MQ on my Mind: Post-Quantum Signatures from the Non-Structured Multivariate Quadratic Problem

Ryad Benadjila[1], Thibauld Feneuil[1,2], and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France
[2] Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France
{ryad.benadjila,thibauld.feneuil,matthieu.rivain}@cryptoexperts.com

Abstract. This paper presents *MQ on my Mind* (MQOM), a digital signature scheme based on the difficulty of solving multivariate systems of quadratic equations (MQ problem). MQOM has been submitted to the NIST call for additional post-quantum signature schemes. MQOM relies on the *MPC-in-the-Head* (MPCitH) paradigm to build a zero-knowledge proof of knowledge (ZK-PoK) for MQ which is then turned into a signature scheme through the Fiat-Shamir heuristic. The underlying MQ problem is *non-structured* in the sense that the system of quadratic equations defining an instance is drawn uniformly at random. This is one of the hardest and most studied problems from multivariate cryptography which hence constitutes a conservative choice to build candidate post-quantum cryptosystems. For the efficient application of the MPCitH paradigm, we design a specific MPC protocol to verify the solution of an MQ instance. Compared to other multivariate signature schemes based on non-structured MQ instances, MQOM achieves the shortest signatures (6.3–7.8 KB) while keeping very short public keys (few dozen of bytes). Other multivariate signature schemes are based on structured MQ problems (less conservative) which either have large public keys (*e.g.* UOV) or use recently proposed variants of these MQ problems (*e.g.* MAYO).

Keywords: Post-Quantum Signatures, MPC in the Head, Multivariate Quadratic Problem

1 Introduction

The advent of a quantum computer capable of breaking classical public-key cryptosystems (RSA, ECC) urges the cryptography research community to find reliable alternatives for public key encryption and signatures. The so-called *post-quantum cryptography* has become a hot topic, especially since 2017 when the NIST launched its post-quantum standardization process. After five years of intense community efforts, the first post-quantum algorithms have been selected for standardization. Recently, the NIST has issued a call for additional post-quantum signature schemes [22] to get further alternatives to the selected ones, the latter being either based on structured lattices (Dilithium [21], Falcon [23]) or hash-based with mitigated performances (SPHINCS+ [16]).

There exist two approaches to build signature schemes. On the one hand, the hash-and-sign paradigm relies on the existence of a trapdoor permutation. This is for instance the case of the lattice-based signature scheme Falcon [23] or the multivariate signature scheme UOV [20]. The existence of a trapdoor might make such schemes vulnerable to structural attacks. On the other hand, signature schemes can be constructed by applying the Fiat-Shamir transform to a (zero-knowledge) identification scheme, which can then rely on weaker hardness assumptions, such as *e.g.* the syndrome decoding problem for random linear codes [25,13].

Multivariate cryptography is an important family of cryptosystems based on allegedly post-quantum hard problems. Among these problems, the non-structured multivariate quadratic problem (*i.e.* solving a multivariate system of random quadratic equations) is probably the hardest and most studied problem. Multivariate signature schemes in the state of the art are of three kinds: (1) variants of UOV, which are fairly efficient and enjoy compact signatures but suffer large public keys, (2) hash-and-sign schemes which circumvent the latter issue by relying on new structured multivariate assumptions (*e.g.* MAYO [7]), (3) signature schemes based on the Fiat-Shamir transform, some of which based on the (conservative) non-structured MQ problem (*e.g.* MQ-DSS [24]). The present paper introduces MQOM, a signature scheme belonging to this third category.

MQOM relies on the *MPC-in-the-Head* (MPCitH) paradigm which builds a zero-knowledge proof of knowledge from a secure multi-party computation (MPC) protocol [17]. While this approach was mainly considered of theoretical interest when introduced in 2007, it has been increasingly applied to build practical schemes over the last years. In particular, the Picnic signature scheme [28] submitted to the NIST standardization process in 2017 relies on this paradigm. Since then, MPCitH signature schemes have been proposed based on various hardness assumptions, see *e.g.* [6,2,11,13,14,12], and generic improvements of the MPCitH techniques have been proposed, see *e.g.* [19,15,1].

To efficiently apply the MPCitH paradigm to the unstructured MQ problem, we design a specific "MPCitH-friendly" MPC protocol that verifies the solution of an MQ instance. The MQOM MPC protocol is obtained by improving previous techniques using batching and polynomial interpolations [12,2,10]. We further use generic techniques to improve the MPCitH transform, namely seed trees [19] and hypercube party computation [1]. Compared to other multivariate signature schemes based on the non-structured MQ problem, MQOM achieves the shortest signatures (6.3–7.8 KB) while keeping very short public keys (a few dozen of bytes).

The paper is organized as follows: In Section 2, we describe the MQOM MPC protocol which verifies the solution of an MQ instance in an "MPCitH-friendly" way. Then Section 3 depicts the MQOM signature scheme obtained by applying the MPCitH transform and the Fiat-Shamir transform to the latter MPC protocol. In Section 4, we analyze the security of MQOM: we prove its unforgeability against chosen message attacks (EUF-CMA) in the random oracle model and we discuss known attacks against the MQ problem as well as a generic forgery attack against this type of signature schemes. Section 5 explains the choice of the various MQ and MPC parameters defining the different instances of MQOM. Section 6 addresses implementation aspects and reports benchmarks. Finally, we compare MQOM to state-of-the-art multivariate signature schemes in Section 7.

2 The MQOM MPC Protocol

In this section, we describe the MQOM MPC protocol which is at the core of the MQOM signature scheme. The MQOM MPC protocol runs a multi-party computation that verifies the correctness of a solution x to a public MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$. The secret solution x is shared between N parties which, after running the protocol, either output Accept if the input sharing is believed to encode a correct MQ solution or Reject otherwise.

2.1 MQ problem

We first recall the definition of the (non-structured) MQ problem for which our MPC protocol verifies an instance.

Definition 1 (Multivariate Quadratic Problem). *Let q be a prime (or a prime power) and let m, n be positive integers. The multivariate quadratic problem with parameters (q, m, n) is the following problem:*

Let $(A_i)_{i\in[1:m]}$, $(b_i)_{i\in[1:m]}$, x and y be such that:

1. *x is uniformly sampled from $\mathbb{F}_q^n$,*
2. *for all $i \in [1:m]$, A_i is uniformly sampled from $\mathbb{F}_q^{n\times n}$,*
3. *for all $i \in [1:m]$, b_i is uniformly sampled from $\mathbb{F}_q^n$,*
4. *for all $i \in [1:m]$, y_i is defined as $y_i := x^\intercal A_i x + b_i^\intercal x$.*

From $\big(\{A_i\}, \{b_i\}, y\big)$, find x.

2.2 MPC model

The MQOM protocol relies on a standard model of the MPCitH paradigm as formalized *e.g.* in [15, Section 3.1]. In this model, the parties receive as input a sharing

$$[\![x]\!] = ([\![x]\!]_1, \dots, [\![x]\!]_N)$$

of the secret witness x. Then, they perform a sequence of the following actions:

1. Invoke a randomness oracle $\mathcal{O}_R$. This oracle sends a random value γ to all the parties. In MPCitH context, these random values are provided by the verifier as challenges.
2. Invoke a hint oracle $\mathcal{O}_H$. For some function ψ, the hint is sampled as $\beta \leftarrow \psi(x, \gamma^1, \gamma^2, \dots; r_\psi)$ where $\gamma^1, \gamma^2, \dots$ are the previous outputs of $\mathcal{O}_R$ and where r_ψ is some fresh randomness. The hint oracle sends a random sharing $[\![\beta]\!]$ of the hint to the parties. In the MPCitH context, the hint is computed by the prover and the obtained shares are committed with the shares of the witness.
3. Compute and broadcast shares. The parties locally compute $[\![\alpha]\!] := [\![\varphi(v)]\!]$ from a sharing $[\![v]\!]$ for some $\mathbb{F}_q$-linear function φ. Then they broadcast the shares $[\![\alpha]\!]_1$, ..., $[\![\alpha]\!]_N$ and publicly reconstruct $\alpha = \varphi(v)$. This local computation process is denoted $[\![\varphi(v)]\!] = \varphi([\![v]\!])$. The function φ can depend on the previous random values from $\mathcal{O}_R$ and on the previous broadcasted values from $\mathcal{O}_H$.

After a given number of rounds, the parties broadcast a final sharing $[\![v]\!]$ and publicly recompute v. If $v = 0$, they output Accept, if $v \neq 0$, they output Reject.

The protocol is said perfectly complete, if on input a sharing $[\![x]\!]$ of the right MQ solution, the parties always output Accept. On the other hand, the protocol has *false positive probability* p, if the probability that the parties output Accept on input $[\![x]\!]$ corresponding to an incorrect MQ solution is at most p. The MQOM protocol is perfectly complete and has false positive probability which we exhibit in Section 2.5.

2.3 Principle of the MPC protocol

The parties aim to verify that their input sharing $[\![x]\!]$ corresponds to a solution $x \in \mathbb{F}_q^n$ of the following system:

$$\begin{cases} y_1 = & x^\intercal A_1 x + b_1^\intercal x \\ \quad\vdots & \\ y_m = & x^\intercal A_m x + b_m^\intercal x \end{cases}$$

for a given MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$.

As a first step of the MQOM protocol, we batch the equations of the MQ system as suggested in [12]. Instead of checking the m equations separately, the parties verify a linear combination of these equations with coefficients $\gamma_1, \ldots, \gamma_m$ that are uniformly sampled by the randomness oracle $\mathcal{O}_R$ from a field extension $\mathbb{F}_{q^\eta}$. The MPC protocol shall then check that

$$\sum_{i=1}^{m} \gamma_i (y_i - x^\intercal A_i x - b_i^\intercal x) = 0. \tag{1}$$

If one of the equations of the MQ system is not satisfied, then Equation 1 is only satisfied with a probability $1/q^\eta$. The above equality rewrites as

$$\begin{aligned} \sum_{i=1}^{m} \gamma_i (y_i - b_i^\intercal x) &= \sum_{i=1}^{m} \gamma_i (x^\intercal A_i x) \\ &= x^\intercal \Big(\sum_{i=1}^{m} \gamma_i A_i \Big) x \\ &= \langle x, w \rangle \quad \text{where} \quad w := \Big(\sum_{i=1}^{m} \gamma_i A_i \Big) x \end{aligned}$$

By defining $z := \sum_{i=1}^{m} \gamma_i (y_i - b_i^\intercal x)$ and $w := (\sum_{i=1}^{m} \gamma_i A_i)\, x$, checking Equation 1 is equivalent to checking $z = \langle x, w \rangle$. On receiving the random coefficients $\gamma_1, \ldots, \gamma_m$ the parties can locally compute a sharing $[\![w]\!]$ of w from the sharing $[\![x]\!]$.

As a second step of the MQOM protocol, the parties verify the inner product $z = \langle x, w \rangle$ from the sharings $[\![x]\!]$, $[\![w]\!]$ and $[\![z]\!]$. For this purpose, we introduce an inner-product verification protocol, using polynomial interpolation techniques as Banquet [2] or Limbo [10] with a slightly improved communication.

The principle is to split the vectors x and z into n_2 chunks of n_1 coordinates for integers n_1, n_2 such that $n_1 \cdot n_2 \geq n$. The chunks are used to interpolate polynomials $X[1](u), \ldots, X[n_2](u) \in \mathbb{F}_q[u]$ and $W[1](u), \ldots, W[n_2](u) \in \mathbb{F}_{q^\eta}[u]$, which satisfy

$$\begin{cases} X[j](f_1) = & x_{(j-1)n_1+1} \\ & \vdots \\ X[j](f_{n_1}) = & x_{(j-1)n_1+n_1} \end{cases} \tag{2}$$

and

$$\begin{cases} W[j](f_1) = & w_{(j-1)n_1+1} \\ & \vdots \\ W[j](f_{n_1}) = & w_{(j-1)n_1+n_1} \end{cases} \tag{3}$$

for every $j \in [1 : n_2]$ where $f_1, \ldots, f_{n_1}$ are n_1 fixed distinct elements from $\mathbb{F}_q$. Let us stress that the $X[j]$'s are polynomials of degree $\deg(X[j]) \leq n_1 - 1$ from $\mathbb{F}_q[u]$ while the $W[j]$'s are polynomials of degree $\deg(W[j]) \leq n_1 - 1$ from $\mathbb{F}_{q^\eta}[u]$. By definition of those polynomials, we have the following equivalence

$$z = \langle x, w \rangle \quad \iff \quad z = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} X[j](f_i) \cdot W[j](f_i)\ .$$

Defining the polynomial $Q_0(u) \in \mathbb{F}_{q^\eta}[u]$ as

$$Q_0(u) := \sum_{j=1}^{n_2} X\,\texttt{[}j\texttt{]}\,(u) \cdot W\,\texttt{[}j\texttt{]}\,(u) \ , \tag{4}$$

we now have $z = \langle x, w \rangle$ if and only if $z = \sum_{i=1}^{n_1} Q_0(f_i)$.

