Differential Fault Attacks on MQOM
Breaking the Heart of Multivariate Evaluation

Vladimir Sarde[1,2], Nicolas Debande[1]

[1] Cryptography & Security Group, IDEMIA Secure Transactions, Pessac, France
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
vladimir.sarde@idemia.com, nicolas.debande@idemia.com

Keywords: MQOM · MPC-in-the-Head · Post-Quantum Signature · Multivariate Cryptography · PIOP · Fault Attack

Abstract. MQOM is one of the fourteen remaining candidates in the second round of the NIST post-quantum signature standardization process. Introduced in 2023, MQOM instantiates the Multi-Party Computation in the Head (MPCitH) paradigm over the well-established hard problem of solving Multivariate Quadratic (MQ) equations. In this paper, we present the first fault attacks on MQOM targeting the MQ evaluation phase, which is a central component of the algorithm. We introduce four differential fault attacks and demonstrate their effectiveness against both unprotected and masked implementations. The first two target the secret key using a random fault model, making them particularly realistic and practical. With as little as one or two injected faults, depending on the variant, the entire secret key can be recovered through linear algebra. The other two attacks exploit faults on the coefficients of the MQ system directly. Our results highlight that the MQ evaluation, despite not being identified as a sensitive component until now, can be exploited using just a few fault injections.

1 Introduction

In 2016, the National Institute of Standards and Technology (NIST) announced a competition for the selection of post-quantum signature schemes. This competition led to the selection of three schemes: Dilithium [14], Falcon [20], and SPHINCS$^+$ [9]. Dilithium and SPHINCS$^+$ have since been standardized, with minor modifications, as ML-DSA in [29] and as SLH-DSA [30]. A draft standard based on Falcon is also expected to be released soon. In 2023, the NIST opened a second call to explore different types of hard problems for securing cryptographic schemes. Among the 40 submitted candidates, 9 are based on the MPCitH framework. Later, the NIST announced the second round of this second call, further confirming the growing interest in this framework, as 6 out of the 14 selected candidates are based on MPCitH. The main difference between these 6 schemes lies in the hard problem on which their security is based on.

Indeed, Mirith [2] is based on the MinRank problem, PERK [10] on the Permuted Kernel problem, RYDE [11] on the Rank Syndrome Decoding problem, SDitH [3] on the Syndrome Decoding problem, FAEST [6] on breaking AES, and MQOM [8], which will be the focus of our study, on the Multivariate Quadratic (MQ) Problem. Finally, it is worth noting that this interest for the MPCitH framework is also reflected in other post-quantum cryptographic competitions. For instance, the Korean post-quantum Cryptography (KpqC [1]) competition recently selected AIMer [26] as one of its two signature algorithms.

The MPCitH paradigm is a versatile framework introduced by [24]. This paradigm enables the design of signature schemes for a given hard problem, potentially post-quantum, by allowing the choice of the one-way function on which the security is based. The principle of the MPCitH paradigm has remained the same, but its exact form has evolved significantly over time. The original scheme was first introduced in [24]. The framework was later popularized by Picnic [13], a candidate in NIST's first post-quantum cryptography standardization call. It reached the final round of the competition as an alternate candidate before being eliminated. Since then, successive improvements have significantly enhanced the performance of these signature schemes. A notable example is the Threshold Computations in the Head (TCitH) framework [17], introduced in 2022, which demonstrates that leveraging Shamir's Secret Sharing (SSS) can reduce significantly the number of required computations. In parallel, an improvement, called hypercube, in the sharing method one year before the end of NIST's second call was proposed in [3]. This method provides such a performance boost that the majority of candidates submitted the following year, including MQOM, adopted this approach. On a third front, a variant named VOLE-in-the-Head (VOLEitH) [6] was also developed and then used within the FAEST scheme [5]. Between the two rounds, a second variant of TCitH was described in [16], proposing a novel proof system denoted as Π_{PC}. Note also that the VOLEitH framework can be seen as a very close variant of the TCitH Π_{PC} framework [16, Sect. 6]. In the case of MQOM, both the TCitH Π_{PC} and the VOLEitH frameworks reduce computational costs and roughly cut the signature size in half, while maintaining the scheme's security compared to the initial submission. As a result, the authors integrated these frameworks into the second-round specification of MQOM, submitted to the NIST signature competition [7]. Consequently, our work is based on this specification. However, this scheme is described neither within the formalism of the TCitH framework nor that of VOLEitH, but rather using a new formalism known as the *Polynomial Interactive Oracle Proof* (PIOP) and the (zero-knowledge) *Polynomial Commitment Scheme* (PCS). This formalism simplifies the description of the framework compared to the original MPCitH. It is nonetheless equivalent to the QuickSilver protocol [31] in the VOLE-in-the-Head setting and to the Π_{PC} proof scheme in the TCitH framework.

The MQOM scheme relies on the one-way function that consists in evaluating a MQ system, avoiding the need for trapdoors thanks to the MPCitH paradigm. Therefore, its security stems solely from symmetric primitives and the MQ problem known to be NP-hard (see [21]). This makes it a conservative scheme as MQ

problem has been extensively studied and has served as the basis for public-key cryptographic schemes such as [15], [27], [28] since the 1980s.

In our study, we focus on the physical security of MQOM, specifically its resilience to fault attacks. These attacks, first introduced in 1996 by Boneh, DeMillo, and Lipton [12], consist in injecting faults during the execution of a cryptographic algorithm. To induce such faults, an attacker can physically disturb the operation of the device using techniques such as electromagnetic probes or laser injection. By analyzing the faulty outputs produced during the disrupted computation, it may be possible to extract information about the secret key. The security of MQOM against fault attacks has not yet been studied in the literature. In the broader context of side-channel and fault attacks on the MPCitH paradigm, preliminary research has been conducted. The first works, introduced in [22], [4] focus on an earlier version of the paradigm, in the context of the Picnic algorithm. More recently, [23] targets the SDitH signature scheme, using the same TCitH variant as MQOM, with a Soft Analytical Side-Channel Attack (SASCA). Finally, the article [25] presents two power-based side-channel attacks using deep learning, as well as two voltage fault injection on a masked implementation of FAEST. These two fault attacks focus on the PCS component of the paradigm, which is common to almost all MPCitH schemes. To counter previously mentioned side-channel attacks, a widely adopted and effective strategy is to apply masking to each sensitive operation, as described in [18].

Contributions. In this context, we present four differential fault attacks targeting the evaluation of the MQ system. We describe these attacks based on an implementation where the secret key is shared in $t \geq 1$ shares, for greater generality. The first two attacks directly target the masked secret key under a random fault model. Depending on the MQOM variant, they can recover the key with as few as 1 to 2 fault injections, combined with some exhaustive search. In contrast, the other two attacks rely on a stuck-at fault model and aim to disrupt public, unmasked components of the MQ system. These can recover the entire key with the same number of fault injections as there are coefficients in the key, i.e. 160 for the $\mathbb{F}_2$ variants and 48 for the $\mathbb{F}_{256}$ ones.

Roadmap. In Sect. 2, we lay out the notations used, recall the MPCitH framework, and describe the workflow of MQOM. Sect. 3 introduces four attacks, each assuming a specific fault injection, and studies how to perform a secret key recovery. Sect. 4 analyses the complexity of each attack with respect to the parameters of the different MQOM variants. Sect. 5 validates the effectiveness of the proposed attacks paths through simulations. Sect. 6 discusses possible countermeasures. Finally, we draw our conclusion in Sect. 7.

2 Preliminaries

2.1 Notations

Throughout this article, we will write elements e of a field in italic lowercase letters, vectors $\mathbf{v}$ in bold lowercase letters and matrices $\mathbf{M}$ in bold uppercase.

We will denote the transpose of a matrix $\mathbf{A}$ by $\mathbf{A}^T$. Additionally, if a matrix is indexed, say $\mathbf{A}_i$ of size $n \times n$, we will represent the index as a superscript on its coefficients, that is $(a_{jk}^{(i)})_{(j,k)\in[\![1,n]\!]}$.

We will use similar notations for polynomials. Specifically, polynomials will be denoted in italic, and vectors of polynomials, i.e. vectors with polynomial entries, will be denoted in bold uppercase. We will also denote by $\mathbb{K}[X]^{\leq d}$ the set of polynomials in $\mathbb{K}[X]$ of degree less than or equal to d.

We will use $\tilde{v}$ to denote a faulted version of the value v.

2.2 Shamir's Secret Sharing

Shamir's Secret Sharing (SSS) consists of sharing a secret $w \in \mathbb{F}$ among N parties. It uses a polynomial of degree l defined as $P_w(X) = w + \sum_{k=1}^{l} a_k X^k$, where the a_k terms are uniformly distributed. The shares are then defined as $w_j = P_w(e_j)$ where $\{e_j\}_{j\in[\![1,N]\!]}$ is a public set of points in $\mathbb{F}$. Given any set of $l+1$ shares, it is possible to recover the secret by interpolating from the evaluations, and then computing $P_w(0) = w$. Conversely, any set of $l_1 \leq l$ shares reveals no information about the secret.

