A Fault Analysis on SNOVA

Gustavo Banegas[1] and Ricardo Villanueva-Polanco[2]

[1]Inria and Laboratoire d'Informatique de l'Ecole polytechnique, Institut Polytechnique de Paris, Palaiseau, France
gustavo@cryptme.in
[2]Technology Innovation Institute, UAE
ricardo.polanco@tii.ae

Abstract. SNOVA, a post-quantum signature scheme with compact key sizes, is a second-round NIST candidate. This paper conducts a fault analysis of SNOVA, targeting permanent and transient faults during signature generation. We propose fault injection strategies that exploit SNOVA's structure, enabling key recovery with as few as 22 to 68 faulty signatures, depending on security levels. A novel fault-assisted reconciliation attack is introduced that effectively extracts the secret key space by solving a quadratic polynomial system. Simulations reveal that transient or permanent faults in signature generation can severely compromise security. We also suggest a lightweight countermeasure to mitigate fault attacks with minimal overhead. Our findings emphasize the need for fault-resistant mechanisms in post-quantum schemes like SNOVA.

1 Introduction

The National Institute of Standards and Technology (NIST) initiated an additional call for post-quantum digital signature proposals to introduce variability in the mathematical foundations of digital signatures. In response, NIST received 40 submissions based on diverse mathematical problems. Among these, 10 submissions were based on multivariate polynomial equations over finite fields, a branch of post-quantum cryptography known as MQ-based cryptography.

MQ-based cryptography relies on the difficulty of solving systems of multivariate quadratic equations over finite fields. The fundamental problem can be defined as follows: given a system of m quadratic equations in n variables over a finite field $\mathbb{F}_q$, find a solution for $\mathbf{x}$ where

$$\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n \text{ such that } \begin{cases} Q_1(\mathbf{x}) = 0 \\ \vdots \\ Q_m(\mathbf{x}) = 0 \end{cases}$$

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. Date of this document: 2025-02-13.

where each Q_i is a quadratic polynomial of the form: $Q_i(x_1, x_2, \ldots, x_n) = \sum_{1\leq j\leq k\leq n} a_{ijk}x_jx_k + \sum_{1\leq j\leq n} b_{ij}x_j + c_i$, with coefficients $a_{ijk}, b_{ij}, c_i \in \mathbb{F}_q$.

The security of MQ-based cryptography is based on the computational hardness of the Multivariate Quadratic problem. Specifically, for large values of n, solving a random system of such equations is known to be NP-hard, making it computationally infeasible for an attacker to solve within a reasonable time frame, even with powerful computational resources.

In addition to the inherent mathematical complexity, implementing robust protections is essential for securing MQ-based cryptographic schemes against both side-channel and fault attacks. Passive side-channel attacks exploit various forms of leakage—such as timing variations, power consumption, or electromagnetic emissions—to gain insights into the cryptographic process. These attacks take advantage of unintended information leaks that arise during the physical implementation of a cryptographic algorithm, rather than exploiting weaknesses in the algorithm itself.

Fault attacks involve deliberately introducing errors during cryptographic execution, such as memory corruption or bit flips, to extract sensitive information by analyzing erroneous outputs. These attacks exploit vulnerabilities in the physical implementation of cryptographic systems, requiring specific countermeasures to ensure resilience.

Techniques such as redundancy checks, error detection, and fault-tolerant designs are commonly employed to mitigate fault attacks. These measures help detect and correct errors induced by fault injection, ensuring the integrity of cryptographic operations.

On the other hand, passive side-channel attacks are countered using methods such as constant-time algorithms, masking techniques, and noise generation. These safeguards prevent attackers from exploiting unintended information leaks, such as power consumption or electromagnetic radiation. Together, these countermeasures enhance the robustness of cryptographic implementations, ensuring that the theoretical security of MQ-based schemes translates into practical resilience against active and passive side-channel attacks in the real world.

1.1 MQ signature schemes

The C^* scheme, introduced in 1988 [19], was one of the first attempts to create a digital signature scheme from the MQ problem. However, it was broken by Patarin in 1995 [22]. Since then, significant progress has been made in multivariate polynomial-based signature schemes, with the Unbalanced Oil-Vinegar (UOV) scheme emerging as a notable and secure example [23].