In a nutshell, the MQOM MPC protocol works as follows:

- The parties locally compute sharings $[\![X\,\texttt{[}j\texttt{]}\,]\!]$ and $[\![W\,\texttt{[}j\texttt{]}\,]\!]$ by interpolation from $[\![x]\!]$ and $[\![w]\!]$ (which is possible since such interpolation is $\mathbb{F}_q$-linear);
- The parties invoke the hint oracle to obtain a sharing $[\![Q_0]\!]$ of the polynomial Q_0;
- At this stage, the parties aim to check that the two following hold:
  (i) the polynomial Q_0 from the hint oracle well satisfies Equation 4,
  (ii) the polynomial Q_0 from the hint oracle well satisfies $z = \sum_{i=1}^{n_1} Q_0(f_i)$.
- To verify Statement (i), the protocol relies on the Schwartz–Zippel lemma. Namely, Equation 4 is verified on a random evaluation point $r \in \mathbb{F}_{q^\eta} \setminus \{f_1, \ldots, f_{n_1}\}$. The parties proceed as follows:
  1. they request a random evaluation point $r \in \mathbb{F}_{q^\eta} \setminus \{f_1, \ldots, f_{n_1}\}$ from the randomness oracle $\mathcal{O}_R$;
  2. they locally compute $[\![\alpha\,\texttt{[}j\texttt{]}\,]\!] = [\![W\,\texttt{[}j\texttt{]}\,]\!](r)$ for all $j \in [1 : n_2]$;
  3. they broadcast $[\![\alpha\,\texttt{[}j\texttt{]}\,]\!]$ and publicly recompute $\alpha\,\texttt{[}j\texttt{]}$ for all $j \in [1 : n_2]$;
  4. they locally compute $[\![v_1]\!] = [\![Q_0]\!](r) - \sum_{j=1}^{n_2} \alpha\,\texttt{[}j\texttt{]} \cdot [\![X\,\texttt{[}j\texttt{]}\,]\!](r)$;
  5. they broadcast $[\![v_1]\!]$ and publicly recompute $v_1 = Q_0(r) - \sum_{j=1}^{n_2} \alpha\,\texttt{[}j\texttt{]} \cdot X\,\texttt{[}j\texttt{]}\,(r)$;
  6. they verify that $v_1 = 0$.
- To verify Statement (ii), the parties proceed as follows:
  1. they locally compute $[\![v_2]\!] = [\![z]\!] - \sum_{i=1}^{n_1} [\![Q_0]\!](f_i)$;
  2. they broadcast $[\![v_2]\!]$ and publicly recompute $v_2 = z - \sum_{i=1}^{n_1} Q_0(f_i)$;
  3. they verify that $v_2 = 0$.

Making the protocol private. The careful reader might have noticed a caveat in the above protocol: the broadcast shares leak information on the secret witness. Specifically, the publicly recomputed values $\alpha\,\texttt{[}j\texttt{]}$ are evaluations of the polynomials $W\,\texttt{[}j\texttt{]}\,(u)$ in a public point r which provide linear combinations of the vector w, namely linear combinations of the secret x. In order to ensure the zero-knowledge property when applying the MPCitH transformation, we should make the protocol $(N-1)$-private. Namely, one should not learn any information about x from any $N-1$ parties' views, including all the broadcast shares.

To deal with this issue we use a standard solution which consists in masking the polynomials $W\,\texttt{[}j\texttt{]}$'s as

$$\tilde{W}\,\texttt{[}j\texttt{]}\,(u) := W\,\texttt{[}j\texttt{]}\,(u) + a\,\texttt{[}j\texttt{]} \cdot V(u)$$

where the $a\,\texttt{[}j\texttt{]}$'s are random elements of $\mathbb{F}_{q^\eta}$ obtained from the hint oracle $\mathcal{O}_H$ and where $V(u)$ is the *vanishing polynomial* of the interpolation set $\{f_1, \ldots, f_{n_1}\}$, which is defined as

$$V(u) := (u - f_1)(u - f_2) \cdots (u - f_{n_1}) \ .$$

The polynomial Q_0 is then replaced by the polynomial Q defined as

$$Q(u) := \sum_{j=1}^{n_2} X\,\texttt{[}j\texttt{]}\,(u) \cdot \tilde{W}\,\texttt{[}j\texttt{]}\,(u) \tag{5}$$

which –by definition of the vanishing polynomial– still satisfies the relation

$$z = \langle x, w \rangle \quad \iff \quad z = \sum_{i=1}^{n_1} Q(f_i) \ . \tag{6}$$

The protocol works the same way as before with Q in place of Q_0 but now the revealed evaluations $\tilde{W}[1](r), \ldots, \tilde{W}[n_1](r)$ do not leak any information about the secret witness. Indeed, $\tilde{W}[j](r) = W[j](r) + a[j] \cdot V(r)$ with $V(r) \neq 0$ since $r \notin \{f_1, \ldots, f_{n_1}\}$ and $a[j]$ are uniformly random over $\mathbb{F}_{q^\eta}$.

Dealing with incomplete last chunk. In practice, the optimal parameters n_1, n_2 in terms of communication of the MPCitH-transformed protocol might be such that $n_1 \cdot n_2$ is strictly greater than n. In that case, the last of the n_2 chunks of x and w are of size smaller than n_1. Let $n_1' < n_1$, the size of the last chunks, such that $n = (n_2 - 1)n_1 + n_1'$. The last chunk polynomials $X[n_2]$ and $W[n_2]$ are defined as the polynomials of degrees $\deg(X[n_2]) \leq n_1$ and $\deg(W[n_2]) \leq n_1'$ respectively satisfying:

$$\begin{cases} X[n_2](f_1) = x_{(n_2-1)n_1+1} \\ \quad \vdots \\ X[n_2](f_{n_1'}) = x_{(n_2-1)n_1+n_1'} \\ X[n_2](f_{n_1'+1}) = 0 \\ \quad \vdots \\ X[n_2](f_{n_1}) = 0 \end{cases} \tag{7}$$

and

$$\begin{cases} W[n_2](f_1) = w_{(n_2-1)n_1+1} \\ \quad \vdots \\ W[n_2](f_{n_1'}) = w_{(n_2-1)n_1+n_1'} \end{cases} \tag{8}$$

This way, whatever the values taken by $W[n_2](f_i)$, we always have $X[n_2](f_i) \cdot \tilde{W}[n_2](f_i) = 0$ for $i \in [n_1' + 1 : n_1]$ so that Equation 6 still holds true.

Optimizing the communication. The MPC protocol as depicted above consists in checking that the publicly reconstructed values v_1 and v_2 satisfy $v_1 = v_2 = 0$, which is

$$\begin{cases} Q(r) - \sum_{j=1}^{n_2} \alpha[j] \cdot X[j](r) = 0 \\ z - \sum_{i=1}^{n_1} Q(f_i) = 0 \end{cases} \tag{9}$$

By denoting q_0 the constant term of $Q(u)$ and letting $Q'(u)$ the degree-$(2n_1 - 2)$ polynomial such that

$$Q(u) = u \cdot Q'(u) + q_0 \ , \tag{10}$$

the above system is equivalent to (swapping the two equations):

$$\begin{cases} q_0 = (n_1)^{-1}\Big(z - \sum_{i=1}^{n_1} f_i \cdot Q'(f_i)\Big) \\ v := r \cdot Q'(r) + q_0 - \sum_{j=1}^{n_2} \alpha[j] \cdot X[j](r) = 0 \end{cases} \tag{11}$$

(assuming that n_1 is co-prime with q, the base field characteristic, which shall always be the case in our context). So rather than requesting a hint $[\![Q]\!]$ and broadcasting the shares of v_1 and v_2, the parties can instead request a hint $[\![Q']\!]$ and check the two equations simultaneously by

– locally computing the sharing $[\![q_0]\!]$ as

$$[\![q_0]\!] = (n_1)^{-1}\Big(z - \sum_{i=1}^{n_1} f_i \cdot [\![Q']\!](f_i)\Big)$$

– locally computing the sharing $[\![v]\!]$ as

$$[\![v]\!] = r \cdot [\![Q']\!](r) + [\![q_0]\!] - \sum_{j=1}^{n_2} \alpha\,[j] \cdot [\![X\,[j]\,]\!](r)$$

– broadcasting $[\![v]\!]$, publicly recomputing v and checking $v = 0$.

This way, the hint (Q' vs. Q) is shorter by one $\mathbb{F}_{q^\eta}$ element (which accounts in the ZK protocol's communication) and the broadcast is reduced by replacing $([\![v_1]\!], [\![v_2]\!])$ by $[\![v]\!]$ (which also accounts as one $\mathbb{F}_{q^\eta}$ element in the ZK protocol).

2.4 Description of the MPC protocol

Wrapping up the different tweaks described above, we obtain the MQOM MPC protocol which is formally depicted in Figure 1.

2.5 False positive probability

Theorem 1. *If x is the right solution to the MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$ and if $[\![Q']\!]$ is genuinely generated as a sharing of Q' by the hint oracle $\mathcal{O}_H$, then the protocol always outputs* Accept. *If x does not satisfy the MQ instance, then whatever the hint returned by $\mathcal{O}_H$, the protocol outputs* Accept *with probability at most $p_1 + (1 - p_1) \cdot p_2$ with*

$$p_1 := \frac{1}{q^\eta} \qquad \text{and} \qquad p_2 := \frac{2n_1 - 1}{q^\eta - n_1}.$$

Proof. The completeness of the protocol holds following the presentation of Section 2.3. Now, let us assume that x is not a solution of the MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$. There are two cases:

1. Either $\langle w, x\rangle = z$, this case occurs with probability $p_1 = \frac{1}{q^\eta}$ over the randomness of $(\gamma_j)_j$;
2. Or, $\langle w, x\rangle \neq z$. Let us define the polynomial P of degree $\deg(P) \leq 2n_1 - 1$ defined w.r.t. Q', X, W and z as:

$$P(u) := u \cdot Q'(u) + q_0 - \sum_{j=1}^{n_2} W\,[j]\,(u) \cdot X\,[j]\,(u).$$

with $q_0 := (n_1)^{-1}\Big(z - \sum_{i=1}^{n_1} f_i \cdot Q'(f_i)\Big)$.

Input: The parties receive a sharing $\llbracket x \rrbracket$ of a solution x to an MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$.

1. The parties invoke the randomness oracle $\mathcal{O}_R$ to get random $\gamma_1, \dots, \gamma_m \in \mathbb{F}_{q^\eta}$.
2. The parties locally compute $\llbracket z \rrbracket = \sum_{i=1}^{m} \gamma_i (y_i - b_i^\intercal \llbracket x \rrbracket)$.
3. The parties locally compute $\llbracket w \rrbracket = \big(\sum_{i=1}^{m} \gamma_i A_i\big) \llbracket x \rrbracket$.

⇒ **Now, the parties should check that $z = \langle x, w \rangle$:**

4. The parties locally interpolate the chunk polynomials $\llbracket X[j] \rrbracket$, $\llbracket W[j] \rrbracket$, for $j \in [1 : n_2]$, as defined in Equation 2, Equation 3, Equation 7 and Equation 8.
5. The parties locally compute $\llbracket \tilde{W}[j] \rrbracket(u) = \llbracket W[j] \rrbracket(u) + \llbracket a[j] \rrbracket \cdot V(u)$ where $V(u)$ is the vanishing polynomial over $\{f_1, \dots, f_{n_1}\}$ defined as $V(u) := \prod_{i=1}^{n_1} (u - f_i)$.
6. The parties invoke the hint oracle $\mathcal{O}_H$ to get sharings $\llbracket a[1] \rrbracket$, $\dots$, $\llbracket a[n_1] \rrbracket$ and $\llbracket Q' \rrbracket$, for random $a[1], \dots, a[n_1] \in \mathbb{F}_{q^\eta}$ and Q' satisfying
$$u \cdot Q'(u) + q_0 := \sum_{j=1}^{n_2} \tilde{W}[j](u) \cdot X[j](u)$$
for some $q_0 \in \mathbb{F}_{q^\eta}$ and with $\tilde{W}[j](u) = W[j](u) + a[j] \cdot V(u)$ for every $j \in [1 : n_2]$.
7. The parties locally compute $\llbracket q_0 \rrbracket = (n_1)^{-1} \Big(z - \sum_{i=1}^{n_1} f_i \cdot \llbracket Q' \rrbracket(f_i)\Big)$.

⇒ **Since they know that $z = \sum_{i=1}^{n_1} q_0 + f_i \cdot Q(f_i)$, the parties just need to check that $q_0 + u \cdot Q'(u) = \sum_{j=1}^{n_2} X[j](u) \cdot \tilde{W}[j](u)$:**

8. The parties invoke the randomness oracle $\mathcal{O}_R$ to get a random $r \in \mathbb{F}_{q^\eta} \backslash \{0, \dots, n_1 - 1\}$.
9. The parties locally compute $\llbracket \alpha[j] \rrbracket = \llbracket \tilde{W}[j] \rrbracket(r)$ for all $j \in [1 : n_2]$.
10. The parties broadcast $\llbracket \alpha[j] \rrbracket$ and publicly recompute $\alpha[j] \in \mathbb{F}_{q^\eta}$ for all $j \in [1 : n_2]$.
11. The parties locally compute $\llbracket v \rrbracket = r \cdot \llbracket Q' \rrbracket(r) + \llbracket q_0 \rrbracket - \sum_{j=1}^{n_2} \alpha[j] \cdot \llbracket X[j] \rrbracket(r)$.
12. The parties broadcast $\llbracket v \rrbracket$ and publicly recompute v.
13. The parties outputs Accept if $v = 0$ and Reject otherwise.