In the *Polynomial Interactive Oracle Proof* (PIOP) framework, which we describe in Sect. 2.5, the classical SSS scheme is modified by placing the secret in the leading coefficient instead of the constant term. Concretely, the sharing polynomial is given by $P_w(X) = \sum_{k=0}^{l-1} a_k X^k + wX^l$, and we denote $P_w(\infty) = w$. Furthermore, within the PIOP setting, we will only make use of SSS schemes of degree 1 or 2. In particular, the secret key $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is shared coordinate-wise using polynomials $P_{x_i}(X) = (x_0)_i + x_i X$, where the $(x_0)_i$ terms are randomly generated. The resulting vector of polynomials is denoted by $\mathbf{P}_\mathbf{x}(X)$.

2.3 Multivariate Quadratics Problem

The Multivariate Quadratics problem (MQ) of parameters $(\mathbb{F}, m, n)$ consists in solving a given system of $m \in \mathbb{N}^*$ quadratic equations with $n \in \mathbb{N}^*$ unknowns over the finite field $\mathbb{F}$. In formal terms, it consists in finding $\mathbf{x} \in \mathbb{F}^n$, given $\mathbf{A}_i$ uniform in $\mathbb{F}^{n\times n}$, $\mathbf{b}_i$ uniform in $\mathbb{F}^n$ and $y_i = \mathbf{x}^T\mathbf{A}_i\mathbf{x} + \mathbf{b}_i^T\mathbf{x}$ for all $i \in [\![1, m]\!]$.

Note that the matrix $\mathbf{A}_i$ can be generated as a triangular matrix without weakening the problem, since the coefficient of the monomial $x_i x_j$ corresponds to $a_{ij} + a_{ji}$, which also follows a uniform distribution. This enables more efficient generation as well as equation evaluation.

2.4 MPCitH Framework

MPCitH signatures are zero-knowledge proofs of knowledge (ZKPoKs) of a preimage of a one-way function denoted by F. The secret key is thus an element $\mathbf{x}$ from the domain of F, and the public key consists of the instance of F together with $\mathbf{y} = F(\mathbf{x})$. This ZKPoK is built from a specific PIOP. This PIOP is then repeated τ times in parallel to achieve the desired security level.

Finally, the Fiat–Shamir heuristic [19] is used to transform this interactive proof into a signature. This transformation replaces the verifier's direct challenge generation with a hash-based approach, where challenges are computed from a hash of relevant data. This function takes among its inputs the message, which ensures that the signature is bound to it. However, for the sake of clarity, we continue to refer to this entity as the verifier throughout the rest of the article.

Concretely, the signature is defined as the transcript of the interaction between the prover and the verifier. To verify the authenticity of the signature, one simply replays the proof as the verifier, using the same hash function and inputs to generate the challenges. The signature is accepted if the transcript is consistent with the result obtained by replaying the proof.

2.5 PIOP

A *Polynomial Interactive Oracle Proof* (PIOP) is an interactive proof in which the prover is allowed to provide the verifier with polynomial oracles $[P_1, P_2, \ldots, P_n]$ for polynomials $P_1, P_2, \ldots, P_n \in \mathbb{K}$. The verifier is guaranteed that these oracles are correct and can query them to obtain the evaluations $P_1(r), P_2(r), \ldots, P_n(r)$ of the polynomials at any point $r \in \Omega$, where $\Omega = \{\omega_0, \omega_1, \ldots, \omega_{N-1}\} \subset \mathbb{K}$ is a predefined set.

The MQOM signature consists of a ZKPoK based on a PIOP involving 1-degree polynomials. The concrete construction details of the polynomial oracle are not essential for understanding either the signature scheme or the fault attacks presented in this work. However, as some faults may have side effects on this construction, we will briefly describe part of the computations involved, though we will keep the presentation at a high level. More detailed explanations can be found in the specification [7]. The polynomial oracle is replaced by a *Polynomial Commitment Scheme* (PCS). The prover generates a vector $\mathbf{P}_\mathbf{x}(X) \in (\mathbb{K}[X]^{\leq 1})^n$ with random constant terms, such that for each i, the evaluation at infinity satisfies $(\mathbf{P}_\mathbf{x}(\infty))_i = P_{x_i}(\infty) = x_i$. This construction relies on a GGM tree, where each leaf corresponds to a randomly generated vector $\bar{\mathbf{x}}_i$. A correction vector $\boldsymbol{\Delta}_\mathbf{x} = \mathbf{x} - \sum_{i=0}^{N-1} \bar{\mathbf{x}}_i$ is computed. The polynomial is finally constructed and is defined by:

$$\mathbf{P}_\mathbf{x} = \boldsymbol{\Delta}_\mathbf{x}\mathbf{P}_0 + \sum_{i=0}^{N-1} \bar{\mathbf{x}}_i \mathbf{P}_i$$

where $\mathbf{P}_i = X - \omega_i$ and we set $\omega_0 = 0$ by convention. Therefore:

$$\mathbf{P}_\mathbf{x}(X) = \mathbf{x} \cdot X - \sum_{i=0}^{N-1} \omega_i \cdot \bar{\mathbf{x}}_i.$$

After constructing the polynomial, the prover sends a commitment of this tree to the verifier (see [7]). The verifier can then request the evaluation of the polynomials at a point $r \in \Omega$. The prover reveals $N-1$ leaves of the tree by revealing the sibling path of a hidden leaf. By design, this is sufficient to reconstruct

the evaluation at the challenge point r, i.e. $\mathbf{P}_{\mathbf{x}}(r)$. Each distinct combinations of $N-1$ leaves corresponds to a different $r \in \Omega$. Note that the hidden leaf masks the other evaluations and therefore masks the exact definition of the polynomial. The verifier thus obtains the desired polynomial evaluation and can verify the authenticity of this value using the previously received commitment of the tree.

2.6 MQOM

Let us now turn to the precise definition of the MQOM signature. To do so, we introduce $\mathbb{F}$, the base field over which we will work, and $\mathbb{K}$, an extension of $\mathbb{F}$ of degree μ.

The One-Way Function. The security of the MQOM signature scheme relies on the hardness of the MQ problem, defined in Sect. 2.3. Note also that in each variant of MQOM, we have $m = n$. We define the one-way function F as the evaluation of the system at a point $\mathbf{x} \in \mathbb{F}^n$. Specifically, we set $F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \in \mathbb{F}^m$, where each $f_i : \mathbb{F}^n \to \mathbb{F}$ is defined by $f_i(\mathbf{x}) = \mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x} - y_i$.

In this notation, solving the MQ instance $(\{\mathbf{A}_i\}, \{\mathbf{b}_i\}, \mathbf{y})$ means finding $\mathbf{x} \in \mathbb{F}^n$ such that $F(\mathbf{x}) = 0 \in \mathbb{F}^m$.

We also extend the definition of F to vectors of degree-1 polynomials by setting $f_i : (\mathbb{K}[X]^{\leq 1})^n \to \mathbb{K}[X]^{\leq 2}$

$$f_i(\mathbf{P}(X)) = \mathbf{P}(X)^T \mathbf{A}_i \ \mathbf{P}(X) + (\mathbf{b}_i)^T \mathbf{P}(X) \cdot X - y_i \cdot X^2.$$

Let $\mathbf{P}_{\mathbf{x}}(X)$ be the vector of polynomials defined in Sect. 2.2. Note that the inclusion of the X and X^2 factors in the definition of f_i ensures the property $F(\mathbf{P})(\infty) = F(\mathbf{x})$, which is the zero vector if and only if $\mathbf{x}$ is indeed a solution of the MQ instance $(A_i, b_i, \mathbf{y})$.

The MQOM PIOP. Before we begin, let us introduce the function Φ used in the PIOP. Let $\beta_1, \ldots, \beta_\mu$ be an $\mathbb{F}$-basis of $\mathbb{K}$. We define ϕ as the $\mathbb{F}$-linear field-embedding isomorphism:

$$\phi : (e_1, \ldots, e_\mu) \in \mathbb{F}^\mu \mapsto \sum_{i=1}^{\mu} e_i \beta_i \in \mathbb{K}.$$

This function is extended to $\mathbb{F}^m$, where m is a multiple of μ (which is the case for m in all variants of MQOM), by:

$$\Phi : (e_1, \ldots, e_m) \in \mathbb{F}^m \mapsto (\phi(e_1, \ldots, e_\mu), \ldots, \phi(e_{m-\mu+1}, \ldots, e_m)) \in \mathbb{K}^{\frac{m}{\mu}}.$$

We now present the PIOP used in MQOM. This part closely follows the MQOM specification [7] and is illustrated in Fig. 1, which is also taken from the specification. Security arguments are omitted here.

