtain a symmetric key with any other node. Blundo *et al.* [1] introduced an elegant and efficient key predistribution scheme based on symmetric bi-variate polynomials over a finite field. Key pre-distribution schemes are prone to *collusion attacks*: various nodes may collude to obtain the key between a pair of (non-colluding) nodes. For example, if the degree of the bi-variate polynomial in the scheme of [1] is α, then $\alpha+1$ or more colluding nodes can find the functions of each other node in the system by simple interpolation, thus breaking this scheme completely.

The HIMMO scheme [2,3] is an efficient key predistribution scheme with strong collusion resistance. The security analysis in [2] shows that HIMMO leads in a natural way to lattice problems of which the solution will not significantly benefit from the the potential future introduction of quantum computers [6]. Furthermore, HIMMO has very low bandwidth, computation, and memory requirements. These facts make HIMMO an interesting quantum-safe alternative.

We define $\langle x\rangle_q$ as $x - \lfloor \frac{x}{q} \rfloor q$, which for positive q is the smallest non-negative residue of x on division by q.

In the present paper, we further study the following problem, which forms the basis for one of the collusion attacks on HIMMO from [2]:

MMO problem Let b, B, α, N and $m \geq 2$ be given integers, where N is odd and has bit length $(\alpha+1)B+b$. Let $q_1, \ldots, q_m$ be m integers of the form $q_i = N - \beta_i 2^b$, where $0 < \beta_i < 2^B$. Let $h : \mathbb{Z} \to \mathbb{Z}_N$ be given by

$$h(x) = \Big\langle \sum_{i=1}^{m} \langle f_i(x) \rangle_{q_i} \Big\rangle_N$$

for some unknown polynomials f_i of degree $\alpha \in \mathbb{Z}_{q_i}[x]$, $i = 1, \ldots, m$. Given an oracle which for every query $x \in [0, 2^B)$ returns $h(x)$, find $h(x_0)$ for a given non-queried $x_0 \in [0, 2^B)$ using at most c queries.

A natural way of addressing this problem is by first finding the moduli $q_1, \ldots, q_m$ and polynomials $f_i \in \mathbb{Z}_{q_i}[x]$, and then using this information to obtain $h(x_i)$ for the given x_i. In fact, in [4], the authors studied the MMO problem *with known moduli*, which differs from the above MMO problem statement by the fact that the moduli $q_1, \ldots, q_m$ are given. It was shown that if c pairs $(x_1, h(x_1)), \ldots, (x_c, h(x_c))$ are given, finding polynomials $f_i \in \mathbb{Z}_{q_i}[x]$ such that $h(x_j) = \sum_{i=1}^{m} \langle f_i(x_j) \rangle_{q_i}$ for $1 \leq j \leq c$ is equivalent to finding a point in a certain lattice that is close to a target vector (depending on the observation $h(x_1), \ldots, h(x_c)$). A similar result is valid if the function h we are looking for is of the form $h(x) = \langle \sum_{i=1}^{m} \langle f_i(x) \rangle_{q_i} \rangle_N$.

The HIMMO problem considered in the present paper also requires that the moduli $q_1, \ldots, q_m$ are determined, and that is where the main contributions of the paper are. The paper contents are as follows. In Section 2, we provide a short description of HIMMO and show the motivation for the MMO problem. Section 3 describes results for a brute-force quantum search on the MMO problem. In Section 4, we present an iterative heuristic algorithm that queries the oracle at some specially chosen values x and then creates a list of possible values for the secret moduli $q_1, \ldots, q_m$. We also consider quantum speed-ups for the algorithm. The numerical experiments reported show that, although the algorithm is much faster than brute force approaches for small parameter values, solving the MMO problem for realistic parameter values remains infeasible. We conclude the paper in Section 5.

2 HIMMO and the MMO problem

In this section, we provide a short description of HIMMO and show the relevance of the MMO problem in a collusion attack on HIMMO.

HIMMO has several public parameters, *viz.*, the positive integers $B, b, \alpha, m \geq 2$, and the public modulus N, an odd integer of length exactly $(\alpha+1)B+b$ bits. A trusted third party randomly generates secret root keying material, consisting of

– m distinct random moduli $q_1, q_2, \ldots, q_m$ of the form $q_i = N - 2^b \beta_i$, where $0 < \beta_i < 2^B$ and at least one of $\beta_1, \ldots, \beta_m$ is odd.
– for $1 \leq i \leq m$ and $0 \leq j \leq k \leq \alpha$, a random integer $R_{j,k}^{(i)}$ with $0 \leq R_{j,k}^{(i)} \leq q_i - 1$, and for $0 \leq k < j \leq \alpha$, $R_{j,k}^{(i)} = R_{k,j}^{(i)}$.