Fig. 1: The MQOM MPC protocol.

We have

$$\begin{aligned}
\sum_{i=1}^{n_1} P(f_i) &= \sum_{i=1}^{n_1} f_i \cdot Q'(f_i) + (n_1 \cdot q_0) \\
&\quad - \sum_{j=1}^{n_2} \sum_{i=1}^{n_1} W\texttt{[}j\texttt{]}(f_i) \cdot X\texttt{[}j\texttt{]}(f_i) \\
&= \sum_{i=1}^{n_1} f_i \cdot Q'(f_i) + \left(z - \sum_{i=1}^{n_1} f_i \cdot Q'(f_i) \right) \\
&\quad - \sum_{j=1}^{n_2} \sum_{i=1}^{n_1} W\texttt{[}j\texttt{]}(f_i) \cdot X\texttt{[}j\texttt{]}(f_i) \\
&\qquad \text{(by definition of } q_0\text{)} \\
&= \sum_{i=1}^{n_1} f_i \cdot Q'(f_i) + \left(z - \sum_{i=1}^{n_1} f_i \cdot Q'(f_i) \right) \\
&\quad - \langle w, x \rangle \\
&\qquad \text{(by definition of } X \text{ and } W\text{)} \\
&= z - \langle w, x \rangle.
\end{aligned}$$

Since $\langle w, x \rangle \neq z$, we have that $\sum_{i=1}^{n_1} P(f_i) \neq 0$ which implies that P is not the null polynomial. Then, according to the Schwartz-Zippel Lemma, the probability that $v := P(r)$ is zero is at most $p_2 = \frac{2n_1 - 1}{q^\eta - n_1}$.

To sum up, we get

$$\begin{aligned}
\Pr[\textsc{Accept}] &\leq \Pr[\langle w, x \rangle = z] \\
&\quad + \Pr[\langle w, x \rangle \neq z] \cdot \Pr[\textsc{Accept} \mid \langle w, x \rangle \neq z] \\
&= p_1 + (1 - p_1) \cdot p_2
\end{aligned}$$

3 The MQOM Signature Scheme

3.1 From MPC to signature

To obtain the MQOM signature scheme, we first apply the MPCitH transform to the MQOM MPC protocol. This gives us a zero-knowledge (ZK) proof of knowledge (PoK) protocol with a soundness error of about $1/N$ for N being the number of parties (which is slightly degraded by the false positive probability p of the MPC protocol). To make the soundness error negligible, we rely on parallel repetitions of this protocol. We use seed trees to optimize the communication [19] and the hypercube technique to optimize the computation [1]. Finally, to turn the obtained sound ZK-PoK protocol into a signature scheme, we use the Fiat-Shamir transform. Those different steps are summarized hereafter.

The MPCitH transform. The MPC-in-the-Head paradigm turns the MQOM MPC protocol into a ZK-PoK with the following blueprint:

1. *Sharing and commitments:* the prover generates a sharing of the witness $[\![x]\!]$ and separately commits to each share;
2. *First challenge:* the verifier challenges the prover with the random coefficients $\gamma_1, \ldots, \gamma_m$ of the MPC protocol (in place of the randomness oracle $\mathcal{O}_R$);
3. *Hint and commitments:* the prover computes the hint $(a\texttt{[1]}, \ldots, a\texttt{[}n_2\texttt{]}, Q')$ as described in Section 2, generates a sharing of the hint $([\![a\texttt{[1]}]\!], \ldots, [\![a\texttt{[}n_2\texttt{]}]\!], [\![Q']\!])$ and commits to each share separately (*i.e.* the i-th shares $[\![\cdot]\!]_i$ of the different elements are committed all together while the shares of different indices are committed separately);
4. *Second challenge:* the verifier challenges the prover with the random verification point r of the MPC protocol (in place of the randomness oracle $\mathcal{O}_R$);
5. *MPC simulation:* the prover runs the MPC protocol *in their head* to compute the shares broadcasted by the parties, $[\![\alpha\texttt{[1]}]\!]$, …, $[\![\alpha\texttt{[}n_2\texttt{]}]\!]$, $[\![v]\!]$, and sends them to the verifier;
6. *Third challenge:* the verifier challenges the prover to open the views of the parties of indices $[1 : N] \setminus \{i^*\}$;
7. *Views opening:* the prover returns the shares $[\![x]\!]_i$, $[\![a\texttt{[1]}]\!]_i$, …, $[\![a\texttt{[}n_2\texttt{]}]\!]_i$, $[\![Q']\!]_i$ for every $i \in [1 : N] \setminus \{i^*\}$;

   NB: The above shares are called the input shares (witness share and hint shares) of the party i. From the input shares of the party i, and given the previously received broadcast shares, the verifier can now fully recompute the view of party i, for every $i \in [1 : N] \setminus \{i^*\}$.
8. *Verification:* the verifier checks the consistency of the commitments and the MPC computation for the revealed parties. Namely they:
   – verify the commitments of the opened witness shares,
   – verify the commitments of the opened hint shares,
   – recompute the broadcast shares for each party $i \in [1 : N] \setminus \{i^*\}$ and verify that they match the received broadcast shares from the prover,
   – verify that the plain broadcast value v equals 0.

MPCitH optimization: Seed tree. As suggested in [19], we use a seed tree (a.k.a. a puncturable PRF) to generate and commit the shares. The principle is to derive the input shares (witness and hint shares) of each party i pseudorandomly from a random seed seed_i, with the following pattern:

$$\begin{array}{llll}
\mathsf{seed}_1 \longrightarrow & [\![x]\!]_1, & [\![a]\!]_1, & [\![Q']\!]_1 \\
\vdots & \vdots & \vdots & \vdots \\
\mathsf{seed}_{N-1} \longrightarrow & [\![x]\!]_{N-1}, & [\![a]\!]_{N-1}, & [\![Q']\!]_{N-1} \\
\mathsf{seed}_N \longrightarrow & & [\![a]\!]_N &
\end{array}$$

where $[\![a]\!]_i = ([\![a\texttt{[1]}]\!]_i, \ldots, [\![a\texttt{[}n_2\texttt{]}]\!]_i)$.

For the last party, only the share $[\![a]\!]_N$ can be derived from the seed seed_N. This is because $a = \sum_{i=1}^{N} [\![a]\!]_i$ is uniformly distributed. The shares $[\![x]\!]_N$ and $[\![Q']\!]_N$ are then computed as

$$[\![x]\!]_N = x - \sum_{i=1}^{N-1} [\![x]\!]_i \quad \text{and} \quad [\![Q']\!]_N = Q' - \sum_{i=1}^{N-1} [\![Q']\!]_i \ .$$

These shares are called the *auxiliary values* of the input sharings. In this paradigm, opening the input shares of the parties in the set $[1 : N] \setminus \{i^*\}$ simply consists in revealing the seeds $\{\mathsf{seed}_i\}_{i \in [1:N] \setminus \{i^*\}}$ and the auxiliary values $([\![x]\!]_N, [\![Q']\!]_N)$ if $i^* \neq N$.

To further enable a compact opening of $N-1$ seeds out of N, we rely on a *tree PRG* (a.k.a. seed tree). The N seeds are generated from a common *root seed* rseed by

$$\{\mathsf{seed}_i\}_{i\in[1:N]} \leftarrow \mathrm{TreePRG}(\mathsf{rseed}) \ .$$

The principle is to label the root of a binary tree of depth $\lceil \log_2 N \rceil$ with $\mathsf{rseed} = \mathsf{seed}_1^{(0)}$. Then, one inductively labels the children of each level-ℓ node with the output of a PRG applied to the node's label. For MQOM, this PRG is simply a hash function (denoted Hash_4) which takes a 2λ-bit salt salt and the node index i as auxiliary input:

$$\left(\mathsf{seed}_{2i-1}^{(\ell+1)} \parallel \mathsf{seed}_{2i}^{(\ell+1)}\right) \leftarrow \mathrm{Hash}_4\left(\mathsf{salt} \parallel i \parallel \mathsf{seed}_i^{(\ell)}\right)$$

where the level $\ell = 0$ corresponds to the root and the level $\ell = \log_2 N$ corresponds to the leaves, the N seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_N$ that are used for the shares. To reveal all the seeds but seed_{i^*}, one can reveal the sibling labels of the path from the root rseed to the leaf seed_{i^*}, which we denote:

$$\mathsf{path}_{i^*} \leftarrow \mathrm{GetSiblingPath}(\mathsf{rseed}, i^*) \ .$$

For λ-bit seeds, path_{i^*} is composed of $\log_2 N$ labels of λ bits (assuming N is a power of 2). This means that the seeds $\{\mathsf{seed}_i\}_{i\in[1:N]\setminus\{i^*\}}$ can be opened by sending $\lambda \cdot \log_2 N$ bits instead of $\lambda(N-1)$ bits.

Let Commit denote a commitment function (which definition is specified later on). Using the seed tree optimization the commitments of the shares are done as follows:

- In Step 1 (*Sharing and commitments*), the tree PRG is used the generate the seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_N$. The shares are committed by sending $\mathsf{com}_i = \mathrm{Commit}(\mathsf{seed}_i)$ for every $i \in [1 : N-1]$ and $\mathsf{com}_N = \mathrm{Commit}(\mathsf{seed}_N \parallel [\![x]\!]_N)$ to the verifier.
- In Step 3 (*Hint and commitments*), the prover has received the coefficients $\gamma_1, \ldots, \gamma_m$ and is now able to compute Q'. All the shares $[\![Q']\!]_1, \ldots, [\![Q']\!]_{N-1}$ (which are derived from the seeds) are already committed and the prover only has to compute $[\![Q']\!]_N$ and commit it by sending $\mathsf{com}'_N = \mathrm{Commit}([\![Q']\!]_N)$ to the verifier.

MPCitH optimization: Hypercube party computation. We use the hypercube optimization from [1] to reduce the number of party computations performed in Step 5 (*MPC simulation*) of the above MPCitH transform, from N down to $\log_2 N + 1$.

The principle is to exploit the linearity of the MPC computation to batch it into groups of parties. Denoting $[\![\mathsf{in}]\!]_i$ the input share (witness and hint share) of party i, it computes a broadcast share $[\![\mathsf{broad}]\!]_i = \varphi([\![\mathsf{in}]\!]_i) = [\![\varphi(\mathsf{in})]\!]_i$. We then have that for any set of parties $I \subseteq [1 : N]$, we can perform one party computation on $\sum_{i\in I} [\![\mathsf{in}]\!]_i$ to get the sum of broadcast shares $\sum_{i\in I} [\![\mathsf{broad}]\!]_i$. Now let us consider a partition $I(0) \cup I(1) = [1 : N]$ of the parties. With two batched computations we get two sums of broadcast shares, which sum up to the plain broadcast value

$$\sum_{i\in I(0)} [\![\mathsf{broad}]\!]_i + \sum_{i\in I(1)} [\![\mathsf{broad}]\!]_i = \mathsf{broad} \ . \qquad (12)$$

Assume the prover solely sends those two sums of broadcast shares, for some sets $|I(0)| = |I(1)| = N/2$, instead of the N broadcast shares. In case of cheating (*i.e.* at least one party broadcast is inconsistent), the malicious prover is discovered whenever the cheating party is not in the same set as i^* (the non-opened party), which happens with probability 1/2. Repeating this with another partition $I'(0) \cup I'(1) = [1:N]$, one can obtain a soundness error of $\frac{1}{2} \times \frac{1}{2}$ provided that $I(b) \cap I'(b') = N/4$ for every $b, b' \in \{0,1\}$. Repeating this $\log_2 N$ times, we obtain the hypercube optimization.

To get a working sequence of $D = \log_2 N$ partitions, one can see each party as a vertex $i \equiv (i_1, \ldots, i_D)$ in a hypercube of dimension D (where $(i_1, \ldots, i_D)$ is the binary representation of i). Let us define, for every $d \in [1:D]$ and $b \in \{0,1\}$,

$$I(d,b) = \left\{ i \equiv (i_1, \ldots, i_{d-1}, b, i_{d+1}, \ldots, i_D) \mid i_1, \ldots, i_D \in \{0,1\} \right\} . \tag{13}$$

Each pair $(I(d,0), I(d,1))$ is a partition of the vertices belonging to two opposite faces of the hypercube. During Step 5 (*MPC simulation*) of the above MPCitH transform, the prover shall only compute and send:

– the sum of broadcast shares $\sum_{i \in I(d,0)} [\![\mathsf{broad}]\!]_i$ for every $d \in [1:D]$,
– the plain broadcast value broad.

This way the prover only performs $D + 1 = \log_2 N + 1$ party computations (one of them being computed on the plain values).

Then in Step 8 (*Verification*) of the MPCitH transform, the verifier checks the consistency of the broadcast shares for every $d \in [1:D]$ as follows:

– If $i^* \in I(d,1)$, the verifier knows all the input shares $[\![\mathsf{in}]\!]_i$ for $i \in I(d,0)$. They recompute the sum $\sum_{i \in I(d,0)} [\![\mathsf{broad}]\!]_i$ by applying the party computation to $\sum_{i \in I(d,0)} [\![\mathsf{in}]\!]_i$, and check its consistency to the sum previously sent by the prover.
– If $i^* \in I(d,0)$, the verifier knows all the input shares $[\![\mathsf{in}]\!]_i$ for $i \in I(d,1)$. They first recompute the sum $\sum_{i \in I(d,1)} [\![\mathsf{broad}]\!]_i$ by applying the party computation to $\sum_{i \in I(d,1)} [\![\mathsf{in}]\!]_i$ and then recover $\sum_{i \in I(d,0)} [\![\mathsf{broad}]\!]_i$ using Equation 12 (since they further received the plain broadcast broad from the prover).