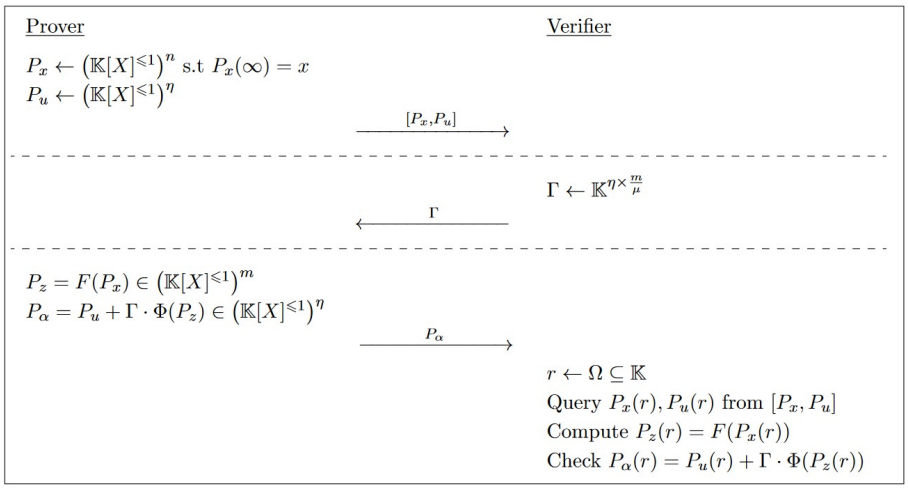

Fig. 1. MQOM PIOP protocol, see [7]

At the beginning of the protocol, the prover selects two vectors of polynomials. The first, $\mathbf{P}_u(X) \in (\mathbb{K}[X]^{\leq 1})^\eta$, is chosen at random, where η is a parameter. The second, $\mathbf{P_x}(X) \in (\mathbb{K}[X]^{\leq 1})^n$, is constructed such that $(\mathbf{P_x}(\infty))_i = P_{x_i}(\infty) = x_i$. The constant term of $\mathbf{P_x}(X)$ is randomly chosen in $\mathbb{K}$. The prover then gives the verifier access to these polynomials through an oracle. Next, the verifier samples a matrix $\mathbf{\Gamma} \in \mathbb{K}^{\eta \times \frac{m}{\mu}}$. The prover computes $\mathbf{P}_z = F(\mathbf{P_x})$ using the polynomial version of the function F. Writing $\mathbf{P}_z = \mathbf{z_0} + \mathbf{z_1}X + \mathbf{z_2}X^2$, we observe that $\mathbf{z_2} = F(\mathbf{P_x})(\infty) = F(\mathbf{x})$ by the definition of F, and therefore $\mathbf{z_2} = 0$ by the definition of $\mathbf{x}$. The prover computes thus $\Phi(\mathbf{P}_z) = \Phi(\mathbf{z_0}) + \Phi(\mathbf{z_1})X$.

Remark. Variables $\mathbf{z_i} \in \mathbb{K}^m$, while Φ is originally defined on $\mathbb{F}^m$, so we naturally extend its definition to this space. This extension does not affect the properties of MQOM since the key part, $\Phi(\mathbf{z_2}) = \Phi(F(\mathbf{x}))$, remains well-defined as $F : \mathbb{F}^n \mapsto \mathbb{F}^m$.

Next, the prover computes $\mathbf{\Gamma} \cdot \Phi(\mathbf{P}_z) = \mathbf{\Gamma} \cdot \Phi(\mathbf{z_0}) + \mathbf{\Gamma} \cdot \Phi(\mathbf{z_1})X$ and sends directly (not as an oracle) to the verifier the degree-1 polynomial $\mathbf{P}_\alpha = \mathbf{P}_u + \mathbf{\Gamma} \cdot \Phi(\mathbf{P}_z)$. Finally, the verifier validates the proof by sampling a random point $r \in \Omega \subset \mathbb{K}$ and querying the oracles for $\mathbf{P_x}$ and $\mathbf{P}_u$ to get $\mathbf{P_x}(r)$ and $\mathbf{P}_u(r)$. The verifier recomputes $\mathbf{P}_\alpha(r) = \mathbf{P}_u(r) + \mathbf{\Gamma} \cdot \Phi(F(\mathbf{P_x}(r)))$ and checks that it is consistent with the polynomial $\mathbf{P}_\alpha$ provided by the prover.

In this protocol, Φ and $\mathbf{\Gamma}$ are used to batch computations, aiming at reducing the size of the polynomials exchanged between the prover and the verifier — and thus the overall size of a MQOM signature. Batching with Φ is always used, but batching with $\mathbf{\Gamma}$ is only used in some variants because it can be more computationally expensive. In variants where this batching is disabled, the generation

and transmission of $\mathbf{\Gamma}$ are omitted (as indicated by dashed lines in Fig. 1), resulting in a 3-round protocol instead of 5. Throughout the rest of the protocol, $\mathbf{\Gamma}$ can either be ignored or replaced by the identity matrix $\mathbf{I}^{\frac{m}{\mu} \times \frac{m}{\mu}}$.

The Variants. The MQOM scheme supports three security levels, aligned with NIST levels I, III, and V. For each level, there are eight variants described in detail in [7]. These differ by the choice of base field ($\mathbb{F}_2$ or $\mathbb{F}_{256}$), whether batching with $\mathbf{\Gamma}$ is used, and the value of N (either 256 or 2048). Recall that N corresponds to the number of leaves in the GGM tree used in the polynomial commitment scheme (see Sect. 2.4), i.e. the number of possible points $|\Omega|$ for which the verifier can query the polynomial oracles. These three sources of variation yield the $2 \times 2 \times 2 = 8$ possible variants. Variants instantiated over $\mathbb{F}_2$ are more expensive in terms of computation but produce smaller signatures. Similarly, using $\mathbf{\Gamma}$ reduces the signature size at the cost of additional computation. When batching is activated, one extra exchange occurs in the ZKP (see Fig. 1), increasing the number of rounds from 3 to 5. Finally, the choice of N provides a third trade-off between size and efficiency: variants with $N = 256$ are "short", whereas those with $N = 2048$ are "fast". Note that the extension field $\mathbb{K}$ used after in the computations is chosen based on the value of N, since it must satisfy $|\mathbb{K}| > N = |\Omega|$. For simplicity, the authors select a single extension field compatible with both base fields: they set $\mathbb{K} = \mathbb{F}_{2^{16}}$ when $N = 2048$, and $\mathbb{K} = \mathbb{F}_{256}$ when $N = 256$.

In this work we focus on the NIST level I variants to demonstrate our attacks. These eight variants are summarized in Table 1, where each row corresponds to the 3-round and 5-round version for a given parameter set. The table also recalls the number of key coefficients n, the degree μ of the extension $\mathbb{K}$ of $\mathbb{F}$, and the size η of the output polynomial $\mathbf{P}_\alpha(X)$. Note that our attacks also apply in theory to the higher security levels, with only the complexity being affected.

Table 1. MQOM variants

Parameter sets	$\lvert\mathbb{F}\rvert$	$m = n$	μ	η
L1-gf2-short-3r/5r	2	160	16	10/8
L1-gf2-fast-3r/5r	2	160	8	20/16
L1-gf256-short-3r/5r	256	48	2	24/8
L1-gf256-fast-3r/5r	256	48	1	48/16

2.7 Adversary models

In this paper, we assume the attacker can modify one or more coefficients of $\mathbf{x}$, $\mathbf{A}_i$, or $\mathbf{b}_i$. To describe these modifications, we use two well-known fault models: the *random* and *stuck-at* models. There are various ways these effects can happen, such as instructions being skipped or changed, or values being altered during loading or CPU manipulation.

- In the random fault model, the targeted value is modified in a random and uncontrolled manner. This model captures all possible fault effects that could be injected, making it the most general and encompassing fault model.
- In the *stuck-at* model, the attacker can force the targeted value to a specific, known value. This includes, in particular, the *stuck-at-0* fault model, where all 1s in the target are flipped to 0, while 0s stay the same.

3 Theory Behind the Faults

We now present four fault attacks on MQOM, grouped into two types of scenarios based on how the fault injection is used to recover $\mathbf{x}$. In the first attack scenario, we exploit a fault injection on $\mathbf{x}$ with a *random* fault model. This attack can be instantiated in two different ways (in Sect. 3.1 and Sect. 3.2), depending on where the fault is injected. In the second scenario, we target $\mathbf{A}_i$ (Sect. 3.3) or $\mathbf{b}_i$ (in Sect. 3.4) with a *stuck-at* model. Note that while these four faults reveal secret information, they yield signatures that fail verification. However, performing verification of signatures generated via MPCitH protocols is computationally costly, as discussed in Sect. 6. This section outlines the theoretical principles behind the attacks.