The TTP provides, over a secure channel, each node in the system with a unique random B bit identifier ξ, with $0 \leq \xi < 2^B$, and with the coefficients of the key generating polynomial G_ξ:

$$G_\xi(y) = \sum_{k=0}^{\alpha} G_{\xi,k} y^k \text{ where } G_{\xi,k} = \langle \sum_{i=1}^{m} \langle \sum_{j=0}^{\alpha} R_{j,k}^{(i)} \xi^j \rangle_{q_i} \rangle_N.$$

Two nodes ξ and η in the system that wish to communicate with each other agree on a common b bits key based on $G_\xi(\eta)$ and $G_\eta(\xi)$.

In a collusion attack, multiple nodes $\xi_1, \ldots, \xi_c$ co-operate to obtain information on the key between two other nodes. In the collusion attack studied in the present paper, they try to obtain the root keying material, with which the key generating polynomial G_ξ of any node ξ can be determined. For obtaining the root keying material, the colluding nodes can use, for each $k \in \{0, \dots, \alpha\}$, the c equations

$$G_{\xi_\ell,k} = \langle \sum_{i=1}^{m} \langle \sum_{j=0}^{\alpha} R_{j,k}^{(i)} \xi_\ell^j \rangle_{q_i} \rangle_N \text{ for } 1 \le \ell \le c.$$

So for $0 \le k \le \alpha$, the colluding nodes know the value of the polynomial $h_k(x) = \langle \sum_{i=1}^m \langle f_i^k(x) \rangle_{q_i} \rangle_N$ for $x = \xi_1, \dots, \xi_c$, where $q_1, \dots, q_m$ and the polynomials $f_i^k(x) = \sum_{j=0}^{\alpha} R_{j,k}^{(i)} x^j$ (for $1 \le i \le m$) are unknown. That is, the nodes face an MMO problem for h_k after observation of c input-output pairs.

We note the MMO problem allows an attacker to use any strategy to query the oracle. In the remainder of the paper, we use this freedom to be in a good position to obtain information on the moduli $q_1, \dots, q_m$.

3 Preliminaries: classical and quantum brute-force search

A way to solve the MMO problem is to find m moduli of the form $q_i = N - 2^b \beta_i$, with $0 < \beta_i < 2^B$, and for each modulus, $\alpha + 1$ polynomial coefficients in the range $[0, q_i)$. As N is a $(\alpha+1)B + b$-bit number, the entire search space has $\mathcal{N}$ elements, where

$$\mathcal{N} = \sum_{\beta_1=1}^{2^B-1} \cdots \sum_{\beta_m=1}^{2^B-1} \prod_{i=1}^{m} (N - 2^B \beta_i)^{\alpha+1} \approx 2^{m((\alpha+1)^2+1)B+(\alpha+1)b} \qquad (1)$$

A brute force approach would be to check for each of these elements whether the resulting function $h(x)$ satisfies $h(x_j) = y_j$ in each of the c points.

Using Grover's algorithm [5], a quantum computer could speed up this brute-force attack with the aid of a 'quantum oracle', a unitary operator that flips the sign of state $|(\beta_1, f_{10}, \dots, f_{1\alpha}), \dots, (\beta_m, f_{m0}, \dots, f_{m\alpha})\rangle$ if

$$\langle \sum_{i=1}^{m} \langle \sum_{j=0}^{\alpha} f_{ij} x_k^j \rangle_{N-2^B\beta_i} \rangle_N = y_k \text{ for all } 1 \le k \le c,$$

and leaves it invariant otherwise. Grover's algorithm [5] requires $O(\sqrt{\mathcal{N}})$ 'queries' to this quantum oracle, compared to $c\mathcal{N}$ classical function evaluations. However, the running time would still be very far away from feasible for the HIMMO parameters recommended in [2], e.g., $m \ge 10, \alpha \ge 50$ and realistic values of b and B, e.g. $b = 256$, $B = 384$.