This way the verifier only performs $D = \log_2 N$ party computations.

Parallel repetition. After applying the MPCitH transform to the MQOM MPC protocol we obtain a ZK-PoK for the MQ problem with soundness error

$$\varepsilon = \frac{1}{N} + p\big(1 - \frac{1}{N}\big)$$

where $p = p_1 + (1 - p_1) \cdot p_2$ is the false positive probability given by Theorem 1.

In order to scale this to $2^{-\lambda}$ for a target security level λ, we use parallel repetition. This means that the ZK-PoK is repeated τ times in parallel to reach a global soundness error of ε^τ. We stress that a soundness error of ε^τ does not imply an unforgeability of ε^τ once the scheme is made non-interactive using the Fiat-Shamir transform. While fixing the number of repetitions τ to achieve some level of security in the non-interactive setting, one needs to take into account possible forgery attacks such as the one described in Section 4.2.

The Fiat-Shamir transform. The Fiat-Shamir heuristic turns the latter ZK-PoK into the MQOM signature scheme. The principle is to replace the verifier challenge of the ZK-PoK with the outputs of hash functions taking the previous prover's communication as input.

Specifically:

1. A first hash h_1 is derived by hashing the share commitments of Step 1 (over all the τ repetitions). This hash is then pseudorandomly expanded into the challenge of Step 2 (different coefficients $\gamma_1, \ldots, \gamma_m$ for the τ repetitions).
2. A second hash h_2 is derived by hashing h_1 together with the hint auxiliary share commitments of Step 3 (over all the τ repetitions). This hash is then pseudorandomly expanded into the challenge of Step 4 (different evaluation points r for the τ repetitions).
3. A third hash h_3 is derived by hashing h_2 together with the broadcast values sent by the prover (over all the τ repetitions). This hash is then pseudorandomly expanded into the challenge of Step 6 (different non-opened party indices i^* for the τ repetitions).

The above hash computations take further inputs:

– *Message:* We introduce the message in the hash computation to obtain a message-binding signature. We choose to introduce the message in the third hash h_3 only. This enables message-independent pre-computation of Steps 1-to-5, which are the computationally-expansive steps of the signing algorithm.
– *Salt:* For security reasons (*i.e.* to avoid possible collisions), we introduce a salt of 2λ bits. This salt is further passed as an argument of the tree PRG (in each node) and the commitments of the shares. The salt is added to the signature to allow the verification.
– *Public key:* We also introduce the public key in the first hash h_1. This makes any forgery attempt specific to a given user, hence preventing adversarial strategies that would invest in heavy precomputation to ease forgery for all (or many of) the users.

Key generation. The key generation of MQOM consists in pseudorandomly generating an MQ instance, with triangular matrices A_i. It takes as input a root seed $\mathsf{seed}_{\mathsf{root}}$ from which it derives two further seeds $\mathsf{seed}_{\mathsf{x}}$ and $\mathsf{seed}_{\mathsf{eq}}$. The secret witness x is derived from $\mathsf{seed}_{\mathsf{x}}$ while the MQ equations $\{A_i, b_i\}$ are derived from $\mathsf{seed}_{\mathsf{eq}}$. The MQ output y is then computed from $\{A_i, b_i\}$ and x. Finally, the key pair is defined and returned as $pk := (\mathsf{seed}_{\mathsf{eq}}, y)$ and $sk := (\mathsf{seed}_{\mathsf{eq}}, y, x)$ with x and y in serialized form.

3.2 Description of the signature scheme

The high-level description of the MQOM signature scheme is wrapped up in Figure 2. All the pseudorandomness in the MQOM signature scheme is derived from an extendable output hash function (XOF). The commitment function Commit is defined as a hash function Hash_0 with input prefix the salt salt, the execution index e and the party index i.

4 Security

4.1 Proof of unforgeability

The MQOM signature scheme aims at providing *unforgeability against chosen message attacks* (EUF-CMA). In this setting, the adversary is given a public key *pk* and they can ask an oracle (called the *signature oracle*) to sign messages $(msg_1, \ldots, msg_r)$ that they can select at will. The goal of the adversary is to generate a pair (msg, σ) such that msg is not one of the requests to the signature oracle and such that σ is a valid signature of msg with respect to pk.

Our security statement is based on the following assumptions:

Input: A secret key $sk := (\mathsf{seed}_{\mathsf{eq}}, y, x)$ and a message $msg \in \{0,1\}^*$.

Phase 0: Initialization.

1. Expand MQ equations $\{A_i, b_i\}_{i\in[1:m]} \leftarrow \mathrm{XOF}(\mathsf{seed}_{\mathsf{eq}})$.
2. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
3. Sample a master seed $\mathsf{mseed} \leftarrow \{0,1\}^{\lambda}$.
4. Expand mseed as τ root seeds $\{\mathsf{rseed}^{[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(\mathsf{salt}, \mathsf{mseed})$.

Phase 1: Sharing and commitments. For each repetition $e \in [1:\tau]$,

1. Generate the leaf seeds $\{\mathsf{seed}_i^{[e]}\}_{i\in[1:N]} \leftarrow \mathrm{TreePRG}(\mathsf{salt}, \mathsf{rseed}^{[e]})$.
2. Randomly generate the shares $(\llbracket x^{[e]}\rrbracket_i, \llbracket a^{[e]}\rrbracket_i, \llbracket Q'^{[e]}\rrbracket_i) \leftarrow \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$ for $i \in [1:N-1]$.
3. Randomly generate the share $\llbracket a^{[e]}\rrbracket_N \leftarrow \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_N^{[e]})$.
4. Compute the last share of the witness $\llbracket x^{[e]}\rrbracket_N = x - \sum_{i=1}^{N-1} \llbracket x^{[e]}\rrbracket_i$.
5. Compute the commitments
   - $\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \mathsf{seed}_i^{[e]})$ for $i \in [1:N-1]$,
   - $\mathsf{com}_N^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, N, \mathsf{seed}_N^{[e]}, \llbracket x^{[e]}\rrbracket_N)$.

Phase 2: First challenge.

1. Compute $h_1 = \mathrm{Hash}_1(pk := (\mathsf{seed}_{\mathsf{eq}}, y), \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \dots, \mathsf{com}_N^{[\tau]})$.
2. Expand h_1 as the coefficients $\{\gamma_1^{[e]}, \dots, \gamma_m^{[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_1)$.

Phase 3: Hint and commitments. For each repetition $e \in [1:\tau]$,

1. Compute the hint $Q'^{[e]}$ from x, $\{A_i, b_i\}_{i\in[1:m]}$, $(\gamma_1^{[e]}, \dots, \gamma_m^{[e]})$ and $a^{[e]} = \sum_{i=1}^{N} \llbracket a^{[e]}\rrbracket_i$.
2. Compute the last share of the hint $\llbracket Q'^{[e]}\rrbracket_N = Q'^{[e]} - \sum_{i=1}^{N-1} \llbracket Q'^{[e]}\rrbracket_i$.
3. Compute the commitment $\mathsf{com}_N'^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, 0, \llbracket Q'^{[e]}\rrbracket_N)$.

Phase 4: Second challenge.

1. Compute $h_2 = \mathrm{Hash}_2(\mathsf{salt}, h_1, \mathsf{com}_N'^{[1]}, \mathsf{com}_N'^{[2]}, \dots, \mathsf{com}_N'^{[\tau]})$.
2. Expand h_2 as the evaluation points $\{r^{[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_2)$.

Phase 5: MPC simulation. For each repetition $e \in [1:\tau]$,

1. Apply the MPC computation $\mathsf{in}^{[e]} := (x, a^{[e]}, Q'^{[e]}) \mapsto \mathsf{broad}^{[e]} := (\alpha^{[e]}, v^{[e]})$.
2. For $d \in [1:D]$, apply the MPC computation $\llbracket \mathsf{in}^{[e]}\rrbracket_{I(d,0)} \mapsto \llbracket \mathsf{broad}^{[e]}\rrbracket_{I(d,0)}$ (where $\llbracket\cdot\rrbracket_{I(d,0)} = \sum_{i\in I(d,0)} \llbracket\cdot\rrbracket_i$).

Phase 6: Third challenge.

1. Compute $h_3 = \mathrm{Hash}_3\left(msg, \mathsf{salt}, h_2, \{\mathsf{broad}^{[e]}\}_{e\in[1:\tau]}, \left\{\llbracket \mathsf{broad}^{[e]}\rrbracket_{I(d,0)}\right\}_{d\in[1:D], e\in[1:\tau]}\right)$.
2. Expand h_3 as the non-opened view indices $\{i^{*[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_3)$.

Phase 7: Views opening. For each repetition $e \in [1:\tau]$,

1. Compute the sibling path of the non-opened seed $\mathsf{path}^{[e]} \leftarrow \mathrm{GetSiblingPath}(\mathsf{salt}, \mathsf{rseed}^{[e]}, i^{*[e]})$.
2. Derive the opened views as
   - $\mathsf{view}^{[e]} = (\mathsf{path}^{[e]}, \llbracket x^{[e]}\rrbracket_N, \llbracket Q'^{[e]}\rrbracket_N)$ if $i^{*[e]} \in [1:N-1]$,
   - $\mathsf{view}^{[e]} = \mathsf{path}^{[e]}$ if $i^{*[e]} = N$.

Phase 8: Signature.

1. Return the signature $\sigma := \left(\mathsf{salt} \parallel h_1 \parallel h_2 \parallel h_3 \parallel \{\mathsf{view}^{[e]}, \mathsf{broad}^{[e]}, \mathsf{com}_{i^{*[e]}}^{[e]}\}_{e\in[1:\tau]}\right)$.

Fig. 2: MQOM – Signing algorithm.

```
INPUT: A public key $pk := (\mathsf{seed}_{\mathsf{eq}}, y)$, a message $msg \in \{0,1\}^*$ and a signature $\sigma$.

Phase 0: Parsing and expansion.

1. $\Big(\mathsf{salt} \parallel h_1 \parallel h_2 \parallel h_3 \parallel \{\mathsf{view}^{[e]}, \mathsf{broad}^{[e]}, \mathsf{com}^{[e]}_{i^{*[e]}}\}_{e\in[1:\tau]}\Big) \leftarrow \sigma$.
2. Expand MQ equations $\{A_i, b_i\}_{i\in[1:m]} \leftarrow \mathrm{XOF}(\mathsf{seed}_{\mathsf{eq}})$.
3. Expand $h_1$ as the coefficients $\{\gamma_1^{[e]}, \ldots, \gamma_m^{[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_1)$.
4. Expand $h_2$ as the evaluation points $\{r^{[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_2)$.
5. Expand $h_3$ as the non-opened view indices $\{i^{*[e]}\}_{e\in[1:\tau]} \leftarrow \mathrm{XOF}(h_3)$.
6. For each repetition $e \in [1:\tau]$, parse the opened views as
     – $(\mathsf{path}^{[e]}, [\![x^{[e]}]\!]_N, [\![Q'^{[e]}]\!]_N) = \mathsf{view}^{[e]}$ if $i^{*[e]} \in [1:N-1]$,
     – $\mathsf{path}^{[e]} = \mathsf{view}^{[e]}$ if $i^{*[e]} = N$.

Phase 1: Recomputing shares and commitments. For each repetition $e \in [1:\tau]$,

1. Recover the open seeds from the sibling path $\{\mathsf{seed}_i^{[e]}\}_{i\in[1:N]} \leftarrow \mathrm{TreePRG}(\mathsf{salt}, \mathsf{path}^{[e]})$.
2. For $i \in [1:N-1] \setminus \{i^{*[e]}\}$,
     – Regenerate the shares $([\![x^{[e]}]\!]_i, [\![a^{[e]}]\!]_i, [\![Q'^{[e]}]\!]_i) \leftarrow \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$ for $i \in [1:N-1]$.
     – Recompute the commitments $\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \mathsf{seed}_i^{[e]})$.
3. If $i^{*[e]} \neq N$,
     – Regenerate the shares $[\![a^{[e]}]\!]_N \leftarrow \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_N^{[e]})$.
     – Recompute the commitment $\mathsf{com}_N^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, N, \mathsf{seed}_N^{[e]})$.
     – Recompute the commitment $\mathsf{com}_N'^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, 0, [\![Q'^{[e]}]\!]_N)$.
   NB: If one of these commitments is incorrect, the recomputed hashes $(h_1', h_2')$ won't match
   $(h_1, h_2)$ in Phase 3 below.

Phase 2: MPC simulation. For each repetition $e \in [1:\tau]$,

1. For $d \in [1:D]$, s.t. $i^{*[e]} \in I(d,1)$,
     – Apply the MPC computation $[\![\mathsf{in}^{[e]}]\!]_{I(d,0)} \mapsto [\![\mathsf{broad}^{[e]}]\!]_{I(d,0)}$.
2. For $d \in [1:D]$, s.t. $i^{*[e]} \in I(d,0)$,
     – Apply the MPC computation $[\![\mathsf{in}^{[e]}]\!]_{I(d,1)} \mapsto [\![\mathsf{broad}^{[e]}]\!]_{I(d,1)}$.
     – Recover $[\![\mathsf{broad}^{[e]}]\!]_{I(d,0)} = \mathsf{broad}^{[e]} - [\![\mathsf{broad}^{[e]}]\!]_{I(d,1)}$.
   with $[\![\cdot]\!]_{I(d,b)} = \sum_{i\in I(d,b)} [\![\cdot]\!]_i$, $[\![\mathsf{in}^{[e]}]\!]_i := ([\![x]\!]_i, [\![a^{[e]}]\!]_i, [\![Q'^{[e]}]\!]_i)$, $[\![\mathsf{broad}^{[e]}]\!]_i := ([\![\alpha^{[e]}]\!]_i, [\![v^{[e]}]\!]_i)$.
   NB: If one of these sums of broadcast shares is incorrect, the recomputed hash $h_3'$ won't match
   $h_3$ in Phase 3 below.

Phase 3: Recomputing hashes.

1. Compute $h_1' = \mathrm{Hash}_1(pk := (\mathsf{seed}_{\mathsf{eq}}, y), \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_N^{[\tau]})$.
2. Compute $h_2' = \mathrm{Hash}_2(\mathsf{salt}, h_1, \mathsf{com}_N'^{[1]}, \mathsf{com}_N'^{[2]}, \ldots, \mathsf{com}_N'^{[\tau]})$.
3. Compute $h_3' = \mathrm{Hash}_3\Big(msg, \mathsf{salt}, h_2, \{\mathsf{broad}^{[e]}\}_{e\in[1:\tau]}, \big\{[\![\mathsf{broad}^{[e]}]\!]_{I(d,0)}\big\}_{d\in[1:D], e\in[1:\tau]}\Big)$.

Phase 4: Verification.

1. For $e \in [1:\tau]$: If $\mathsf{broad}^{[e]} = (\alpha^{[e]}, v^{[e]})$ with $v^{[e]} \neq 0$, then return REJECT.
2. If $(h_1, h_2, h_3) \neq (h_1', h_2', h_3')$, then return REJECT.
3. Else return ACCEPT.
```