3.1 Fault on x After $\mathbf{\Delta_x}$ Generation

In this first attack path, we assume that the attacker is able to inject a *random* fault, i.e. a fault which can be unpredictable for the attacker, into one or several coefficients of $\mathbf{x}$. Under this condition, each fault injection leads to a set of several equations, each depending on $\mathbf{x}$, various values known to the attacker, and δ the random fault. To solve for $\mathbf{x}$, the attacker needs to collect n equations and perform an exhaustive search over (possibly several) δ. Observe that each δ (i.e. each fault affecting an individual coefficient of $\mathbf{x}$) belongs to the base field $\mathbb{F}$ as $\mathbf{x} \in \mathbb{F}^n$. Depending on the specific variant, multiple fault injections may be required to gather a sufficient number of equations. More details are provided in Sect. 4.

For this first subsection we assume that the oracle — which provides the verifier with the evaluations of the $P_{x_i}(X)$ — remains unaffected.

From the Fault Injection to a Set of η-Equation Systems. Without loss of generality, we assume that the coordinate x_1 is faulted to a new value $\tilde{x}_1$ such that $\tilde{x}_1 = x_1 + \delta$, where $\delta \in \mathcal{D}$ the set of possible fault values. The prover has to compute $\mathbf{P_z} = \mathbf{z}_0 + \mathbf{z}_1 X = F(\mathbf{P_x})$. Note that the prover does not compute the term $\mathbf{z}_2 X$ as it is expected to be null. According to the specification [7], the coefficients of $\mathbf{z}_0$ and $\mathbf{z}_1$ are computed separately as follows:

$$\begin{aligned}(\mathbf{t}_0)_i &= \mathbf{A}_i \mathbf{x}_0 \\ (\mathbf{t}_1)_i &= \mathbf{A}_i \mathbf{x} + \mathbf{b}_i \\ (z_0)_i &= (\mathbf{t}_0)_i^T \mathbf{x}_0 \\ (z_1)_i &= (\mathbf{t}_0)_i^T \mathbf{x} + (\mathbf{t}_1)_i^T \mathbf{x}_0.\end{aligned}$$

Only $\mathbf{t}_1$ and $\mathbf{z}_1$ are affected by the fault on $\mathbf{x}$. Now, let us examine how the fault influences these computations.

$$(\tilde{\mathbf{t}}_1)_i = \mathbf{A}_i\tilde{\mathbf{x}} + \mathbf{b}_i = \mathbf{A}_i\mathbf{x} + \mathbf{b}_i + \delta \begin{pmatrix} a_{11}^{(i)} \\ \vdots \\ a_{n1}^{(i)} \end{pmatrix} = (\mathbf{t}_1)_i + \delta \begin{pmatrix} a_{11}^{(i)} \\ \vdots \\ a_{n1}^{(i)} \end{pmatrix}$$

$$\begin{aligned}
(\tilde{z}_1)_i &= (\mathbf{t}_0)_i^T\tilde{\mathbf{x}} + (\tilde{\mathbf{t}}_1)_i^T\mathbf{x}_0 \\
&= (\mathbf{t}_0)_i^T\mathbf{x} + (\mathbf{t}_0)_i^T \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix} + (\mathbf{t}_1)_i^T\mathbf{x}_0 + \delta \begin{pmatrix} a_{11}^{(i)} \\ \vdots \\ a_{n,1}^{(i)} \end{pmatrix}^T \mathbf{x}_0 \\
&= (\mathbf{t}_0)_i^T\mathbf{x} + \sum_{j=1}^{n} a_{1j}^{(i)}(x_0)_j\delta + (\mathbf{t}_1)_i^T\mathbf{x}_0 + \sum_{j=1}^{n} a_{j1}^{(i)}(x_0)_j\delta \\
&= (z_1)_i + \sum_{j=1}^{n}(a_{1j}^{(i)} + a_{j1}^{(i)})(x_0)_j\delta.
\end{aligned}$$

Once the prover computed the faulty $\tilde{\mathbf{P}}_\mathbf{z}$, she can compute $\tilde{\mathbf{P}}_\alpha = \mathbf{P}_\mathbf{u} + \mathbf{\Gamma} \cdot \Phi(\tilde{\mathbf{P}}_\mathbf{z})$. The functions Φ and Γ are defined in Section 2.6. The former combines each successive block of μ coordinates of $\mathbf{z}$ into a single variable, resulting in a vector of size $\frac{m}{\mu}$, while the latter produces η linear combinations of these $\frac{m}{\mu}$ variables. Let us look at the impact of the fault injection:

$$\mathbf{\Gamma} \cdot \Phi(\tilde{\mathbf{z}}_1) = \mathbf{\Gamma} \cdot \begin{pmatrix} \sum_{l=1}^{\mu} \beta_l(\tilde{z}_1)_l \\ \vdots \\ \sum_{l=1}^{\mu} \beta_l(\tilde{z}_1)_{l+(\frac{m}{\mu}-1)\mu} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{m/\mu} \gamma_{1,k} \sum_{l=1}^{\mu} \beta_l(\tilde{z}_1)_t \\ \vdots \\ \sum_{k=1}^{m/\mu} \gamma_{\eta,k} \sum_{l=1}^{\mu} \beta_l(\tilde{z}_1)_t \end{pmatrix},$$

with $t = (k-1)\mu + l$.

The prover then sends the faulted polynomial $\tilde{\mathbf{P}}_\alpha$ to the verifier. Recall that the fault is introduced after the oracle construction, so the verifier still receives the correct values of $[\mathbf{P}_\mathbf{x}, \mathbf{P}_\mathbf{u}]$. As a result, the verifier samples a random $r \in \Omega$ and queries $\mathbf{P}_\mathbf{x}(r)$ and $\mathbf{P}_\mathbf{u}(r)$, which are used to derive the expected (unfaulted) value $\mathbf{P}_\alpha(r)$. The verifier can then compute the difference: $\tilde{\mathbf{P}}_\alpha(r) - \mathbf{P}_\alpha(r)$.

Note that in our computations, each component $(\tilde{z}_1)_i$ of the faulted polynomial includes the original term $(z_1)_i$. Due to the linearity of both Φ and $\mathbf{\Gamma}$, this implies that $\mathbf{\Gamma} \cdot \Phi(\tilde{\mathbf{P}}_\mathbf{z}(r))$ contains $\mathbf{\Gamma} \cdot \Phi(\mathbf{P}_\mathbf{z}(r))$ as a subpart. Moreover, the subtraction cancels the term $\mathbf{P}_\mathbf{u}$ out as it appears in both $\mathbf{P}_\alpha$ and $\tilde{\mathbf{P}}_\alpha$. The difference therefore comes solely from the additional terms introduced in $(\tilde{z}_1)_i$. The difference can thus be expressed as:

$$\tilde{\mathbf{P}}_\alpha(r) - \mathbf{P}_\alpha(r) = \mathbf{\Gamma} \cdot \Phi(\tilde{\mathbf{z}}_1)(r) - \mathbf{\Gamma} \cdot \Phi(\mathbf{z}_1)(r)$$

$$= \begin{pmatrix} (\sum_{k=1}^{m/\mu} \gamma_{1,k} (\sum_{l=1}^{\mu} \beta_l [\sum_{j=1}^{n} (a_{j1}^{(t)} + a_{1j}^{(t)}) \delta(x_0)_j])) \times r \\ \vdots \\ (\sum_{k=1}^{m/\mu} \gamma_{\eta,k} (\sum_{l=1}^{\mu} \beta_l [\sum_{j=1}^{n} (a_{j1}^{(t)} + a_{1j}^{(t)}) \delta(x_0)_j])) \times r \end{pmatrix} \quad \text{with } t = (k-1)\mu + l.$$

The resulting equations involve the public values $\mathbf{A}_i$, Φ, $\mathbf{\Gamma}$, and r, while the unknowns in the system are δ and $\mathbf{x}_0$. The goal is to use these equations to recover $\mathbf{x}$ by solving the system. Since by definition $P_{x_j}(r) = (x_0)_j + rx_j$, we can express $\mathbf{x}_0$ in terms of $\mathbf{x}$ as follows:

$$(x_0)_j = P_{x_j}(r) - rx_j, \ \forall j \in [\![1, n]\!]. \tag{1}$$

This gives us a system of η equations involving δ and $\mathbf{x}$. By exhaustively searching all possible values of δ, one can eventually find the right one and obtain a valid system of η equations in $\mathbf{x}$ to recover the secret. Details about the number of faulty outputs required, the exhaustive search and its complexity are given Sect. 4.

Managing the Multi-Coefficient Perturbation. If multiple coefficients of $\mathbf{x}$ are corrupted, one can construct the η equations for each faulty coefficient separately, and then sum them to form the system to be solved.

Attack Surface. We now analyze at which points an attacker can inject a fault during the signature computation. To compute $(\mathbf{t}_1)_i$ and $(z_1)_i$, the prover may need to load the secret key $\mathbf{x}$ into memory after the computation of $\mathbf{\Delta_x}$ and before the computation of $(\mathbf{t}_1)_i$ and $(z_1)_i$. An attacker can therefore try to fault $\mathbf{x}$ at this very moment. Note that the impact of the fault depends on its timing, it can affect either just one PIOP repetition or all the τ repetitions. It is also worth noting that $\mathbf{x}$ should be masked, however it is sufficient to inject a fault into one of the shares of $\mathbf{x}$, since any random fault is exploitable in this attack.