4 Finding the hidden moduli

4.1 Set-up

The function

$$h(x) = \langle \sum_{i=1}^{m} \langle f_i(x) \rangle_{q_i} \rangle_N$$

depends on the moduli q_i and the polynomial coefficients of the f_i. By taking repeated finite differences, the polynomial coefficients can be eliminated. This results in a set of equations for the moduli. Each solution corresponds to a set of candidates for the q_i, for which one may try to solve for the polynomial coefficients.

4.2 Finite differences

Let Δ be the difference operator:

$$\Delta F(x) = F(x+1) - F(x).$$

We note that for any integer $q \geq 1$ and function $f(x)$ on $\mathbb{Z}$ and we have

$$\begin{aligned}\Delta\langle f(x)\rangle_q &= \langle f(x+1)\rangle_q - \langle f(x)\rangle_q \\ &= \langle f(x+1) - f(x)\rangle_q - \vartheta(x)q = \langle \Delta f(x)\rangle_q - \vartheta(x)q,\end{aligned}$$

where $\vartheta(x) \in \{0,1\}$. Thus by induction, one easily verifies that for any integer $k \geq 2$,

$$\begin{aligned}\Delta^{(k)}\langle f(x)\rangle_q &= \Delta^{(k-1)}\langle f(x+1)\rangle_q - \Delta^{(k-1)}\langle f(x)\rangle_q \\ &= \langle \Delta^{(k)} f(x)\rangle_q - \vartheta^{(k)}(x)q,\end{aligned}$$

for some integers $\vartheta^{(k)}(x) \in \{-2^{k-1}+1, \ldots, 2^{k-1}\}$.

In particular, if f is a degree-α polynomial, then $\langle \Delta^{(\alpha+1)} f(x)\rangle_q = 0$, and so

$$\begin{aligned}\Delta^{(\alpha+1)} h(x) &= \Big\langle -\sum_{i=1}^{m} \vartheta_i^{(\alpha+1)}(x) q_i \Big\rangle_N - \vartheta_0^{(\alpha+1)}(x) N \\ &= \sum_{i=1}^{m} 2^b \vartheta_i^{(\alpha+1)}(x)\beta_i + (\epsilon(x) - \vartheta_0^{(\alpha+1)}(x))N,\end{aligned}$$

where in the last line we made use of the relations $q_i = N - 2^b\beta_i$, and used the fact that m is not too large, so that $m2^{b+B+\alpha} < N$. The number $\epsilon(x)$ is equal to 1 if $\sum_{i=1}^{m} \theta_i^{(\alpha+1)}(x)\beta_i < 0$ and equal to 0 otherwise.

A further modulo-N reduction to the interval $[-(N-1)/2, (N-1)/2]$, denoted by $\{\cdot\}_N$, gives

$$y(x) \overset{\text{def}}{=} 2^{-b}\{\Delta^{(\alpha+1)} h(x)\}_N = \sum_{i=1}^{m} \vartheta_i^{(\alpha+1)}(x)\beta_i. \qquad (2)$$

We remark that the derivation of (2) also holds for the finite difference operator Δ_s, defined as $\Delta_s F(x) = F(x+s) - F(x)$: in fact, each of the $\alpha+1$ derivatives may use a different s.

Now condider the sum of $y(x)$ for a number of consecutive values of x. This sum is itself a $\alpha+1$-st derivative, and thus also a linear combination of the β_i with integer coefficients in $[-2^\alpha+1, 2^\alpha]$:

$$\begin{aligned}\sum_{x=k}^{\ell} y(x) &= 2^{-b}\sum_{x=k}^{\ell}\{\Delta^{(\alpha+1)} h(x)\}_N \\ &= 2^{-b}\{\sum_{x=k}^{\ell}\Delta^{(\alpha+1)} h(x)\}_N, \text{ since } (\ell-k+1)m2^{\alpha+B} < N/2 \\ &= 2^{-b}\{\Delta_{\ell-k+1}\Delta^{(\alpha)} h(k)\}_N \\ &= \sum_{i=1}^{m} C_i\beta_i, \text{ where } -2^\alpha+1 \leq C_i \leq 2^\alpha \text{ for } 1 \leq i \leq m.\end{aligned}$$