We can briefly define UOV as: let v be the number of vinegar variables: $v_1, v_2, \ldots, v_v$, and o be the number of oil variables: $o_1, o_2, \ldots, o_o$.

The private key consists of a secret linear map $\mathcal{T} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$ and a map $\mathcal{F} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^o$, known as the central map, that consists of o UOV quadratic polynomials in $n = v + o$ variables.

The public map is defined as $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$, and therefore consists of o quadratic polynomials P_i in n variables over a finite field $\mathbb{F}_q$.

In the UOV scheme signature process, the message `M` is first processed to generate a digest using a cryptographic hash function. This digest is then combined with a random salt, the combined value is used to derive a target vector $Y = (y_1, \ldots, y_o)$. Next, random values are selected for the vinegar variables $v_1, \ldots, v_v$, where v is the number of vinegar variables. These vinegar variables are substituted into the quadratic polynomials of the central map $\mathcal{F}$, which is part of the private key. Substituting the vinegar variables reduces the system to a set of linear equations in the oil variables $o_1, \ldots, o_o$.

The resulting linear system is then solved to determine the values of the oil variables. This step typically involves Gaussian elimination or other linear algebra techniques. Once the oil variables are computed, the signature is constructed by combining the values of both the vinegar and oil variables into a single vector $X = (v_1, \ldots, v_v, o_1, \ldots, o_o)$. This vector X is then transformed using the private linear transformation $\mathcal{T}$ to produce the final signature $S = \mathcal{T}^{-1}(X)$. The signature, along with the salt, is output as the signed message.

To verify the signature, the verifier uses the public key, which consists of the public map $\mathcal{P}$. The verifier substitutes the signature S into the public polynomials and checks if the result matches the target vector Y. If the values match, the signature is valid; otherwise, it is rejected.

Recently, several UOV-like schemes have been proposed, such as MAYO [5,7] and SNOVA [18, 28]. They offer benefits such as *small* signatures, fast verification, and reasonable public key sizes.

1.2 Side-channel attacks

Fault attacks are classified as **active attacks** because they actively manipulate the data or execution environment of a cryptographic system. Techniques such as electromagnetic pulses, lasers, clock glitches, voltage glitches, and DRAM row hammering vary in precision and complexity [1, 15]. For example, laser-based methods are highly precise but costly, while DRAM row hammering requires extensive profiling. These attacks often involve repeated attempts to induce specific faults, enabling the extraction of cryptographic information.

In contrast, **passive side-channel attacks** do not interfere with the system but instead observe unintended information leaks, such as power consumption, electromagnetic radiation, or timing variations. A recent study by [2] provides a comprehensive overview of passive and active attacks on multivariate quadratic MQ-based cryptographic systems.

Table 1 compares previous fault injection attacks on multivariate signature schemes, highlighting key features such as the number of signatures and faults required, the evaluation method, and any assumptions made.

Recently, [14] presented an attack on a MAYO implementation that successfully recovered the private key. This attack targeted a single execution of MAYO. However, the fault was not in MAYO itself, but rather in the C implementation of Keccak, which is responsible for generating the Vinegar and Oil variables. The paper exploits a vulnerability in the pseudorandom function, using a fault in this component to reveal information leading to private key recovery.

Table 1: Comparison of Previous Fault Attacks on Multivariate Signature Schemes.

Algorithm	**#Signatures**	**#Faults**	**Evaluation**	**Assumptions**
Multiple [12]	Multiple	Multiple	Theoretical	None
UOV/Rainbow [16]	Multiple	Multiple	Theoretical	None
UOV [26]	44–103	Multiple	Theoretical	None
LUOV [20]	Multiple	Multiple	Practical	Key in $\mathbb{F}_2$
Rainbow [3]	Multiple	1	Simulation	Exact memory reuse
UOV [11]	Multiple	2–40	Simulation	Enumeration 2^{41}–2^{89}
MAYO [25]	2	1	Theoretical	None
MAYO [4]	1	1	Practical	Zero-initialization
MAYO [14]	1	1	Practical, Simulation	None
SNOVA permanent fault strategy	22–68	1	Theoretical, Simulation	None
SNOVA Fault-assisted reconciliation attack	1	Multiple	Practical, Simulation	None

In this work, we explore a similar attack. However, instead of targeting Keccak, we focus directly on the vinegar variables by inducing faults, that is, fixing specific bit values, which allows us to recover the private key. Despite this similarity, our approach differs in the algorithm used for key recovery. Moreover, we introduce an SNOVA fault-assisted reconciliation attack that requires only a single signature. This approach is different from previous work.