3.2 Fault on x Before $\mathbf{\Delta_x}$ Generation

The attack introduced above can be extended to cases where a $\mathbf{x}$ coefficient is disturbed before generating $\mathbf{\Delta_x}$, thus affecting the oracle provided to the verifier. In this scenario, the prover's side remains unchanged, and only the first-degree term of $\mathbf{P}_\alpha$ is faulty, similarly to the previous section. Indeed, this fault, like the previous one, should affect the computation of the quadratic term that depends on $\mathbf{x}$. However, this term is not recomputed by the prover. As explained in

Sect. 2.6, the quadratic term $\mathbf{z}_2$ satisfies $\mathbf{z}_2 = F(\mathbf{x}) = 0$ when no fault occurs, and the prover therefore directly assumes this property. However on the verifier's side, unlike previously, the oracle provides a faulty evaluation $\tilde{\mathbf{P}}_{\mathbf{x}}(r)$, causing the error to propagate into both the first and second-degree terms of $\tilde{\mathbf{P}}_{\alpha}$. Indeed, the verifier does not distinguish computations by degree, since she only manipulates an evaluation of $\mathbf{P}_{\mathbf{x}}$ and $\mathbf{P}_{\alpha}$, rather than the coefficients as the prover does. The first-degree term is thus corrupted in the same manner on both the prover and verifier sides, such that the only difference lies in the second-degree term, which is not calculated on the prover's side. Let us now examine how the fault injection impacts the computation of $\mathbf{z}_2$:

$$
\begin{aligned}
&(\tilde{z}_2)_i = (\tilde{\mathbf{x}})^T \mathbf{A}_i \tilde{\mathbf{x}} + (\mathbf{b}_i)^T \tilde{\mathbf{x}} + y_i \\
&= \mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{x}^T \mathbf{A}_i \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \mathbf{A}_i \mathbf{x} + \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \mathbf{A}_i \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \mathbf{b}_i^T \mathbf{x} + \mathbf{b}_i^T \begin{pmatrix} \delta \\ 0 \\ \vdots \\ 0 \end{pmatrix} + y_i \\
&= \mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x} + y_i + \sum_{j=1}^{n} a_{j1}^{(i)} x_j \delta + \sum_{j=1}^{n} a_{1j}^{(i)} x_j \delta + a_{11}^{(i)} \delta^2 + b_1^{(i)} \delta \\
&= (z_2)_i + \sum_{j=1}^{n} (a_{j1}^{(i)} + a_{1j}^{(i)}) x_j \delta + a_{11}^{(i)} \delta^2 + b_1^{(i)} \delta \\
&= \sum_{j=1}^{n} (a_{j1}^{(i)} + a_{1j}^{(i)}) x_j \delta + a_{11}^{(i)} \delta^2 + b_1^{(i)} \delta.
\end{aligned}
$$

Let us set $t = (k-1)\mu + l$. Once the verifier has the faulty $\mathbf{P}_{\alpha}$ from the prover and from the oracle, noted $\tilde{\mathbf{P}}_{\alpha,p}$ and $\tilde{\mathbf{P}}_{\alpha,v}$ respectively, she can build the following system:

$$
\begin{aligned}
&\tilde{\mathbf{P}}_{\alpha,p}(r) - \tilde{\mathbf{P}}_{\alpha,v}(r) = -\boldsymbol{\Gamma} \cdot \Phi(\tilde{\mathbf{z}}_2)(r) \\
&\qquad = - \begin{pmatrix} (\sum_{k=1}^{m/\mu} \gamma_{1,k} (\sum_{l=1}^{\mu} \beta_l [\sum_{j=1}^{n} (a_{j1}^{(t)} + a_{1j}^{(t)}) x_j \delta + a_{11}^{(t)} \delta^2 + b_1^{(t)} \delta])) \times r^2 \\ \vdots \\ (\sum_{k=1}^{m/\mu} \gamma_{\eta,k} (\sum_{l=1}^{\mu} \beta_l [\sum_{j=1}^{n} (a_{j1}^{(t)} + a_{1j}^{(t)}) x_j \delta + a_{11}^{(t)} \delta^2 + b_1^{(t)} \delta])) \times r^2 \end{pmatrix}.
\end{aligned}
$$

This gives us a system of η equations involving δ and $\mathbf{x}$. Similarly to the previous section, the attacker can perform an exhaustive search on δ in order to recover $\mathbf{x}$.

Managing the Multi-Coefficient Perturbation. If multiple coefficients are affected, equations must be constructed for each faulty coefficient and then summed, as in the previous case. Moreover, in this situation, additional terms

arising from the multiplication of faults with themselves must also be included. Indeed, when expanding the above equation with several faults, quadratic terms in the faults appear, whose definition depends on the number of affected coefficients.

Attack Surface. The secret key is used twice in the algorithm before computing $\mathbf{\Delta_x}$. The first usage occurs during the decompression and parsing of the key at the very beginning of the algorithm. The second happens during the computation of δ, a parameter of the GGM tree. To obtain the described result, the fault must affect the computation of $\mathbf{x} - \sum_{i=0}^{N-1} \bar{\mathbf{x}}_i = \Delta_x$ (described in Sect. 2.6). In addition, it must also corrupt the computation of $\mathbf{t}_1$ and $\mathbf{z}_1$. Note that if the fault is injected during the computation of δ, depending on the implementation, it may perturb only a copy of the key without affecting the rest of the computations. This is not the case when the fault is injected into $\mathbf{x}$ during decompression and parsing, since in this case the corrupted copy of the key is used by the prover throughout the entire algorithm.

Remark In practice, $\mathbf{\Delta_x}$ is computed in two parts: the first 128 bits come directly from the GGM tree, thanks to a special property known as "correlated" (see the specification [7]). The remaining bits are obtained from the computation $\mathbf{\Delta_x} = \mathbf{x} - \sum_{i=0}^{N-1} \bar{\mathbf{x}}_i$. In both cases, a fault on $\mathbf{x}$ affects $\mathbf{\Delta_x}$ in such a way that $\tilde{\mathbf{P}}_{\alpha,p}(r) - \tilde{\mathbf{P}}_{\alpha,v}(r) = -\mathbf{\Gamma} \cdot \Phi(\tilde{\mathbf{z}}_2)$.

3.3 Fault on $\mathbf{A}_i$

In the second attack scenario, we now assume that the attacker can set a coefficient of $\mathbf{A}_i$ (or $\mathbf{b}_i$ in Sect. 3.4) to a known value. In particular and without loss of generality, we will consider in the following that the attacker can stuck any coefficients at zero. Under these conditions, the attacker can derive an equation involving one or two coefficients of $\mathbf{x}$, along with other public values. In theory, this allows to recover one coefficient of $\mathbf{x}$ for each injected fault. Note that the *stuck-at* fault model is not compatible with masked values, since each share would have to be forced to a predictable value. However, as $\mathbf{A}_i$ and $\mathbf{b}_i$ are public values, they are very likely to be unmasked. This fault model is therefore entirely relevant in this context.

In this section, we focus on the first option: a disturbance affecting a coefficient of $\mathbf{A}_i$. Without loss of generality, let us assume $\tilde{a}_{jk}^{(i)} = 0$. Now, we can identify three ways to exploit this perturbation, depending on how $\tilde{a}_{jk}^{(i)}$ affects the computation of $\mathbf{P}_\alpha$:

(i) It affects only the computation of $(\mathbf{t}_0)_i$
(ii) It affects only the computation of $(\mathbf{t}_1)_i$
(iii) It affects both $(\mathbf{t}_0)_i$ and $(\mathbf{t}_1)_i$

(i) In the first case, we have:

$$(\tilde{z}_0)_i = (z_0)_i - a_{jk}^{(i)}(x_0)_k(x_0)_j$$

$$(\tilde{z}_1)_i = (z_1)_i - a_{jk}^{(i)}(x_0)_k x_j.$$

Since only a single coordinate is affected, it suffices to trace it throughout the protocol. The multiplicative constant of the function Φ depends only on the position of the coordinate within the block of μ successive coordinates, as Φ partitions $\mathbf{z}$ into such blocks. This position is given by $i \bmod (\mu)$, but since our numbering starts from 1 rather than 0, we must adjust it to $(i-1 \bmod \mu)+1$. Note that, the difference $\tilde{\mathbf{P}}_\alpha(r) - \mathbf{P}_\alpha(r)$ will be the same for each output coordinate. Without loss of generality, consider coordinate $l \in [\![1, \eta]\!]$. This coordinate depends on row l of Γ and on the column corresponding to the index of the block of μ coordinates containing z_i. This index is given by $\lfloor \frac{i}{\mu} \rfloor$, that we must also adjust to $\lfloor \frac{i-1}{\mu} \rfloor + 1$. We obtain therefore:

$$\begin{aligned}
(\tilde{\mathbf{P}}_\alpha(r))_l - (\mathbf{P}_\alpha(r))_l &= -(\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)}(x_0)_k x_j \beta_{(i-1 \bmod \mu)+1}) \times r - \\
&\quad (\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)}(x_0)_k(x_0)_j \beta_{(i-1 \bmod \mu)+1}) \\
&= -\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)} \beta_{(i-1 \bmod \mu)+1}((x_0)_k(x_j) \times r + (x_0)_k(x_0)_j) \\
&= -\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)} \beta_{(i-1 \bmod \mu)+1}(x_0)_k P_{x_j}(r) \quad \text{(See Eq. (1))} \\
&= -\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)} \beta_{(i-1 \bmod \mu)+1}(P_{x_k}(r) - r x_k) P_{x_j}(r).
\end{aligned}$$

We thus derive an equation where only a coefficient of $\mathbf{x}$ is unknown. Once again, each fault allows the recovery of one coefficient of $\mathbf{x}$.