4.3 Basic system of equations and attacks

Fix a number $n \geq 1$ and compute $y(x)$ for n consecutive values $x_0, x_0+1, \ldots, x_0+n-1$, which is possible with $\alpha+1+n$ queries to the oracle computing $h(x)$. This leads to the relations

$$y_j = \sum_{i=1}^{m} c_{ji}\beta_i, \qquad 1 \leq j \leq n$$

with known $y_j = y(x_0+j-1)$, unknown integer coefficients c_{ji} and unknown integers $\beta_i \in [1, 2^B)$. The coefficients c_{ji} satisfy $-2^\alpha + 1 \leq \sum_{j=k}^{\ell} c_{ji} \leq 2^\alpha$, for $1 \leq i \leq m$, $1 \leq k \leq \ell \leq n$. It can be shown that there are $(n+1)2^{n\alpha}$ sequences $(c_{1i}, \ldots, c_{ni})$ that satisfy these constraints.

Guess coefficients A first brute force solution is to guess the m^2 coefficients c_{ji}, $1 \leq i, j \leq m$, and solve for the β_i. This leads to a search space with $((m+1)2^{m\alpha})^m$ elements.

For each element that leads to one or more (if the matrix of coefficients is not of full rank) sets of the β_i. Each set gives a candidate set of moduli $q_i = N - 2^b\beta_i$ with which one may try to find a solution of the MMO problem with known moduli using standard lattice techniques.

Guess moduli A second brute force solution to find candidate moduli is to guess the m numbers $\hat{\beta}_i$ and, for each j, find m small integer coefficients c_{ji} i such that $y_j = \sum_{i=1}^{m} c_{ji}\hat{\beta}_i$ (this corresponds to a closest vector problem in $\mathbb{Z}^m$), and check whether

$$\sum_{j=k}^{\ell} c_{ji} \in [-2^\alpha + 1, 2^\alpha]$$

for all i, k, ℓ.. If such c_{ji} can be found the $\hat{\beta}_i$ give a candidate set of moduli.

Experiments show that this is not a very good technique: if m is small, the probability of finding a candidate set of moduli in this way is negligibly small. On the other hand, when m and α are larger, for example $m = 10$, $\alpha = 50$, it appears that a randomly chosen set of m integers from $[1, 2^B)$ has a good chance of qualifying as a candidate set, even if n is large. This implies that, for larger α and m, our basic idea of eliminating the polynomial coefficients from the problem by taking the $\alpha+1$-st discrete derivative defeats the purpose.

Vanishing linear combinations A third method, usable for small m, is to create vanishing linear combinations, in the following manner.

Let S be an integer $n \times (m-1)$ matrix with of which the elements satisfy the relations

$$\sum_{j=k}^{\ell} S_{ji} \in [-2^\alpha + 1, 2^\alpha] \text{ for all } 1 \leq i \leq m-1, 1 \leq k \leq \ell \leq n.$$

Think of S as the matrix of coefficients c_{ji}, with one column removed, say the k-th. Choose $L > 0$ and consider the $(L+1)^n$ integer linear combinations

$$(\sum_{j=1}^{n} \lambda_j S_{j1}, \ldots, \sum_{j=1}^{n} \lambda_j S_{j\,m-1}), \quad 0 \leq \lambda_j \leq L$$

It can be shown that each of these linear combinations is an integral point in X^{m-1}, where

$$X = \left[- \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)L, \left(1 + \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1) \right) L \right].$$

The interval X contains $1 + \left(1 + 2 \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L$ integral points, so if

$$(L+1)^n > \left(1 + \left(1 + 2 \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1) \right) L \right)^{m-1}, \tag{3}$$

then at least two of the linear combinations coincide. Taking their difference, we conclude that if (3) holds, there exists a nonzero integral vector $\lambda \in [-L, L]^n$ such that $\lambda S = 0$. So then for every $1 \leq k \leq m$, there is a vector λ such that

$$\sum_{j=1}^{n} \lambda_j y_j = \sum_{j=1}^{n} \lambda_j \sum_{i=1}^{m} c_{ji} \beta_i = \sum_{i=1}^{m} (\sum_{j=1}^{n} \lambda_j c_{ji}) \beta_i = (\sum_{j=1}^{n} \lambda_j c_{jk}) \beta_k = M_k \beta_k$$

for some M_k with $|M_k| \leq \left(1 + 2 \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L$.

This suggests an algorithm for finding $\beta_1, \ldots, \beta_m$.