Fig. 3: MQOM – Verification algorithm.

– **MQ hardness.** Solving the considered MQ instance is $(\epsilon_{\text{MQ}}, t)$-*hard* for some $(\epsilon_{\text{MQ}}, t)$ which are implicit functions of the security parameter λ. Formally, any adversary $\mathcal{A}$ on input a random MQ instance $\big(\{A_i\}, \{b_i\}, y\big)$ and running in time at most t has probability at most ϵ_{MQ} to output the solution x of the input instance.
– **Secure pseudorandomness.** The pseudorandomness generated by the extendable output hash function (XOF) is indistinguishable from true randomness (provided that the input seed has sufficient entropy). Formally, an adversary $\mathcal{A}$ running in time at most t has advantage at most ϵ_{PRG} in distinguishing the two following distributions:

$$\mathcal{D}_0 = \{x \leftarrow \{0,1\}^{\ell_{\max}}\} \text{ and } \mathcal{D}_1 = \{x \in \{0,1\}^{\ell_{\max}} \leftarrow \text{XOF}(\mathsf{seed}) \mid \mathsf{seed} \leftarrow \mathcal{S}\}$$

for any distribution $\mathcal{S}$ with at least λ bits of min-entropy and where $\ell_{\max}$ denotes the maximum number of bits sampled from XOF with overwhelming probability (*i.e.* ignoring negligible occurrence of long sequences due to rejection sampling).
– **ROM.** The hash functions Hash_0, Hash_1, Hash_2, Hash_3, Hash_4 behave as random oracles. Formally, our security statement only holds in the random oracle model where the Hash_i's are modeled as random oracles.

Under the above security assumptions, we get the following result:

Theorem 2. *Let* Hash_0*,* Hash_1*,* Hash_2*,* Hash_3*,* Hash_4 *be modelled as random oracles, and let* p_1*,* p_2 *be the probabilities from Theorem 1. Let* $\mathcal{A}$ *be an adversary against the EUF-CMA security of the scheme running in time* t *and making a total of* Q_i *queries to* Hash_i *for* $i \in \{0,1,2,3,4\}$ *and* Q_{sign} *queries to the signing oracle. Let further denote* Q *the total number of random oracle queries, which is*

$$Q = Q_0 + Q_1 + Q_2 + Q_3 + Q_4 + N_{Hash} \cdot Q_{Sig}$$

where $N_{Hash} = \tau(2N+1) + 3$ *is the total number of hash calls (or random oracle queries) in a signature computation. Under the above assumptions,* $\mathcal{A}$*'s advantage in the EUF-CMA game is upper-bounded as*

$$\epsilon_{\text{EUF-CMA}} \leq \frac{(\tau N + 2)Q^2}{2^{2\lambda}} + \epsilon_{\text{MQ}} + \epsilon_{\text{PRG}} + \Pr[X + Y + Z = \tau] \ ,$$

with

– $X = \max_{i \in [1:Q_1]}\{X_i\}$ *with* $X_i \sim \mathcal{B}(\tau,\, p_1)$*,*
– $Y = \max_{i \in [1:Q_2]}\{Y_i\}$ *with* $Y_i \sim \mathcal{B}(\tau - X,\, p_2)$*,*
– $Z = \max_{i \in [1:Q_3]}\{Z_i\}$ *with* $Z_i \sim \mathcal{B}\big(\tau - X - Y,\, \frac{1}{N}\big)$*,*

where $\mathcal{B}(n_0, p_0)$ *denotes the binomial distribution with* n_0 *the number of trials and* p_0 *the success probability of each trial.*

The following proof of Theorem 2 is adapted from the security proof of the Banquet signature scheme [2] (for the main part, while the end of the proof is adapted from [15]).

Proof. We first prove the security of MQOM against unforgeability with key only (EUF-KO) and then show how this translates to standard EUF-CMA security. For the sake of simplicity, we first assume that an MQOM public key is an instance $\big(\{A_i\}, \{b_i\}, y\big)$ (namely we ignore the pseudorandom generation from $\mathsf{seed_{eq}}$) and that the outputs from XOF are truly random values. We relax these assumptions at the end of the proof.

EUF-KO security. Assume an EUF-KO adversary $\mathcal{A}$ exists which runs in time t and outputs a valid forgery with probability $\epsilon_{\text{EUF-KO}}$. We construct an algorithm $\mathcal{R}$ (the reduction) solving MQ in time t and probability ϵ_{MQ} from $\mathcal{A}$.

The reduction $\mathcal{R}$ receives an MQ challenge $\big(\{A_i\}, \{b_i\}, y\big)$ and aims to find the solution x. It interacts with $\mathcal{A}$ by simulating the EUF-KO game in which the public key is defined as the MQ challenge $pk := \big(\{A_i\}, \{b_i\}, y\big)$. The reduction answers the hash queries of $\mathcal{A}$ with random values and maintains four lists $\mathcal{Q}_1$, $\mathcal{Q}_0$, $\mathcal{Q}_2$, $\mathcal{Q}_3$, $\mathcal{Q}_4$ such that $(q, h) \in \mathcal{Q}_i$ iff q has been queried to Hash_i and the random value h was picked and answered to this query. $\mathcal{R}$ also maintains a list Bad of "bad answers" to hash queries. Whenever a random oracle is queried on a new input q, a random answer h is sampled. If $h \in \mathsf{Bad}$, then $\mathcal{R}$ aborts, otherwise $\mathcal{R}$ adds h to Bad. Additionally:

– for any query
$$q_1 = (pk, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_N^{[\tau]}) \ ,$$
to Hash_1, $\mathcal{R}$ adds all the digests $\mathsf{com}_i^{[e]}$ to Bad;

– for any query
$$q_2 = (\mathsf{salt}, h_1, \mathsf{com}_N'^{[1]}, \mathsf{com}_N'^{[2]}, \ldots, \mathsf{com}_N'^{[\tau]}) \ ,$$
to Hash_2, $\mathcal{R}$ adds h_1 and all the digests $\mathsf{com}_N'^{[e]}$ to Bad;

– for any query
$$q_3 = (msg, \mathsf{salt}, h_2, \ldots) \ ,$$
$\mathcal{R}$ adds h_2 to Bad.

Then we get

$$\begin{aligned}
\Pr[\mathcal{R} \text{ aborts}] &= (\#\text{ times a digest is sampled}) \cdot \Pr[\mathcal{R} \text{ aborts at that digest}] \\
&\leq Q_{\text{RO}} \cdot \frac{\max |\mathsf{Bad}|}{2^{2\lambda}} \\
&= Q_{\text{RO}} \cdot \frac{Q_0 + (\tau N + 1)Q_1 + (N+2)Q_2 + 2Q_3}{2^{2\lambda}} \\
&\leq \frac{(\tau N + 1)Q_{\text{RO}}^2}{2^{2\lambda}} \qquad (14)
\end{aligned}$$

where $Q_{\text{RO}} = Q_0 + Q_1 + Q_2 + Q_3 + Q_4$.

In addition, $\mathcal{R}$ maintains associative arrays $\mathcal{T}_{\mathsf{sh}}$ and $\mathcal{T}_{\mathsf{x}}$ to keep track of the shares (and seeds) and candidate witnesses it gets from $\mathcal{A}$'s queries. Specifically, for each query

$$q_1 = (pk, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_N^{[\tau]}) \ ,$$

and for each $e \in [1:\tau]$, $\mathcal{R}$ checks whether

– $\forall i \in [1 : N-1]$: $\exists (q_0^{(e,i)}, h_0^{(e,i)}) \in \mathcal{Q}_0$ s.t.
$$q_0^{(e,i)} = (\mathsf{salt}, e, i, \mathsf{seed}_i^{[e]})$$
and $h_0^{(e,i)} = \mathsf{com}_i^{[e]}$,

– $\exists(q_0^{(e,N)}, h_0^{(e,N)}) \in \mathcal{Q}_0$ s.t.

$$q_0^{(e,N)} = (\mathsf{salt}, e, N, \mathsf{seed}_N^{[e]}, [\![x^{[e]}]\!]_N)$$

and $h_0^{(e,N)} = \mathsf{com}_N^{[e]}$.

If so, $\mathcal{R}$ defines

$$\begin{aligned}\mathcal{T}_{\mathsf{sh}}[q_1, e, i] &:= ([\![x^{[e]}]\!]_i, [\![a^{[e]}]\!]_i, [\![Q'^{[e]}]\!]_i) \\ &= \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_i^{[e]})\end{aligned}$$

for every $i \in [1 : N-1]$,

$$\mathcal{T}_{\mathsf{sh}}[q_1, e, N] := ([\![x^{[e]}]\!]_N, [\![a^{[e]}]\!]_N)$$

with $[\![a^{[e]}]\!]_N = \mathrm{XOF}(\mathsf{salt}, \mathsf{seed}_N^{[e]})$ and with $[\![x^{[e]}]\!]_N$ from $q_0^{(e,N)}$, and

$$\mathcal{T}_{\mathsf{x}}[q_1, e] := \sum_{i=1}^{N} [\![x^{[e]}]\!]_i \ .$$

Upon reception of a valid message-signature pair from $\mathcal{A}$, $\mathcal{R}$ checks $\mathcal{T}_{\mathsf{x}}$ for possible candidate solutions to the MQ challenge. If one of the $\mathcal{T}_{\mathsf{x}}[q_1, e]$ stores a correct solution x to the MQ challenge $\big(\{A_i\}, \{b_i\}, y\big)$, then $\mathcal{R}$ returns x, otherwise $\mathcal{R}$ returns $\perp$. We have

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ ouputs } \perp] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{R} \text{ ouputs } x] \\ &\leq \Pr[\mathcal{R} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ ouputs } \perp] + \Pr[\mathcal{R} \text{ ouputs } x]\end{aligned}$$

which is

$$\epsilon_{\text{EUF-KO}} \leq \Pr[\mathcal{R} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ ouputs } \perp] + \epsilon_{\text{MQ}} \ . \tag{15}$$

$\Pr[\mathcal{R}$ aborts$]$ is upper bounded by Equation 14. It remains to upper bound $\Pr[\mathcal{A}$ wins $\mid \mathcal{R}$ ouputs $\perp]$.

We now analyze the probability of $\mathcal{A}$ winning the EUF-KO game conditioned on the event that $\mathcal{R}$ outputs $\perp$, *i.e.*, that no suitable solution x was found from the hash queries.