(ii) When the fault injection affects only the computation of $(\mathbf{t}_1)_i$, we have:

$$(\tilde{z}_1)_i = (z_1)_i - a_{jk}^{(i)} x_k (x_0)_j.$$

By continuing the protocol, we obtain:

$$\begin{aligned}
(\tilde{\mathbf{P}}_\alpha(r))_l - (\mathbf{P}_\alpha(r))_l &= -(\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)} x_k (x_0)_j \beta_{(i-1 \bmod \mu)+1}) \times r \\
&= -(\gamma_{l,\lfloor \frac{i-1}{\mu} \rfloor+1} a_{jk}^{(i)} x_k (P_{x_j}(r) - r x_j) \beta_{(i-1 \bmod \mu)+1}) \times r.
\end{aligned}$$

The attacker obtains an equation that only depends on two coefficients of $\mathbf{x}$. With the combination of several equations, she can recover $\mathbf{x}$ entirely, see Sect. 4.

(iii) Finally, in the case where the fault affects both the computation of $(\mathbf{t}_0)_i$ and $(\mathbf{t}_1)_i$, the result of the difference is simply the sum of the outcomes from the two previous cases. As a result, the attacker again obtains an equation that depends only on the two variables x_k et x_j when $a_{jk}^{(i)}$ is disturbed.

Managing the Multi-Coefficient Perturbation. If multiple coefficients of $a_{jk}^{(i)}$ are corrupted, affecting either $(\mathbf{t}_1)_i$ alone or both $(\mathbf{t}_0)_i$ and $(\mathbf{t}_1)_i$, the attacker will obtain several quadratic equations involving multiple coefficients of $\mathbf{x}$. In such cases, the resulting system can be challenging to solve.

In contrast, if multiple coefficients of $a_{jk}^{(i)}$ are corrupted affecting $(\mathbf{t}_0)_i$ alone, the attacker obtains several linear equations. To ensure these are linearly independent, the attacker must target different coefficients.

Attack Surface. To impact both $(\mathbf{t}_0)_i$ and $(\mathbf{t}_1)_i$, the attacker can target the generation of any $(\mathbf{A}_i)_{i\in[\![1,m]\!]}$ within the `ExpandEquations` function (see [7]). Let us take a closer look at how a $\mathbf{A}_i$ is generated. It begins with a public seed that is expanded into m sub-seeds s_i using a SHAKE-128 computation. Each s_i is then transformed by computing $\mathtt{AES}_K(s_i)\oplus\psi(s_i)$, where K is a public salt and ψ is a public function. The resulting values are parsed to construct the coefficients of $\mathbf{A}_i$. This gives the attacker several opportunities to inject a fault: by altering the output of the AES, the output of ψ, or by bypassing the final XOR.

Alternatively, the attacker can target the matrix-vector product $\mathbf{A}_i\mathbf{x}_0$ (resp. $\mathbf{A}_i\mathbf{x}$) to affect only the component $(\mathbf{t}_0)_i$ (resp. $(\mathbf{t}_1)_i$). Since $\mathbf{x}_0$ (resp. $\mathbf{x}$) is masked, the multiplications by the constants $a_{jk}^{(i)}$ are performed share by share. However, the matrices $\mathbf{A}_i$ are not masked, as they are public. As a result, the attacker can target the values of the coefficients $a_{jk}^{(i)}$ during their loading or manipulation, either directly or through pointer manipulation. All of these approaches can induce a stuck-at fault.

It is worth noting that the execution of MQOM requires a large number of multiplications over a field of characteristic 2, particularly during the multiplication by the matrix $\mathbf{A}_i$. These multiplications are defined over the spaces $\mathbb{F}\times\mathbb{F}$ when computing $(\mathbf{t}_1)_i$, and $\mathbb{F}\times\mathbb{K}$ when computing $(\mathbf{t}_0)_i$. As highlighted in [18], there are two main methods to implement such multiplications. The first is to compute them directly using logical operations. The second relies on the use of precomputed lookup tables, particularly effective when the base field is $\mathbb{F}_{256}$. In both cases, the implementation requires manipulating registers that contain only a small number of coefficients. For instance, in logic-based multiplications involving elements from $\mathbb{K}$, the number of coefficients that can be packed into a register is limited to roughly $(\textit{register size})/\log_2(|\mathbb{K}|)$. In the case of table-based multiplication, the implementation directly handles the coefficients one by one. In both scenarios, this provides the attacker with a favorable attack surface, allowing faults to be injected into only a small number of coefficients.

3.4 Fault on $\mathbf{b}_i$

The second option in the second attack scenario, introduced in the previous section, is to inject a fault into a coefficient of $\mathbf{b}_i$. Without loss of generality, let's assume $\tilde{b}_j^{(i)} = 0$. This will cause an error in the computation of $((\mathbf{t}_1)_i)_j$, which will then propagate to the calculation of $(z_1)_i$. As a result, we get:

$$(\tilde{z}_1)_i = (z_1)_i - b_j^{(i)}(x_0)_j.$$

By continuing the protocol, we obtain:

$$\begin{aligned}(\tilde{\mathbf{P}}_\alpha(r))_l - (\mathbf{P}_\alpha(r))_l &= -(\gamma_{l,\lfloor\frac{i-1}{\mu}\rfloor+1} b_j^{(i)} (x_0)_j \beta_{(i-1 \bmod \mu)+1}) \times r \\ &= -(\gamma_{l,\lfloor\frac{i-1}{\mu}\rfloor+1} b_j^{(i)} (P_{x_j}(r) - r x_j) \beta_{(i-1 \bmod \mu)+1}) \times r.\end{aligned}$$

As everything is public but x_j, each fault allows us to recover one coefficient of $\mathbf{x}$. Note that the $\eta-1$ other coefficients of $\tilde{\mathbf{P}}_\alpha(r)$ do not bring more information on $\mathbf{x}$. The attacker thus needs to repeat the operation n times to fully recover $\mathbf{x}$.

Managing the Multi-Coefficient Perturbation. If multiple coefficients of $b_j^{(i)}$ are corrupted, the attacker obtains several linear equations. To ensure these are linearly independent, the attacker must target different coefficients.

Attack Surface. The attacker can target the generation of any $\mathbf{b}_i$, which follows a similar process to that of $\mathbf{A}_i$. Moreover, just as with the coefficients $a_{jk}^{(i)}$, the attacker can target the loading or manipulation of $\mathbf{b}_i$ during the addition $(\mathbf{t}_1)_i = \mathbf{A}_i\mathbf{x}+\mathbf{b}_i$ itself. Indeed, since $\mathbf{b}_i$ is public and thus not masked, it remains a viable target for inducing faults.

4 Complexity

Fault on x. If faults affect both $\mathbf{x}$ and $\mathbf{\Delta_x}$, or only $\mathbf{x}$, the attacker must obtain a full-rank system of n equations in order to recover $\mathbf{x}$. When $\mathbf{x}$ is consistently corrupted across all τ repetitions, the attacker collects $\tau \times \eta$ equations. However, due to the structure of the transformation Φ, at most $\frac{n}{\mu}$ of these equations are linearly independent. Indeed, Φ groups the unknowns into blocks of size μ using a fixed basis (see 2.6). As a result, under a given fault, the coefficients of $\mathbf{P}_z$ will be identical within each block across the τ repetitions, and consequently, the batches of μ coordinates produced by Φ will also be identical. Then, a new matrix Γ is sampled and multiplied with each group of μ coordinates. Therefore, the coordinates within the same group cannot be linearly separated, yet thanks to Γ, we still obtain as many degrees of freedom as there are groups created by Φ, i.e., $\frac{n}{\mu}$. In contrast, if $\mathbf{x}$ is faulted in only a single repetition among the τ, the attacker obtains at most $\eta \leq \frac{n}{\mu}$ equations.