1. Given α, b, B, m, find n, L satisfying (3) that minimise $(2L+1)^n$. Let Λ^+ be the set of integral vectors in $[-L, L]^n$ of which the first non-zero coordinate is positive. Let Cand $= [1, 2^B)$.
2. Choose a number $x_0 \in [0, 2^B - \alpha - n)$ and obtain $y_j = y(x_0 + j - 1)$, $1 \leq j \leq n$ by $\alpha + 1 + n$ queries to the oracle that calculates $h(x)$.
3. Set NewCand $= \emptyset$.
4. For each $\lambda \in \Lambda^+$, calculate $\sum_{j=1}^{n} \lambda_j y_j$, and if the result can be factored as $M\beta$, with $|M| \leq \left(1 + 2 \left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L$ and $\beta \in$ Cand, add β to NewCand.
5. Set Cand $\leftarrow$ NewCand.
6. If $|\text{Cand}| > m$, go back to step 2.
7. Return Cand.

However, this algorithm is flawed. The problem lies in step 4: it may happen that

$$\sum_{j=1}^{n} \lambda_j y_j = 0$$

for a $\lambda \in \Lambda^+$. In that case every β in Cand is added to NewCand and survives this round, so that the number of candidates does not decrease. A simple modification to the algorithm is to change the condition '...if the result can be factored as...' to '...if the result is non-zero and can be factored as...'. However, that leads to the elimination of a correct β_k, if for all vectors λ that would have resulted in it appearing as a factor of the result, the result happens to be zero.

This may be remedied somewhat by keeping a score for each candidate β, and incrementing it by one only when it appears as a factor of a non-zero result. This leads to the following, improved, algorithm.

1. Given α, b, B, m, find n, L satisfying (3) that minimise $(2L+1)^n$. Let Λ^+ be the set of integral vectors in $[-L, L]^n$ of which the first non-zero coordinate is positive. Choose a number of rounds, $N_{\text{rounds}} > 0$ and set Cand $= \emptyset$. Create an array, Scores, indexed by the elements of Cand. Do steps 2–5 N_{rounds} times.

2. Choose a number $x_0 \in [0, 2^B - \alpha - n)$ and obtain $y_j = y(x_0 + j - 1)$, $1 \leq j \leq n$ by $\alpha + 1 + n$ queries to the oracle that calculates $h(x)$.
3. Set NewCand $= \emptyset$.
4. For each $\lambda \in \Lambda^+$, calculate $\sum_{j=1}^{n} \lambda_j y_j$, and if the result is non-zero and can be factored as $M\beta$, with $|M| \leq \left(1 + 2\left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L$, and $\beta \in [1, 2^B)$, add β to NewCand.
5. For each $\beta \in$ NewCand, check if $\beta \in$ Cand. If yes, increment Scores[β], if no, add β to Cand and set Scores[β] to 1.

The idea is that the real values of $\beta_1, \ldots, \beta_m$ obtain a high score after sufficiently many rounds. Now the technique from the **Guess moduli** section above can be applied, but rather than picking the m candidate βs randomly from $[1, 2^B)$, they are chosen from the highest scoring βs.

The amount of work in each round is proportional to the size of Λ^+, which equals $((2L+1)^n - 1)/2$. Note that n, L are chosen in this algorithm to minimise this workload per round, conditional on (3).

We tested the algorithm for a few instances with low α and m. The results from a typical run with $\alpha = 2$, $m = 3$ and $b = 256$, $B = 384$ are given in Table 2. After 1000 rounds the three wanted β_is are among the top 5 candidates.

Birthday speed-up for vanishing linear combinations The vanishing linear combinations method is based on the occurrence of collisions, that is, different integral linear combinations of the y_j with coefficients in $[0, L]$ that give the same result; a collision is guaranteed to occur if (3) holds. When $\sqrt{\gamma |X|^{m-1}}$ different linear combinations are sampled, the probability of a collision is at least $1 - \exp(-\gamma/2)$. Replacing the condition (3) in the algorithm by

$$(L+1)^{2n} > \gamma \left(1 + \left(1 + 2\left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L\right)^{m-1} \tag{4}$$