Cheating in the first round. For any query $(q_1, h_1) \in \mathcal{Q}_1$, and its corresponding expanded answer $\{\gamma_1^{[e]}, \dots, \gamma_m^{[e]}\}_{e \in [1:\tau]} = \mathrm{XOF}(h_1)$, let $G_1(q_1, h_1)$ be the set of indices $e \in [1 : \tau]$ of "good executions" where both $\mathcal{T}_{\mathsf{x}}[q_1, e] = x^{[e]}$ is non-empty and the batched relation (Equation 1) holds for $x^{[e]}$ and the $\{\gamma_i^{[e]}\}_i$. Since $\mathcal{R}$ outputs $\perp$, $x^{[e]}$ is not the right solution to the MQ challenge. Then by the uniformity of the $\{\gamma_i^{[e]}\}_i$ (for now, XOF is assumed to be a truly random function), each $e \in [1 : \tau]$ has the same independent probability of being in $G_1(q_1, h_1)$. According to Theorem 1, this probability is $p_1 = \frac{1}{q^\eta}$. We therefore have that $\#G_1(q_1, h_1) \sim X_{q_1}$ where $X_{q_1} = \mathcal{B}(\tau, p_1)$, the binomial distribution with τ trials, each with success probability p_1. Letting $(q_{1\mathsf{best}}, h_{1\mathsf{best}})$ denote the query-response pair which maximizes $\#G_1(q_1, h_1)$, we then have that

$$\#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}}) \sim X = \max_{(q_1,h_1) \in \mathcal{Q}_1} \{X_{q_1}\}.$$

Cheating in the second round. For any query $(q_2, h_2) \in \mathcal{Q}_2$, and its corresponding expanded answer $\{r^{[e]}\}_{e \in [1:\tau]} = \mathrm{XOF}(h_2)$, we define $G_2(q_2, h_2)$ the set of indices $e \in [1 : \tau]$ of "good executions" as follows. Let

$$q_2 = (\mathsf{salt}, h_1, \mathsf{com}'^{[1]}_N, \mathsf{com}'^{[2]}_N, \dots, \mathsf{com}'^{[\tau]}_N) \ .$$

Then $e \in G_2(q_2, h_2)$ if the following conditions hold:

1. $\exists\, q_1 = (pk, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_N^{[\tau]})$ such that $(q_1, h_1) \in \mathcal{Q}_1$ with same salt as in q_2;
2. $\mathcal{T}_{\mathsf{x}}[q_1, e] = x^{[e]}$ is non-empty;

   At this stage (from the above condition), we have that $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] = ([\![x^{[e]}]\!]_i, [\![a^{[e]}]\!]_i, [\![Q'^{[e]}]\!]_i)$ is non-empty for every $i \in [1 : N-1]$.
3. $\forall i \in [1:N]$, $\exists\, q_0^{[e]} = (\mathsf{salt}, e, 0, [\![Q'^{[e]}]\!]_N)$ such that $(q_0^{[e]}, \mathsf{com}'^{[e]}_N) \in \mathcal{Q}_0$;

   Let $Q'^{[e]} = \sum_{i=1}^{N} [\![Q'^{[e]}]\!]_i$. Let $w^{[e]}$ and $z^{[e]}$ be defined from $x^{[e]}$ and the $\{\gamma_i^{[e]}\}_i$ (according to the description of Section 2.3). And let $P^{[e]}$ the polynomial defined from $Q'^{[e]}$, $x^{[e]}$, $w^{[e]}$ and $z^{[e]}$ as in the proof of Theorem 1.
4. The random point $r^{[e]}$ is such that $P^{[e]}(r^{[e]}) = 0$.

If $e \in G_1(q_1, h_1)$ then all the above conditions are fulfilled (in particular, we have $z^{[e]} = \langle x^{[e]}, w^{[e]} \rangle$ implying $P^{[e]} = 0$) and hence $e \in G_2(q_2, h_2)$ as well. Whenever $e \notin G_1(q_1, h_1)$, we have $P^{[e]} \neq 0$ and $P^{[e]}(r^{[e]}) = 0$ with probability $p_2 = \frac{2n_1 - 1}{q^\eta - n_1}$ over the randomness of $r^{[e]}$ (which holds from the Schwartz-Zippel Lemma – see the proof of Theorem 2). Namely, every "bad execution vs. round 1", $e \in [1:\tau] \setminus G_1(q_1, h_1)$, has the same independent probability p_2 of being in $G_2(q_2, h_2)$ (*i.e.* of being a "good execution vs. round 2"). We therefore have that

$$\#\big(G_2(q_2, h_2) \setminus G_1(q_1, h_1)\big) \sim \mathcal{B}\left(\tau - \tau_1, p_2\right)$$

where $\tau_1 = \#G_1(q_1, h_1)$. By taking $\tau_1 = X$, the random variable maximized by $(q_{1\mathsf{best}}, h_{1\mathsf{best}})$ and defining $(q_{2\mathsf{best}}, h_{2\mathsf{best}})$ the query-response pair which maximizes $\#G_2(q_2, h_2)$, we then have that

$$\#G_2(q_{2\mathsf{best}}, h_{2\mathsf{best}}) \sim X + Y$$

where

$$Y = \max_{(q_2, h_2) \in \mathcal{Q}_2} \{Y_{q_2}\} \quad \text{with} \quad Y_{q_2} \sim \mathcal{B}\left(\tau - X, p_2\right) \ .$$

Cheating in the third round. Each hash query

$$q_3 = (msg, \mathsf{salt}, h_2, \ldots)$$

that $\mathcal{A}$ makes to Hash_3 can only be used in a winning signature if there exists a corresponding query $(q_2, h_2) \in \mathcal{Q}_2$. Then for each "bad" second-round execution $e \in [1:\tau] \setminus G_2(q_2, h_2)$, either the verification protocol failed, in which case $\mathcal{A}$ could not have won, or the verification protocol passed, despite $P^{[e]}(r^{[e]}) \neq 0$. This implies that exactly one of the parties must have cheated during the MPC execution of the verification protocol. (Less than one and the verification protocol would have failed; more than one and the verification of the signature would have failed.) Since the third-round challenge $\{i^{*[e]}\}_{e \in [1:\tau]} = \mathrm{XOF}(h_3)$ is distributed uniformly at random (assuming XOF is a random function), the probability that this happens for all such "bad" second-round executions e is

$$\left(\frac{1}{N}\right)^{\tau - \#G_2(q_2, h_2)} \leq \left(\frac{1}{N}\right)^{\tau - (X+Y)} \ .$$

The probability that this happens for at least one of the Q_3 queries made to Hash_3 then satisfies

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{R} \text{ outputs } \perp] \leq \Pr[X + Y + Z = \tau] \tag{16}$$

with Z the random variable defined as $Z = \max_{i \in [1:Q_3]}\{Z_i\}$ for $Z_i \sim \mathcal{B}\big(\tau - X - Y,\ \frac{1}{N}\big)$.

Wrapping up. From Equation 14, Equation 15 and Equation 16, we finally get

$$\epsilon_{\text{EUF-KO}} \leq \frac{(\tau N + 1)Q_{\text{RO}}^2}{2^{2\lambda}} + \epsilon_{\text{MQ}} + \Pr[X + Y + Z = \tau] \ . \tag{17}$$

EUF-CMA security. We consider the above reduction algorithm $\mathcal{R}$ which we extend to answer the signing queries of the adversary $\mathcal{A}$. We consider two games:

- Game 1: $\mathcal{R}$ uses a signature oracle $\mathcal{O}_{\text{Sig}}(sk, pk, \cdot)$ which perfectly answers signing queries from $\mathcal{A}$;
- Game 2: the signature oracle is replaced by a simulator $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ answering signing queries from $\mathcal{A}$ without being given the secret key as input.

In Game 1, $\mathcal{R}$ behaves exactly as above while additionally answering the signing queries from $\mathcal{A}$ using $\mathcal{O}_{\text{Sig}}(sk, pk, \cdot)$. The signing oracle $\mathcal{O}_{\text{Sig}}(sk, pk, \cdot)$ makes queries to the random oracles Hash_0, Hash_1, Hash_2, Hash_3 and Hash_4 which are answered by $\mathcal{R}$ as above, and it computes the signature from the input message and the key pair (sk, pk) as described in Figure 2. The total number of random oracle queries is hence of

$$Q = Q_{\text{RO}} + N_{\text{Hash}} \cdot Q_{\text{Sig}}$$

where $Q_{\text{RO}} = Q_0 + Q_1 + Q_2 + Q_3 + Q_4$ is the total number of random oracle queries made by $\mathcal{A}$ (*i.e.* outside the signature queries) and $N_{\text{Hash}} = \tau(2N + 1) + 3$ is the total number of hash calls (or random oracle queries) in a signature computation ($\tau(N + 1)$ calls to Hash_0, τN calls to Hash_4, and one call to each Hash_1, Hash_2 and Hash_3).

The signing queries being perfectly answered, $\mathcal{A}$ produces a valid signature with probability $\epsilon_{\text{EUF-CMA}}$. Then, following the same lines as the EUF-KO proof, $\mathcal{R}$ interacting with $\mathcal{A}$ and $\mathcal{O}_{\text{Sig}}(sk, pk, \cdot)$ recovers the MQ solution x (part of sk) with probability ϵ_{Game1} such that

$$\epsilon_{\text{EUF-CMA}} \leq \epsilon_{\text{Game1}} + \frac{(\tau N + 1)Q^2}{2^{2\lambda}} + \Pr[X + Y + Z = \tau] \ .$$

We note that this is insufficient to prove our security statement since the reduction $\mathcal{R}$ should not have access to the oracle $\mathcal{O}_{\text{Sig}}(sk, pk, \cdot)$, which is why we need to transit to Game 2.

Game 2 is similar to Game 1 but the signing oracle is replaced by a simulator $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ which does not take the secret key sk as input. Additionally, $\mathcal{R}$ keeps a list $\mathcal{S}l$ of all the salts appearing in random oracle queries. The signing simulator $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ starts by picking a random salt value (as in a genuine signature generation) and aborts if the generated value is already in $\mathcal{S}l$. This way, all the seeds generated by the tree PRG can be considered as fresh random values independent of any previous oracle responses. The signing simulator follows the same principle as the honest-verifier zero-knowledge simulator of the MQOM ZK-PoK (*i.e.* the proof-of-knowledge obtained when applying the MPCitH transformation to the MQOM MPC protocol). By knowing the challenges beforehand, it can generate a signature with perfect distribution without knowing the secret key. In the random oracle model, this simply means that $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ randomly generates the answers h_1, h_2 and h_3 of the oracles Hash_1, Hash_2 and Hash_3 before they are actually queried in the signature generation. From these hashes, $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ deduces the challenges by application of the XOF. Then it generates the shares as in a genuine signature generation, with the following differences:

1. x is randomly drawn (and is hence an incorrect solution to the MQ instance with overwhelming probability);
2. for every execution $e \in [1:\tau]$, $Q'^{[e]}$ is generated as a random degree-$(2n_1-2)$ polynomial for which the MPC computation $(x, a^{[e]}, Q'^{[e]}) \mapsto (\alpha^{[e]}, v^{[e]})$ is such that $v^{[e]} = 0$.

The shares revealed in the signature are uniformly distributed under the constraint $v^{[e]} = 0$, namely they are identically distributed to those of a genuine signature. In the absence of abortion, we thus get that an answer of $\mathcal{S}_{\text{Sig}}$ on input (pk, msg) is identically distributed to an answer of $\mathcal{O}_{\text{Sig}}$ on input (sk, pk, msg) for any message msg.

The probability of abortion due to collisions in Bad is the same for $\mathcal{O}_{\text{Sig}}$ and $\mathcal{S}_{\text{Sig}}$. Indeed, they both do the same amounts of queries to the random oracles, the simulator just does them in a different order and handles the queries for h_1, h_2 and h_3 directly. $\mathcal{S}_{\text{Sig}}$ may further abort in case of salt collision, which happens with probability at most $(Q_{\text{RO}} + Q_{\text{Sig}})/2^{2\lambda}$. We deduce that the success probability ϵ_{Game2} of $\mathcal{R}$ to recover x while interacting with $\mathcal{A}$ and simulating signing queries with $\mathcal{S}_{\text{Sig}}(pk, \cdot)$ satisfies

$$|\epsilon_{\text{Game1}} - \epsilon_{\text{Game2}}| \leq \frac{Q_{\text{Sig}}(Q_{\text{RO}} + Q_{\text{Sig}})}{2^{2\lambda}} \leq \frac{Q^2}{2^{2\lambda}} \ .$$

which implies

$$\epsilon_{\text{EUF-CMA}} \leq \epsilon_{\text{Game2}} + \frac{(\tau N + 2)Q^2}{2^{2\lambda}} + \Pr[X + Y + Z = \tau] \ .$$

Accounting pseudorandomness. The final step of the proof consists in replacing the idealized XOF, which is assumed to output true randomness, by the real XOF, which is assumed to be $(t, \epsilon_{\text{PRG}})$-indistinguishable from true randomness. We further consider real MQOM key pairs in which the MQ equations $\big(\{A_i\}, \{b_i\}\big)$ are pseudorandomly generated (with XOF) from a random $\mathsf{seed_{eq}}$. Let Game 3 be similar to Game 2 with the real pseudorandomness setting such that $\epsilon_{\text{Game3}} = \epsilon_{\text{MQ}}$ is the probability that $\mathcal{R}$ recovers x while interacting with the adversary $\mathcal{A}$ against the real MQOM scheme. Clearly, we have $|\epsilon_{\text{Game2}} - \epsilon_{\text{Game3}}| \leq \epsilon_{\text{PRG}}$, which finally gives:

$$\epsilon_{\text{EUF-CMA}} \leq \frac{(\tau N + 2)Q^2}{2^{2\lambda}} + \epsilon_{\text{MQ}} + \epsilon_{\text{PRG}} + \Pr[X + Y + Z = \tau] \ .$$