Note, that the coefficients of these obtained equations lie in the extension field $\mathbb{K}$, while the unknowns, namely the secret key components and the fault δ, are in the base field $\mathbb{F}$. Recall that the extension $\mathbb{K}$ is of degree μ. As a result, the attacker can reinterpret a system of t equations over $\mathbb{K}$ as a system of $\mu \cdot t$ equations over the base field $\mathbb{F}$, by identifying each equation degree-wise. For instance, in the variant gf2-fast, the coefficients $a_{i,j}$ of the system's equations belong to $\mathbb{K} = {}^{\mathbb{F}_2[X]}\!/\!_{(X^8+X^4+X^3+X+1)}$, while the unknowns x_i and δ are

elements of $\mathbb{F}_2$. Identifying each multiplication in $\mathbb{K}$, $(\sum_{k=0}^{7}(a_{i,j})_k X^k) \times x_i\delta$ as eight multiplications $\sum_{k=0}^{7}((a_{i,j})_k \times x_i\delta)X^k$ in $\mathbb{F}_2$ allows the attacker to identify the equations according to their degree and effectively multiply the number of available equations by a factor of $\mu = 8$.

Moreover, to eventually obtain a full-rank system of n equations over $\mathbb{F}$, the attacker may need to combine multiple subsystems arising from different faults. It is essential that each fault targets a different coefficient of $\mathbf{x}$ in order to obtain a linearly independent system.

We experimentally determined the rank of the system obtained when injecting faults, each targeting a single coefficient of $\mathbf{x}$, assuming that the attacker recovers $\frac{m}{\mu}$ equations per fault. We consider this setting to be generalizable to other configurations. Indeed, note that the more coefficients are perturbed, the more the equation coefficients tend to follow a uniform distribution, which in turn increases the likelihood of obtaining a high-rank system. Moreover, if the attacker has access to fewer equations, typically $\eta \leq \frac{m}{\mu}$, the probability of obtaining a linearly dependent system decreases. For each variant, we measured the rank of the system of $\mu \times \frac{m}{\mu} = m = n$ equations and n unknowns obtained by identifying the coefficients in the extension field representation. It is important to note that, since our tests consider faults that affect only a single coefficient of $\mathbf{x}$, the resulting system, by construction, involves only $n-1$ coefficients, i.e., all except the one targeted by the fault. As a result, the maximum achievable rank for such a system is $n-1$. Table 2 summarizes the ranks observed for the systems. The variants corresponding to 3-round and 5-round instances (i.e., with or without the Γ transformation) are grouped together, as their behavior was found to be similar. Likewise, no significant difference was observed between systems derived from faults affecting both $\mathbf{x}$ and $\mathbf{\Delta_x}$ and those affecting only $\mathbf{x}$.

Table 2. Rank of the system for fault on one coefficient, with $\frac{m}{\mu}$ equations recovered.

Variant	μ	1 Fault	2 Faults
L1-gf2-short-3r/5r	16	90	160
L1-gf2-fast-3r/5r	8	158	160
L1-gf256-short-3r/5r	2	47	48
L1-gf256-fast-3r/5r	1	47	48

Once the attacker obtains a full-rank system of n equations, recovering $\mathbf{x}$ requires testing all possible values of δ across the different subsystems. Let $\mathcal{D}$ denote the set of possible fault values, where each $\delta \in \mathcal{D}$ affects $|\delta|$ coefficients. This defines a fault space of size $|\mathcal{D}| = |\mathbb{F}|^{|\delta|}$, leading to an overall search space of size $|\mathbb{F}|^{t\cdot|\delta|}$ when t faults are combined. Note that if the system is not full-rank, the attacker must enlarge the search space to account for the missing coefficients of $\mathbf{x}$. This table is constructed using the ranks observed in Table 2,

under the assumption that systems of rank $n-1$ become full-rank when multiple coefficients of $\mathbf{x}$ are simultaneously affected by a single fault, since in that case, all coefficients appear in the equations.

Table 3. Search space according to $|\delta|$

Variant	t	$\|\delta\| = 1$ (1 bits)	$\|\delta\| = 16$ (16 bits)	$\|\delta\| = 32$ (32 bits)
L1-gf2-short-3r/5r	2	2^2	2^{32}	2^{64}
L1-gf2-fast-3r/5r	1	$2^1 \times 2^2$	2^{16}	2^{32}

Variant	t	$\|\delta\| = 1$ (8 bits)	$\|\delta\| = 2$ (16 bits)	$\|\delta\| = 4$ (32 bits)
L1-gf256-short/fast-3r/5r	1	$2^8 \times 2^8$	2^{16}	2^{32}

For each possible value of δ, the attacker can solve the linear system corresponding to obtain a candidate $\mathbf{x}_{hyp}$. To test this hypothesis, one can recompute coordinates of the public key $\mathbf{y}$ using the m equations: $\mathbf{x}_{hyp}^T \mathbf{A}_i \mathbf{x}_{hyp} + \mathbf{b}_i^T \mathbf{x}_{hyp} \stackrel{?}{=} \mathbf{y}_i$. An incorrect guess will be rejected with high probability, specifically, with probability $1 - \left(\frac{1}{|\mathbb{F}|}\right)^e$, where e is the number of equations used for the check, and $1 \leq e \leq m$. Therefore, on average, the needed number of verifications to recover $\mathbf{x}$ is about $|\mathbb{F}|^{\mu|\delta|+1}$ when $|\mathbb{F}| = 2$, and about $|\mathbb{F}|^{\mu|\delta|}$ when $|\mathbb{F}| = 256$.

As shown in Table 3, all attacks remain practical, even as the number of affected coefficients increases. It is worth noting that the gf2-short-3r/5r variants, for which the complexity is the highest, are also the least suited for embedded systems, as they are considerably more demanding in terms of computation.

Fault on $\mathbf{A}_i$ and $\mathbf{b}_i$. When $a_{jk}^{(i)}$ or $b_j^{(i)}$ is corrupted, each fault reduces the key's entropy by $\log |\mathbb{F}|$. The attacker could then recover $\mathbf{x}$ in two steps: first, by gathering enough equations to significantly shrink the search space; and second, by performing an exhaustive search to fully recover $\mathbf{x}$.

Concretely, in the case of an attack on one coefficient that yields an equation with a single unknown (i.e., a fault on $\mathbf{b}_i$ or a type-(i) fault on $\mathbf{A}_i$), an attacker capable of performing an exhaustive search over 2^{32} bits can recover the secret key with 44 faults on the $\mathbb{F}_{256}$ variants and with 128 faults on the $\mathbb{F}_2$ variants. For other attacks on $\mathbf{A}_i$, an attacker can achieve the same complexity by strategically targeting the affected coefficients — for example, by targeting coefficients $a_{i,j}$ with a fixed index i for all faults. Finally, in the case of faults that perturb several coefficients, the complexity remains the same provided the attacker can now solve 2^{32} linear systems. These systems are lighter than those arising from faults on $\mathbf{x}$ alone or on both $\mathbf{x}$ and $\boldsymbol{\Delta}_{\mathbf{x}}$, we therefore refer to the practical timings presented in Sect. 5 for these attacks.

5 Validation Using Simulation

For each of our attacks, success fully depends on the fault injection success rate. Once the correct faults are injected, the probability of recovering the key is theoretically 1. However, the fault injection success rate depends on the specific implementation targeted and the component running the code. To the best of our knowledge, there is no public implementation of MQOM for embedded systems. Although a reference implementation of the algorithm exists, an embedded version would likely differ significantly for at least two reasons: first, its high complexity would require new optimizations and component-specific adjustments; second, it would need protections against side-channel attacks, such as masking schemes. Therefore, we believe that practical experiments would not be meaningful on the reference implementation for evaluating the feasibility of our attacks in general. Note that we relied on the work of [18] to identify the elements protected by a masking scheme and thus to define a realistic attacker model in this context. However, that work does not provide a masked implementation on which we could perform experiments.

On our side, we use the attacker model based on the specification provided in [7], as well as the masking scheme described in [18]. We then confirmed the effectiveness of our attack through the use of simulated faults. More specifically, we manually injected all the previously described faults on each variant and then exploited the resulting faulty signatures to recover the secret key. We successfully perform the attack on $\mathbf{A}_i$ and $\mathbf{b}_i$. For the fault attacks targeting $\mathbf{x}$, we also successfully confirmed that $\mathbf{x}$ is indeed a solution to the system of equations described in Section 3, assuming the guess for δ is correct. We obtained a full-rank system under the conditions outlined in Sect.4. Finally, we performed an exhaustive search by iteratively applying Gaussian elimination, focusing on the variant MQOM2-L1-gf256-fast-3r, which appears to be the most suitable for embedded systems. Note that the short variant over $\mathbb{F}_{256}$ would lead to a similar system. In contrast, the variants defined over $\mathbb{F}_2$ are theoretically more computationally demanding, as they yield systems of size 160×160, compared to 48×48 in the previous cases. However, this complexity gap can be mitigated by observing that, in $\mathbb{F}_2$, roughly half of the equations are already reduced for any given Gaussian pivot. Furthermore, while solving a system over $\mathbb{F}_{256}$ typically requires memory accesses to perform multiplications via lookup tables, solving over $\mathbb{F}_2$ can be efficiently implemented using basic bitwise operations, allowing multiple coefficients to be packed into a single register and processed in parallel.