results in a much smaller workload per round, depending on the value of γ that is chosen. The resulting workload for $\gamma = 1$ is shown in Table 3. It is a nice side effect of the lower workload that fewer candidates are generated per round. A disadvantage is that the probability that a β_i survives a round is lowered somewhat, since now the collision that would create the vanishing linear combination that gives this value of β is not guaranteed to occur. This implies that the scores are lower and that more rounds may be needed to obtain a set of β-values with higher scores than the rest. In practice the advantages clearly outweigh the disadvantages. Applying this algorithm to the same instance with $\alpha = 2$, $m = 3$ as before, we obtain the results given in Table 4. After 100 rounds the three correct β_i are in the top 7, after 1000 rounds they have moved into the top 4 and after 10000 rounds they take the top 3 places. Note that the workload per round for $\alpha = 2$, $m = 3$ is only $4 = 2^2$ in this birthday attack, as opposed to $364 = 2^{8.5}$ in the full attack, so that we can run 10 times as many rounds and still be about 9 times as fast.

4.4 Quantum speedups?

Step 4 of our algorithm can be rephrased as follows. Define for

$$1 \leq M \leq \left(1 + 2\left\lfloor \frac{n+1}{2} \right\rfloor (2^\alpha - 1)\right) L$$

Table 1. The workload-minimising parameters L, n and the logarithm of the resulting workload per round for $1 \leq \alpha \leq 20$ and $2 \leq m \leq 10$.

L					m				
α	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	2	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	2	1	1	1	1	1	1	1	1
11	4	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1
14	2	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1
17	4	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	1
20	3	1	1	1	1	1	1	1	1

n					m				
α	2	3	4	5	6	7	8	9	10
1	2	4	6	10	13	16	20	24	28
2	2	6	10	14	19	24	29	34	40
3	2	8	13	18	24	30	36	42	48
4	4	10	15	22	28	35	42	48	56
5	5	11	18	25	32	39	47	55	62
6	4	13	20	28	36	44	52	60	69
7	7	14	22	31	39	48	57	66	76
8	7	16	24	34	43	52	62	72	82
9	8	17	26	36	46	57	67	78	88
10	6	18	29	39	50	61	72	83	94
11	5	20	31	42	53	65	77	89	101
12	10	21	33	45	57	69	82	94	107
13	11	23	35	47	60	73	86	100	113
14	8	24	37	50	64	77	91	105	119
15	12	25	39	53	67	81	96	110	125
16	13	27	41	56	70	85	101	116	131
17	7	28	43	58	74	89	105	121	137
18	14	29	45	61	77	93	110	127	143
19	15	31	47	64	80	97	115	132	149
20	9	32	49	66	84	101	119	137	155

$\log_2(\lvert\Lambda^+\rvert)$					m				
α	2	3	4	5	6	7	8	9	10
1	2.0	5.3	8.5	14.8	19.6	24.4	30.7	37.0	43.4
2	2.0	8.5	14.8	21.2	29.1	37.0	45.0	52.9	62.4
3	4.6	11.7	19.6	27.5	37.0	46.5	56.1	65.6	75.1
4	5.3	14.8	22.8	33.9	43.4	54.5	65.6	75.1	87.8
5	6.9	16.4	27.5	38.6	49.7	60.8	73.5	86.2	97.3
6	8.3	19.6	30.7	43.4	56.1	68.7	81.4	94.1	108.4
7	10.1	21.2	33.9	48.1	60.8	75.1	89.3	103.6	119.5
8	10.1	24.4	37.0	52.9	67.2	81.4	97.3	113.1	129.0
9	11.7	25.9	40.2	56.1	71.9	89.3	105.2	122.6	138.5
10	12.9	27.5	45.0	60.8	78.2	95.7	113.1	130.6	148.0
11	14.8	30.7	48.1	65.6	83.0	102.0	121.0	140.1	159.1
12	14.8	32.3	51.3	70.3	89.3	108.4	129.0	148.0	168.6
13	16.4	35.5	54.5	73.5	94.1	114.7	135.3	157.5	178.1
14	17.6	37.0	57.6	78.2	100.4	121.0	143.2	165.4	187.6
15	18.0	38.6	60.8	83.0	105.2	127.4	151.2	173.3	197.1
16	19.6	41.8	64.0	87.8	109.9	133.7	159.1	182.9	206.6
17	21.2	43.4	67.2	90.9	116.3	140.1	165.4	190.8	216.1
18	21.2	45.0	70.3	95.7	121.0	146.4	173.3	200.3	225.6
19	22.8	48.1	73.5	100.4	125.8	152.7	181.3	208.2	235.2
20	24.3	49.7	76.7	103.6	132.1	159.1	187.6	216.1	244.7

Table 2. Result for an instance with $\alpha = 2$, $m = 3$ and $B = b = 256$ after 10, 100, and 1000 rounds. The scores are given as $(s, n)_j$ triplets, where s = score, n = number of candidates with this score j = number of β_is with this score.