4.2 Signature forgery attacks

While applying the Fiat-Shamir transform to a zero-knowledge proof of knowledge (ZK-PoK) with several rounds and parallel repetitions, one gets a security drop between the soundness of the ZK-PoK and the unforgeability of the signature scheme. Namely, the forgery cost is lower than $\frac{1}{\varepsilon}$, where ε is the soundness error of the original ZK-PoK. The best forgery attack against Fiat-Shamir-based schemes with several parallel executions ($\tau > 1$) is the attack from [18]. The latter is described for a 5-round scheme, but it can be easily generalized for any scheme with more rounds. The attack works as follows:

1. The adversary chooses an attack strategy. It consists in choosing three non-negative integers τ_1, τ_2, τ_3 such that $\tau = \tau_1 + \tau_2 + \tau_3$.
2. For each forgery attempt, they generate cheating commitments (*i.e.* which do not correspond to a correct witness) which for each repetition fool the MPC protocol with probability p_1 in round

1 (commitment – challenge 1) and probability p_2 in round 2 (response – challenge 2), where p_1, p_2 are as defined in Theorem 1. They repeat this step until the MPC protocol is fooled in round 1 for τ_1 repetitions among τ. Since the probability that a cheating commitment fools the MPC protocol w.r.t. a random first challenge (expanded from h_1) is p_1 for one repetition, the probability to get τ_1 fooled repetitions is

$$\begin{aligned}\mathsf{PMF}(\tau, \tau_1, p_1) &:= \Pr\left(\begin{array}{c}\text{fool at least } \tau_1 \\ \text{of the } \tau \text{ repetitions} \\ \text{in round 1 with unitary} \\ \text{probability of } p_1\end{array}\right) \\ &= \sum_{k=\tau_1}^{\tau} \binom{\tau}{k} p_1^k (1-p_1)^{\tau-k} \ .\end{aligned}$$

Thus this step is repeated $\frac{1}{\mathsf{PMF}(\tau,\tau_1,p_1)}$ times on average. We denote $I_1 \subseteq [1:\tau]$ the set of indices of the fooled repetitions for a forgery attempt successfully passing this step.

3. For a forgery attempt successfully passing the previous step, the adversary tries different round-2 responses to the first challenge generated from h_1. Each response attempt gives rise to a different hash h_2 and corresponding random round-2 challenge which can be fooled with probability p_2 for each repetition. This step is repeated until the generated challenge is fooled for τ_2 repetitions among $[1:\tau] \setminus I_1$. The probability of succeeding for one round-2 response attempt is

$$\mathsf{PMF}(\tau-\tau_1, \tau_2, p_2) = \sum_{k=\tau_2}^{\tau-\tau_1} \binom{\tau-\tau_1}{k} p_2^k (1-p_2)^{\tau-\tau_1-k} \ .$$

Thus this step is repeated $\frac{1}{\mathsf{PMF}(\tau-\tau_1,\tau_2,p_2)}$ times on average. We denote $I_2 \subseteq [1:\tau] \setminus I_1$ the set of indices of the fooled repetitions for a forgery attempt successfully passing this step.

4. Finally, the adversary guesses the unopened parties' indexes in round 3 for the remaining repetitions $e \in [1:\tau]\backslash(I_1 \cup I_2)$ (the repetitions which have not been fooled yet). They generate the round-3 responses accordingly, *i.e.* cheating on the guessed non-opened party only and such that the broadcast shares pass the verification (*i.e.* correspond to $v = 0$). They repeat this step until the guess is correct, which happens with probability

$$\left(\frac{1}{N}\right)^{\tau-\tau_1-\tau_2} \ .$$

Thus this last step is repeated $N^{\tau-\tau_1-\tau_2}$ times on average.

The forgery cost of this attack corresponds to the cost of the optimal strategy, namely

$$\mathsf{cost}_{\mathsf{forge}} = \max_{\tau_1,\tau_2,\tau_3 \,:\, \tau=\tau_1+\tau_2+\tau_3} \left\{ \frac{1}{\mathsf{PMF}(\tau,\tau_1,p_1)} + \frac{1}{\mathsf{PMF}(\tau-\tau_1,\tau_2,p_2)} + N^{\tau_3} \right\} \ , \tag{18}$$

where p_1 and p_2 are defined in Theorem 1.

The parameters of MQOM (see Table 3 hereafter) have been chosen such that the associated forgery cost is at least of 2^{128} for Category I, of 2^{192} for Category III and of 2^{256} for Category V defined by NIST [22]. We stress that here a forgery cost of 2^λ means computing more than 2^λ hashes, which is relevant with respect to the definition of Categories I, II and V (assuming that a hash computation is comparable or superior to an AES computation in gate count).

4.3 MQ Security

The security of the MQOM signature scheme relies on the hardness to solve an instance of the multivariate quadratic problem, since the secret key is a solution of the MQ instance represented by the public key. There exist many algorithms to solve the MQ problem. Their complexity depends on several parameters: the number n of unknowns, the number m of quadratic equations, the size q of the field, the characteristic of the field, and the number of solutions. The optimal algorithm might vary depending on the values of these parameters.

We used the MQ estimator [3] to evaluate the computational costs of all the existing algorithms. This tool takes the parameters (q, m, n) of an MQ problem and estimates the running times (and the memory usage) of the most important classical algorithms. The MQ estimator takes two parameters as inputs:

- a real number $w \in [2, 3]$ such that the complexity to multiply two $n \times n$ matrices is $O(n^w)$.
- a real number $\theta \in [0, 2]$ such that $(\log_2 q)^\theta$ is the ratio between the field operation complexity and the bit complexity.

Regarding the complexity of the matrix multiplication, we set w as $\log_2(7)$. The same choice was made in MQ-DSS [9] and in the numerical results of the MQ estimator [3]. While there exist some algorithms with asymptotically smaller w, those algorithms have a huge constant factor. In practice, the best an adversary can hope is $w = \log_2(7)$ obtained using Strassen algorithm [26]. Moreover, we take $\theta = 2$ as suggested in [3].

We only consider the MQ instances for which the number n of unknowns and the number m of equations are the same (*i.e.* $m = n$), since they correspond to the harder instances. Indeed, adding more equations would provide information about the system, while having more variables would enable us to reduce the instances by fixing some of them. Table 1 summarizes the selected MQ parameters for MQOM for the three security categories together with the corresponding MQ estimator output. The choice of these parameters is explained in the next section.

Table 1: Complexities of different attacks on our MQ instances.

Parameter Set	**MQ Parameters**		**Algorithms**		
	q	$m = n$	HybridF5	FXL	Crossbred
MQOM-L1-gf31	31	49	164.2	142.8	143.9
MQOM-L1-gf251	251	43	168.9	144.4	151.6
MQOM-L3-gf31	31	77	244.6	207.6	232.6
MQOM-L3-gf251	251	68	251.4	209.8	262.1
MQOM-L5-gf31	31	106	326.9	276.2	325.8
MQOM-L5-gf251	251	93	334.9	274.1	369.3

5 Choice of the Parameters

The motivation for choosing multiple parameters aiming at different signature sizes and speeds is to cover various use cases. While speed can be the main objective in situations where saving CPU cycles is crucial, shorter signatures are useful when size is a bottleneck. Such use cases involve for instance network communication or very constrained embedded contexts:

- Saving network bandwidth: an example is TLS sessions making use of signatures. When scaling to many servers and clients, any decrease in the signature size will provide a significant decrease in bandwidth (and limited performance impact in networks where many routers only transfer the raw packets in-between the two parties without performing cryptographic computations). The overhead on signature and verification computation is acceptable regarding the economic gain on network pressure.
- Constrained embedded devices: in very small micro-controllers, the flash size is usually limited (a few dozens of kilobytes). When storing e.g. X.509 certificates with embedded signatures, saving bytes in flash at the expense of computation can be necessary. In such cases, computation time is usually less an issue as the stored signature is often used for occasional operations that can be "slow", e.g. firmware verification or local user data authenticity at boot time.

We first fix the base field characteristic, testing values ranging from $q = 17$ to $q = 251$. For each tested q, we took the number of equations m to be equal to the number of unknowns n and selected this parameter to achieve the target level of security (categories I, III and V) according to the *MQ estimator* [3] as explained in Section 4.3.

Regarding the MPC parameters, we first fix the number of parties to $N = 256$ (or equivalently the hypercube dimension $D = 8$) to achieve running times of a few milliseconds while keeping short signatures. Then for each tested q, and given the selected MQ parameters $n = m$, we exhaust the relevant MPC parameters (n_1, n_2, η).

Given $(q, n, m, N, n_1, n_2, \eta)$, we deduce the number of repetitions τ necessary to achieve a forgery cost larger than λ bits, when λ is 128, 192 and 256 respectively for Categories I, III and V. Currently, the best forgery attack is obtained by applying the approach of [18] and its cost is given by Equation 18 (see description Section 4.2). Then from $(q, n, m, N, n_1, n_2, \eta)$ and τ, we deduce the size of a signature. For each q (and associated $n = m$), we keep the MPC parameters (n_1, n_2, η) leading to the shortest signature for $\lambda = 128$. Table 2 summarizes the obtained parameters and sizes for the different tested values of q.

Table 2: Tested fields $\mathbb{F}_q$, corresponding MQ parameters $n = m$, and optimal parameters for MQOM in terms of signature size for $\lambda = 128$ (with $N = 256$).

q	$n = m$	n_1	n_2	η	τ	Size
17	54	5	11	10	20	6 528
19	53	5	11	10	20	6 528
23	51	4	13	10	20	6 489
29	50	5	10	10	20	6 368
31	49	5	10	10	20	6 348
$37 \to 53$	48	4	12	6	23	6 615
$59 \to 61$	47	4	12	6	23	6 615
$67 \to 73$	47	4	12	7	20	6 508
$79 \to 83$	46	4	12	7	20	6 488
$89 \to 127$	45	5	9	6	22	6 640
$131 \to 137$	45	5	9	5	22	6 618
$139 \to 173$	44	4	11	5	22	6 596
$179 \to 251$	43	4	11	5	22	6 575

We chose to propose the instances obtained for $q = 31$ and $q = 251$. The former achieves the shorter signature size for the tested fields. Moreover, an MQ instance with $q = 31$ was previously considered in the MQ-DSS [9] which might already have motivated some cryptanalysis attempts on those parameters. On the other hand, $q = 251$ gives the larger prime field whose elements hold in single bytes. Moreover, it requires a smaller extension degree η than $q = 31$ which makes the underlying arithmetic faster. Such an instance might also be more amenable to future improvements of MPCitH based on threshold secret sharing [15].

Finally, we chose to add a "fast" variant relying on $N = 32$ parties. Compared to the "short" variant with $N = 256$, the "fast" variant is 2 to 3 times faster for the considered instances, for an overhead of ∼20% in the signature size.

The different MQOM instances and their parameters are summarized in Table 3 (together with benchmarks described in the next section).

6 Implementations and Benchmarks

6.1 Signature and keys sizes

The formula for the maximum signature size in bytes depending on the MQ parameters is bounded by (where $|e \in \mathbb{F}_q|$ is the size in bytes of the elements in the field $\mathbb{F}_q$):

$$|\sigma| \;\le\; \lambda + \underbrace{|e \in \mathbb{F}_q| \cdot \tau \cdot \big(n + \eta \cdot (2n_1 + n_2 - 1)\big)}_{\text{Field elements of MPC communication}} + \underbrace{\frac{\tau \cdot \lambda \cdot (\log_2 N + 4)}{8}}_{\text{(de)commitments}} \qquad (19)$$

For $q = 31$, elements in the field are packed on 5 bits, and $|e \in \mathbb{F}_q| = \frac{5}{8}$. For $q = 251$, elements are on 8 bits and $|e \in \mathbb{F}_q| = 1$.

The signature size in Equation 19 splits into two main components: the field elements communicated in the MPC protocol, and the commitments and decommitments (mainly the sibling paths of seed trees). The remaining λ bytes include the salt and three hashes (each of size 2λ bits). To reach a security of λ bits, the number of iterations τ is at least $\lambda/\log_2 N$ and can approach this lower bound whenever the false positive probabilities p_1 and p_2 are small enough (*i.e.* when q and η are high enough). This implies that the (de)commitment part is at least of $\lambda^2/8$ bytes (which is 2 KB for $\lambda = 128$) whatever the value of N. With our chosen parameters for $\lambda = 128$, this part represents ≈ 4 KB for the short instances and ≈ 5 KB for the fast instances. For the MPC communication part, increasing N might result in a logarithmic decrease of τ (as $\tau \ge \lambda/\log_2 N$) but only if it is accompanied by an increase of η which mitigates the gain. With our chosen parameters for $\lambda = 128$, the MPC communication part represents ≈ 2.5 KB for the short instances and ≈ 2.7 KB for the fast instances.

From Section 3, the sizes in bytes of the secret key sk and the public key pk are:

$$|sk| = \frac{\lambda}{8} + |e \in \mathbb{F}_q| \cdot m$$
$$|pk| = |sk| + n$$

This results in 125 bytes for sk (218 bytes for pk) for MQOM-L5-gf251, the largest of our MQOM variants.