The experiments on MQOM2-L1-gf256-fast-3r were carried out on a laptop equipped with an Intel Core i5-1145G7 processor (2.60 GHz, 4 cores / 8 threads). The algorithm was implemented in C and executed under the Windows Subsystem for Linux (WSL). It is worth noticing that the implementation did not exploit parallelism; hence, only sequential performance was evaluated. The average solving time of 2^{24} systems is approximately 602,78 seconds CPU. By extrapolating from our results to an exhaustive search over a space of size 2^{32}, an attacker would, on average, solve 2^{31} systems in less than 22 hours.

These results demonstrate the practical feasibility of such attacks on a standard desktop machine.

6 Countermeasures

An effective countermeasure against all these attacks would be to perform a verification of the signature. However, signature verification in MPCitH-based schemes is known to be particularly costly. Therefore, alternative countermeasures may be preferred.

To prevent attacks on $\mathbf{x}$ (introduced in Sect. 3.1), the developer could verify its integrity at the end of the signature computation, e.g. using a checksum.

To mitigate attacks on $\mathbf{b}_i$ or $\mathbf{A}_i$ (see Sect. 3.3 and Sect. 3.4), one can randomize the processing order of their coefficients through shuffling during the generation of these parameters and during the evaluation of the MQ system. This results in several possible equations per fault injection, significantly increasing the complexity of any attempted exploitation. Another possibility is to mask the public values $\mathbf{b}_i$ and $\mathbf{A}_i$. Indeed, masking inherently randomizes the underlying data, preventing the attacker from fixing it to a constant without also controlling the mask, which is assumed to be unknown. As attacks introduced on Sect. 3.3 and Sect. 3.4 assume a *stuck-at* random fault model, masking the public values counteracts these attacks.

7 Conclusion

MPCitH-based signature schemes currently appear to be favored by standardisation institutes in the ongoing post-quantum selection processes. While these schemes offer strong theoretical security guarantees, they still lack analysis from a side-channel and fault attack perspective.

In this work, we have shown that the component corresponding to the evaluation of the one-way function within the PIOP part of the MPCitH framework is vulnerable to differential fault attacks. In the case of MQOM, this corresponds to the evaluation of the Multivariate Quadratic system. The four proposed attacks are realistic, applicable across a wide attack surface, and effective against all variants of MQOM. It is worth noting that the fastest MQOM variants, which seems to be the most suitable for resource-constrained embedded systems, also turns out to be the most vulnerable: 1 fault injection is sufficient to fully recover $\mathbf{x}$ in less than one day. Finally, we discussed the security impact of different countermeasures that could be deployed to mitigate these attacks.

In a broader context, the results presented in this paper provide a framework for similar attacks targeting the PIOP component of other MPCitH-based signature schemes. Indeed, comparable differential fault attacks may be applied to other schemes by analyzing the structure and characteristics of their underlying hard problems.

Acknowledgments. We thank the reviewers for their insightful comments.

References

1. Korean post-quantum cryptography competition (2023-2025), https://www.kpqc.or.kr/competition_02.html
2. Adj, G., Aragon, N., Barbero, S., Bardet, M., Bellini, E., Bidoux, L., Chi-Domínguez, J.J., Dyseryn, V., Esser, A., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M., Rivera-Zamarripa, L., Sanna, C., Tillich, J.P., Verbel, J., Zweydinger, F.: Mirath signature scheme, algorithm specifications and supporting documentation. Submission to the NIST's post-quantum cryptography standardization process (2025)
3. Aguilar Melchor, C., Feneuil, T., Gama, N., Gueron, S., Howe, J., Joseph, D., Joux, A., Persichetti, E., Randrianarisoa, T., Rivain, M., et al.: Sdith: Syndrome decoding in-the-head. Submission to the NIST Post-Quantum Standardization project (2023)
4. Aranha, D.F., Berndt, S., Eisenbarth, T., Seker, O., Takahashi, A., Wilke, L., Zaverucha, G.: Side-channel protections for picnic signatures. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 239–282 (2021)
5. Baum, C., Beullens, W., Braun, L., de Saint Guilhem, C.D., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: Faest v2: Algorithm specifications version 2.0. Submission to the NIST's post-quantum cryptography standardization process (2025), https://faest.info/faest-spec-v2.0.pdf
6. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14085, pp. 581–615. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4"19
7. Benadjila, R., Bouillaguet, C., Feneuil, T., Rivain, M.: Mqom: Mq on my mind algorithm specifications and supporting documentation (version 2.0). Submission to the NIST's post-quantum cryptography standardization process (2025), https://mqom.org/docs/mqom-v2.0.pdf
8. Benadjila, R., Feneuil, T., Rivain, M.: MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In: 9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024. pp. 468–485. IEEE (2024). https://doi.org/10.1109/EUROSP60621.2024.00032
9. Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., et al.: Sphincs (2017)
10. Bettaieb, S., Bidoux, L., Dyseryn, V., Esser, A., Gaborit, P., Kulkarni, M., Palumbi, M.: PERK: compact signature scheme based on a new variant of the permuted kernel problem. Des. Codes Cryptogr. **92**(8), 2131–2157 (2024)
11. Bidoux, L., Chi-Domínguez, J., Feneuil, T., Gaborit, P., Joux, A., Rivain, M., Vinçotte, A.: RYDE: A digital signature scheme based on rank-syndrome-decoding problem with mpcith paradigm. CoRR **abs/2307.08726** (2023)
12. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy, W. (ed.) Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding. Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer (1997)

13. Chase, M., Derler, D., Goldfeder, S., Katz, J., Kolesnikov, V., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Wang, X., et al.: The picnic signature scheme. Submission to NIST Post-Quantum Cryptography project (2020)
14. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018)
15. Fell, H., Diffie, W.: Analysis of a public key approach based on polynomial substitution. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 340–349. Springer (1985)
16. Feneuil, T., Rivain, M.: Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. IACR Cryptol. ePrint Arch. p. 1573 (2023)
17. Feneuil, T., Rivain, M.: Threshold linear secret sharing to the rescue of mpc-in-the-head. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14438, pp. 441–473. Springer (2023)
18. Feneuil, T., Rivain, M., Warmé-Janville, A.: Masking-friendly post-quantum signatures in the threshold-computation-in-the-head framework. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(4), 667–710 (Sep 2025). https://doi.org/10.46586/tches.v2025.i4.667-710, https://tches.iacr.org/index.php/TCHES/article/view/12425
19. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
20. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Submission to the NIST's post-quantum cryptography standardization process **36**(5), 1–75 (2018)
21. Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 29. wh freeman New York (2002)
22. Gellersen, T., Seker, O., Eisenbarth, T.: Differential power analysis of the picnic signature scheme. In: Cheon, J.H., Tillich, J. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12841, pp. 177–194. Springer (2021)
23. Godard, J., Aragon, N., Gaborit, P., Loiseau, A., Maillard, J.: Single trace side-channel attack on the mpc-in-the-head framework. IACR Cryptol. ePrint Arch. p. 1882 (2024)
24. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007. pp. 21–30. ACM (2007)
25. Jendral, S., Dubrova, E.: Side-channel and fault injection attacks on voleith signature schemes: A case study of masked FAEST. IACR Cryptol. ePrint Arch. p. 378 (2025)
26. Lee, J., Cho, A.J., Kim, S., Kwon, J., Lee, B., Lee, J., Lee, S., Moon, D., Son, M., Yoon, H., et al.: The aimer signature scheme

27. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In: Advances in Cryptology—EUROCRYPT'88: Workshop on the Theory and Application of Cryptographic Techniques Davos, Switzerland, May 25–27, 1988 Proceedings 7. pp. 419–453. Springer (1988)
28. Patarin, J.: Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In: International conference on the theory and applications of cryptographic techniques. pp. 33–48. Springer (1996)
29. of Standards, N.I., Technology: Module-lattice-based digital signature standard (fips 204). Federal Information Processing Standards Publication FIPS 204, U.S. Department of Commerce, NIST Information Technology Laboratory (August 2024). https://doi.org/10.6028/NIST.FIPS.204, https://doi.org/10.6028/NIST.FIPS.204
30. of Standards, N.I., Technology: Stateless hash-based digital signature standard (fips 205). Federal Information Processing Standards Publication FIPS 205, U.S. Department of Commerce, NIST Information Technology Laboratory (August 2024). https://doi.org/10.6028/NIST.FIPS.205, https://doi.org/10.6028/NIST.FIPS.205
31. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 2986–3001. ACM (2021)