N_{rounds}	highest scores
10	$(10,4)_2$, $(9,12)_0$, $(8,9)_1$, $(7,23)_0$, $(6,50)_0$, $(5,17)_0$, $(4,112)_0$, $(3,227)_0$, $(2,333)_0$, $(1,1005)_0$
100	$(96,2)_1$, $(93,2)_1$, $(89,2)_0$, $(86,1)_1$, $(82,2)_0$, $(81,4)_0$, $(80,2)_0$, $(78,4)_0$, ...
1000	$(980,1)_1$, $(974,1)_0$, $(860,2)_1$, $(859,1)_1$, $(839,1)_0$, $(838,1)_0$, $(817,2)_0$, $(810,2)_0$, $(804,3)_0$, ...

Table 3. The base-2 logarithm of the workload per round for the birthday algorithm with $\gamma = 1$.

$\log_2(\lvert\Lambda^+\rvert)$	m								
α	2	3	4	5	6	7	8	9	10
1	2.0	2.0	2.0	5.3	6.9	8.5	11.7	14.8	16.4
2	2.0	2.0	5.3	8.5	11.7	14.8	18.0	21.2	25.9
3	2.0	4.6	8.3	11.7	14.8	19.6	24.4	27.5	32.3
4	2.0	5.3	10.1	14.8	18.0	22.8	27.5	33.9	38.6
5	2.0	6.9	11.7	16.4	21.2	27.5	32.3	38.6	43.4
6	3.6	8.3	13.3	19.6	24.4	30.7	37.0	43.4	49.7
7	3.6	10.1	14.8	21.2	27.5	33.9	40.2	48.1	54.5
8	4.6	10.1	16.4	24.4	30.7	37.0	45.0	52.9	59.2
9	5.3	11.7	18.0	25.9	33.9	40.2	48.1	56.1	64.0
10	5.3	12.9	21.2	27.5	37.0	45.0	52.9	60.8	68.7
11	6.4	14.8	22.2	30.7	38.6	48.1	56.1	65.6	75.1
12	6.9	14.8	24.4	32.3	41.8	51.3	60.8	70.3	79.8
13	6.9	16.4	25.9	35.5	45.0	54.5	64.0	73.5	84.6
14	8.3	17.6	27.5	37.0	46.5	57.6	67.2	78.2	89.3
15	8.3	18.0	29.1	38.6	49.7	60.8	71.9	83.0	94.1
16	8.5	19.6	30.7	41.8	52.9	64.0	75.1	87.8	98.9
17	10.1	21.2	32.3	43.4	54.5	67.2	79.8	90.9	103.6
18	10.1	21.2	33.8	45.0	57.6	70.3	83.0	95.7	108.4
19	10.1	22.8	35.5	48.1	60.8	73.5	86.2	100.4	113.1
20	11.7	24.3	37.0	49.7	62.4	76.7	90.9	103.6	117.9

Table 4. Result from running the birthday attack on the same instance with $\alpha = 2$, $m = 3$ and $B = b = 256$ as before, after 10, 100, 1000 and 10000 rounds. The scores are given as $(s, n)_j$ triplets, where $s =$ score, $n =$ number of candidates with this score $j =$ number of β_is with this score.

N_{rounds}	highest scores
10	$(4,1)_0$, $(3,3)_1$, $(2,7)_1$, $(1,52)_1$
100	$(23,1)_1$, $(22,1)_0$, $(19,3)_1$, $(15,2)_1$, $(13,4)_0$, $(12,1)_0$, $(11,3)_0$, $(10,3)_0$, $(9,6)_0$, $\ldots$
1000	$(224,1)_1$, $(216,1)_1$, $(212,1)_0$, $(200,1)_1$, $(152,1)_0$, $(138,1)_0$, $(136,1)_0$, $(132,1)_0$, $\ldots$
10000	$(2175,1)_1$, $(2127,1)_1$, $(2086,1)_1$, $(2033,1)_0$, $(1416,1)_0$ $(1276,1)_0$, $(1275,1)_0$, $(1262,1)_0$, $\ldots$

the function $F_M : \Lambda^+ \to [0, 2^B)$ as

$$F_M(\lambda) = \begin{cases} \beta & \text{if } \left|\sum_{j=1}^{n} \lambda_j y_j\right| = M\beta \text{ with } 1 \leq \beta < 2^B \\ 0 & \text{otherwise.} \end{cases}$$