Table 3: Different MQOM instances, underlying MQ and MPC parameters and benchmarks, for NIST Security Categories I, III, and V. Benchmarks are provided for signing and verification in milliseconds (ms) and millions of cycles (Mc), on an `AVX2` Intel Core i7-10700K at 3.8 GHz platform. For each category, the shortest and fastest instances are highlighted.

MQOM Variants	**NIST Security**		**MQ Parameters**		**MPC Parameters**					**Sig. size (Bytes)**		**Sig. perf.**		**Verif. perf.**	
	Category	Bits	q	$m = n$	$N = 2^D$	n_1	n_2	η	τ	Avg.	Max.	Time (ms)	Cycles (Mc)	Time (ms)	Cycles (Mc)
MQOM-L1-gf31-short	I	143	31	49	256	5	10	10	20	6348	6352	11.7	44.3	11.0	41.7
MQOM-L1-gf31-fast	I	143	31	49	32	5	10	6	35	7621	7657	4.6	17.6	4.1	15.5
MQOM-L1-gf251-short	I	143	251	43	256	4	11	5	22	6575	6578	7.5	28.5	7.2	27.3
MQOM-L1-gf251-fast	I	143	251	43	32	4	11	4	34	7809	7850	3.0	11.5	2.7	10.2
MQOM-L3-gf31-short	III	207	31	77	256	6	13	11	30	13837	13846	28.5	108.1	27	102.2
MQOM-L3-gf31-fast	III	207	31	77	32	6	13	7	51	16590	16669	14.8	56.3	13.5	51.2
MQOM-L3-gf251-short	III	207	251	68	256	5	14	7	30	14257	14266	18.3	69.5	17.3	65.5
MQOM-L3-gf251-fast	III	207	251	68	32	5	14	4	52	17161	17252	8.6	32.8	7.8	29.6
MQOM-L5-gf31-short	V	272	31	106	256	6	18	10	42	24147	24158	59.2	224.4	56.3	213.6
MQOM-L5-gf31-fast	V	272	31	106	32	6	18	8	66	28917	29036	41.2	156.2	38.5	146.2
MQOM-L5-gf251-short	V	272	251	93	256	6	16	7	41	24926	24942	39.0	148.0	37.5	142.2
MQOM-L5-gf251-fast	V	272	251	93	32	6	16	5	66	29919	30092	21.5	81.5	19.9	75.6

6.2 Computation overview

On the performance side, signature and verification computations are very similar as one can observe from Figure 2 and Figure 3: both compute almost identical sharings and commitments, both perform the same MPC simulation, and both make almost the same number of XOF expansions and hash calls (the verification being slightly more efficient as it avoids some prover's plain and hint computations). One major element to notice is that the core MPC computation mainly scales with the MQ problem parameters, the base field q, and the number of variables/equations $n = m$, as well as the MPC parameters (n_1, n_2, η), and only logarithmically scales with the number of parties N. On the other hand, the *sharing and commitments* phase linearly scales with the number of parties N. For the "fast" variants from Table 3, decreasing N drastically reduces the timings of the MPC computation, explaining the 2 to 3 improvement factor. The signature size, on the other hand, remains contained while switching from $N = 256$ to $N = 32$. When N grows, the bottleneck of the whole signature and verification algorithms becomes the *sharing and commitments* phase (as confirmed by the benchmarks in Section 6.3) which is mainly composed of XOF and hash primitives.

The key pair generation time is very fast and negligible when compared to signing and verification, as the computation merely consists of a small set of calls to XOF (scaling with n and m).

6.3 Implementation and optimizations

The implementation of the variants of MQOM makes use of optimized implementations over finite fields with precomputations where possible. Field elements on 5 bits ($q = 31$) or 8 bits ($q = 251$) allow operations over native `uint8_t` (for the elements in "uncompressed" form), with modular operations and overflows handling on larger words: the number of reductions modulo q can be optimized to absorb (unreduced) additions and multiplications in chains. All the mathematical objects are represented as multi-dimensional arrays of field elements: this is obvious for vectors and matrices, and extension fields η are vectors of base field elements. Computations over these extension fields are also simplified whenever possible: the Lagrange interpolation of the X and W polynomials can be reduced to simple matrix multiplications over the base field. The MQ equations computation cost is halved by sampling triangular matrices for the $\{A_i\}_{i\in[1:m]}$ during key generation without affecting the distribution of the MQ instance. For the MPC protocol, some elements that appear multiple times throughout various phases can be mutualized: this is among other things the case for $\sum_{i=1}^{m} \gamma_i A_i$ used during hint, broadcast and party computations.

XOF is used for pseudorandomness generation during the expansion of MQ equations, MPC view-opening, seeds and shares. Rejection sampling is performed for field elements generation with a $\frac{1}{32}$ (resp. $\frac{5}{256}$) rejection rate when $q = 31$ (resp. $q = 251$), which keeps the sampling loop reasonable. Hash is used for commitments, the Fiat-Shamir transform, and the node derivation of the seed trees.

For XOF and hashing, we provide the different primitives that are used on Table 4: SHA3 and SHAKE are both based on Keccak and allow for a simplified code as the same basis is used for both. This allows for optimization factoring as any improvement on the underlying Keccak implementation would benefit the two primitives. For MQOM, we use two implementations from the XKCP library [4]: a portable pure C code (for the portable reference implementations), and an assembly-optimized implementation using Intel's `AVX2` 256-bit SIMD instructions (allowing to parallelize 4 instances of the primitive in one call for a speedup of a factor 3 over the basic `x86_64` assembly code).

Table 4: Symmetric cryptography primitives for NIST Security Categories I, III, and V.

	Category I	Category III	Category V
Hash	SHA3-256	SHA3-384	SHA3-512
XOF	SHAKE-128	SHAKE-256	SHAKE-256

Finally, the message is only introduced in the third hash meaning that Phases 0 to 5 of the signing algorithm (see Figure 2) and Phases 0 to 2 of the verification algorithm (see Figure 3) are message-independent. This means that for a given MQOM key pair, message-independent phases can be precomputed, leaving a very marginal computation whenever the message is available (one hash and derivation of sibling paths for the signature).

6.4 Benchmarks

Table 3 provides benchmarks of our implementation of MQOM compiled for an `AVX2` platform (an Intel Core i7-10700K at 3.8 GHz). All the code is in C except for the assembly-optimized SIMD SHA3 and SHAKE. As expected, reducing N from 256 to 32 for our "fast" variants provides a significant speedup over the "short" variants while keeping reasonable signature sizes: MQOM-L1-gf251-fast achieves around 3 ms for signing and verifying for ≈ 7.8 kilobytes signatures.

A detailed analysis of the signing execution trace shows that 70% of the time is spent in Phase 1 (see Figure 2) for MQOM-L1-gf251-fast, and 85% for MQOM-L1-gf251-short: this exhibits XOF/hashing bottleneck (for pseudorandomness and commitments) and the strong dependency with the number of parties N. The MPC emulation part is quite marginal as it takes only 15% (resp. 6%) of the MQOM-L1-gf251-fast computation (resp. MQOM-L1-gf251-short). As expected, the phases that only depend on the message take less than 0.05% of the time, meaning that signing (or verification) with message-independent precomputation shall only take microseconds.

Finally, key pair generation takes 0.1 (resp. 1.6) millisecond for MQOM-L1-gf251-fast (resp. MQOM-L5-gf31-short), showcasing a marginal cost in all the instances as expected.

6.5 Possible improvements

For now, our implementation only makes use of assembly `AVX2` SIMD for the mere hash and XOF primitives. Although further optimizations on the Keccak core will probably be marginal, achieving any speedup in the C code using these primitives should provide clear impacts on performance. A drastic improvement for MQOM would obviously also come from dedicated hardware accelerated instructions for XOF and hash (e.g. the `ARMv8.2` specification has dedicated Keccak SIMD extensions, implemented in CPUs such as Apple's A13 SoC).

The MPC repetition factor τ makes MQOM highly parallelizable, either using a multi-threading paradigm or vectorization of the instances through SIMD: this should improve all the instances' speed. Inside the repetitions, further parallelism can be additionally applied to the parties, seed trees and commitments computations.

Finally, field computations are in pure C and performed over `uint8_t` in native machine words: these could benefit from optimized assembly improvements (possibly making use of parallelization with SIMD), mostly speeding up the "fast" variants where the XOF and hash bottlenecks are less present.

7 Comparison

In this section, we compare MQOM to other multivariate signature schemes from the literature. An important property of MQOM is to rely on the non-structured MQ problem which is conservative in terms of (post-quantum) security. Table 5 lists all the signature schemes based on the non-structured MQ problem in the current state of the art. All these schemes are obtained through the Fiat-Shamir transform applied to a ZK-PoK for the MQ problem. We can observe that MQOM achieves the shortest signature sizes among them. In particular, MQOM signatures are about 10% shorter than the previous shortest due to [12] while based on more conservative MQ parameters. MQOM signatures are more than 30% shorter than the other non-structured MQ signatures in the state of the art.

Let us further compare MQOM with the other multivariate signature schemes submitted to the NIST call for additional post-quantum signatures [22]. Most of them rely on the hash-and-sign paradigm and have small signature sizes (less than 500 bytes for the first security level):

- either they are close variants of the UOV signature scheme [20] and thus have a large public key (tens of kilobytes),
- or they have competitive public key size but rely on new structured MQ assumptions that have not yet passed the test of time (and are hence less conservative in terms of security).

Excluding MQOM, the only multivariate scheme built from a Fiat-Shamir transform in the new NIST call is Biscuit [5]. It achieves smaller sizes (4.8–6.7 KB) but relies on a structured variant of the MQ problem which is arguably less conservative than the original problem.

Table 5: MQOM – Comparison with the other signature schemes built as the Fiat-Shamir transformation of a zero-knowledge proof for the MQ problem.

Schemes	MQ Parameters		MQ Security	Communcation Cost		Running Times	
	q	$m = n$	(in bits)	Public key	Signature Size	Signing	Verification
MQ-DSS [8]	31	48	141	46 B	28400 B	5.5 Mc	3.6 Mc
MudFish [6]	4	88	149	38 B	14400 B	14.8 Mc	15.3 Mc
Mesquite [27] – Fast	4	88	149	38 B	9492 B	15.4 Mc	12.1 Mc
Mesquite [27] – Compact	4	88	149	38 B	8844 B	30.7 Mc	24.4 Mc
Fen22-gf251 [12] – Fast	251	40	135	56 B	8488 B	8.3 Mc	-
Fen22-gf251 [12] – Short	251	40	135	56 B	7114 B	22.8 Mc	-
MQOM-L1-gf251 – Fast	251	43	144	59 B	7809 B	11.5 Mc	10.16 Mc
MQOM-L1-gf251 – Short	251	43	144	59 B	6575 B	28.5 Mc	27.3 Mc
MQOM-L1-gf31 – Fast	31	49	143	47 B	7621 B	17.7 Mc	15.5 Mc
MQOM-L1-gf31 – Short	31	49	143	47 B	6348 B	44.4 Mc	41.7 Mc

References

1. Carlos Aguilar Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 564–596. Springer, Heidelberg, April 2023. 2, 9, 11
2. Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Heidelberg, May 2021. 2, 4, 16
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, *AFRICACRYPT 22*, volume 2022 of *LNCS*, pages 323–347. Springer Nature, July 2022. 23, 24
4. Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. eXtended Keccak Code Package , 2023. `https://github.com/XKCP/XKCP`. 27
5. Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit: Shorter MPC-based Signature from PoSSo, 2023. `https://www.biscuit-pqc.org/`. 28
6. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020. 2, 29
7. Ward Beullens. MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 355–376. Springer, Heidelberg, September / October 2022. 2
8. Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass MQ-based identification to MQ-based signatures. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 135–165. Springer, Heidelberg, December 2016. 29
9. Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. MQDSS specifications. Version 2.1, 4 2020. `https://mqdss.org/files/mqdssVer2point1.pdf`. 23, 25
10. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 3022–3036. ACM Press, November 2021. 2, 4
11. Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 843–857. ACM Press, November 2022. 2
12. Thibauld Feneuil. Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. Cryptology ePrint Archive, Report 2022/1512, 2022. `https://eprint.iacr.org/2022/1512`. 2, 4, 28, 29
13. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 541–572. Springer, Heidelberg, August 2022. 1, 2

14. Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 371–402. Springer, Heidelberg, December 2022. 2
15. Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of MPC-in-the-head. Cryptology ePrint Archive, Report 2022/1407, 2022. `https://eprint.iacr.org/2022/1407`. 2, 3, 16, 25
16. Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS$^+$. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. 1
17. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007. 2
18. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2020. 21, 24
19. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018. 2, 9, 10
20. Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 206–222. Springer, Heidelberg, May 1999. 1, 28
21. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. 1
22. National Institute of Standards and Technology (NIST). Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process, 2022. `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`. 1, 22, 28
23. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. 1
24. Simona Samardjiska, Ming-Shing Chen, Andreas Hulsing, Joost Rijneveld, and Peter Schwabe. MQDSS. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`. 2
25. Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 13–21. Springer, Heidelberg, August 1994. 1
26. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 354-356(13), 8 1969. `https://doi.org/10.1007/BF02165411`. 23
27. William Wang. Shorter signatures from MQ. Cryptology ePrint Archive, Report 2022/344, 2022. `https://eprint.iacr.org/2022/344`. 29
28. Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`. 2