Let R_M be the image of Λ^+ under F_M. Then the set NewCand is given by the union of all the R_M, minus $\{0\}$. We are not aware of a quantum algorithm that can calculate NewCand faster than its classical counterpart.

5 Conclusions

The HIMMO scheme is a potential quantum-safe alternative since its underlying design principles, the HI and MMO problems [2],[3] are related to lattice problems. HIMMO exhibits excellent performance compared with other quantum-safe alternatives, in particular regarding bandwidth needs, while providing several security services such as key agreement, implicit certificates, or source authentication. In order to further evaluate HIMMO's security, this paper focuses on the MMO problem with unknown moduli and several methods for tackling it. A quantum brute force search of the secret moduli and polynomial coefficients would have a running time $O(2^{(m((\alpha+1)^2+1)B+(\alpha+1)b)/2})$, too large to allow for brute force attacks for proposed HIMMO parameter values. Therefore, we introduced the method of using finite differences in order to eliminate the polynomial coefficients from the problem. By guessing the coefficients, we come up with $((m+1)2^{m\alpha})^m$ possible choices for the unknown moduli. For each such choice, we need to solve an MMO problem with known moduli to verify if we have found a solution for the MMO problem with unknown moduli. Alternatively, we can guess the moduli (2^{mB} choices) and for each guess solve a close vector problem in $\mathbb{Z}_m$ to verify if the guessed moduli form a valid set of moduli. Experiments indicate that this method is not very attractive. With the method of vanishing linear combinations, we run N_{rounds} rounds, each with a workload of $((2L+1)^n - 1)/2$, where L and n satisfy (3). In the birthday speed-up, the workload per round again is $((2L+1)^n - 1)/2$, but L and n satisfy the much weaker condition from (4). After the iterations, a small set of candidates for the β_i's remain if m and α are small. The results of the workloads per round collected in Tables 1 and 3 show that for proposed values m and α, the method of vanishing linear combinations and its birthday speed-up are infeasible as well. As a conclusion, although the methods in this paper can help to attack the MMO problem and HIMMO scheme with small parameters (`https://www.himmo-scheme.com/challenge/`), HIMMO remains safe for proposed parameters $m = 10$ and $\alpha = 50$ even if a quantum computer were available. Other HIMMO parameters such as the key or identifier size would need to be increased in order to generate long enough keys (256 bits) for later usage or to ensure secure binding between credentials and long identifiers (384 bits) by means of a collision resistance hash function, corresponding to a security level of 128 bits.

References

1. C. Blundo, A. de Santis, A.Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly secure key distribution for dynamic conferences. *Information and Computation*, 146:1–23, 1998.
2. O. Garcia-Morchon, D. Gomez-Perez, J. Gutierrez, R. Rietman, B. Schoenmakers, and L. Tolhuizen. HIMMO: A Lightweight Collusion-Resistant Key Predistribution Scheme, 2015. Cryptology ePrint Archive, Report 2014-698, Version of Aug. 18, 2015.

3. O. Garcia-Morchon and L. Tolhuizen. Towards full collusion-resistant ID based establishment of pairwise keys. In *Exteded abstracts of Third International Conference on Symbolic Computation and Cryptography and Third Workshop on Mathematical Cryptography*, pages 30–35, July 9-13 2012.
4. Oscar García-Morchón, Domingo Gómez-Pérez, Jaime Gutiérrez, Ronald Rietman, and Ludo Tolhuizen. The MMO problem. In *Proc. ISSAC'14*, pages 186–193. ACM, 2014.
5. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219. ACM, 1996.
6. T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest vectors faster using quantum serarch. *Designs, Codes and Cryptography*, 77:375–400, 2015. DOI 10.1007/s10623-015-0067-5.
7. T. Matsumoto and H. Imai. On the key predistribution system: a practical solution to the key distribution problem. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, LNCS 293, pages 185–193. Springer, 1988.