1.3 Our contributions

In this paper, we investigate fault injection attacks on SNOVA, showing that inducing permanent or transient faults during signature generation can reveal partial private key information. We analyze scenarios where an attacker recovers rows of the private matrix T by fixing $\mathbb{F}_{16}$ elements or bits in vinegar variables, demonstrating that enough faulty signatures can compromise the private key. Additionally, we explore a reconciliation attack where transient faults in vinegar variable generation allow recovery of the secret space. Simulations support our findings.

Our detailed examination highlights the critical need for fault attack countermeasures to implement SNOVA. Therefore, we also provide a countermeasure and its corresponding analysis to counteract our fault attacks. Finally, our findings underscore the importance of incorporating comprehensive security strategies to protect against this type of attack, ensuring the integrity and reliability of cryptographic systems.

Paper organization The paper is structured as follows: Section 2 outlines the SNOVA signature scheme. Section 3 details fault attacks on SNOVA, including attack scenarios and key recovery algorithms. Section Section 4 validates these attacks. Section 5 proposes countermeasures against such attacks. Section 6 concludes with implications and future directions.

2 A simple non-commutative UOV scheme

SNOVA is a recently proposed UOV-like signature scheme, as outlined in [28], and has been submitted to the NIST competition for Post-Quantum Digital Signature Schemes.

Let $v, o, l \in \mathbb{N}$ with $v > o$ and $\mathbb{F}_q$ a finite field with q being a power of a prime number. Set $n = v + o$ and $m = o$. By $[m]$ we denote the set $\{1, \ldots, m\}$ and by $\mathcal{R}$ we denote the ring of $l \times l$ matrices over the finite field $\mathbb{F}_q$. Also, by $U = (U_1, \ldots, U_n)^t \in \mathcal{R}^n$ we denote a column vector with n entries from $\mathcal{R}$. Let $Q \in \mathcal{R}$, we denote by Λ_Q, the diagonal matrix of $nl \times nl$, with Q blocks along the diagonal.

The subring $\mathbb{F}_q[S]$ *of* $\mathcal{R}$ is defined as $\mathbb{F}_q[S] = \{a_0 + a_1 S + \ldots + a_{l-1} S^{l-1} | a_0, a_1, \ldots, a_{l-1} \in \mathbb{F}_q\}$, where S is a $l \times l$ symmetric matrix with irreducible characteristic polynomial. Note that the elements in $\mathbb{F}_q[S]$ are symmetric and all commute. Additionally, the non-zero elements in $\mathbb{F}_q[S]$ are invertible. In particular, $\mathbb{F}_q[S]$ is a finite field with $\mathbb{F}_q[S] \cong \mathbb{F}_{q^l}$.

The central map is defined as $\mathcal{F} = [\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m] : \mathcal{R}^n \to \mathcal{R}^m$, where $\mathcal{F}_i$ is given by $\mathcal{F}_k(X_1, X_2, \ldots, X_n) = \sum_{\alpha=1}^{l^2} A_\alpha \cdot \big(\sum_{(i,j)\in\Omega} X_i^t \cdot (Q_{\alpha 1} F_{k,ij} Q_{\alpha 2}) X_j\big) \cdot B_\alpha$ where $\Omega = \{(i,j) : 1 \le i,j \le n\} \setminus \{(i,j) : m+1 \le i,j \le n\}$, $F_{k,ij} \xleftarrow{\$} \mathcal{R}$, $A_\alpha, B_\alpha \xleftarrow{\$} \mathcal{R}$ (invertibles), and $Q_{\alpha 1}, Q_{\alpha 2} \xleftarrow{\$} \mathbb{F}_q[S]$ (invertibles). Set $F_k = [F_{k,ij}]_{(i,j)\in\Omega}$ for each $k \in [m]$.

The invertible linear map is defined as $\mathcal{T} : \mathcal{R}^n \to \mathcal{R}^n$ corresponding to the matrix

$$T = \begin{bmatrix} I^{11} & T^{12} \\ O & I^{22} \end{bmatrix},$$

where T^{12} is a $v \times o$ matrix consisting of nonzero entries $T^{12}_{i,j}$ chosen randomly from $\mathbb{F}_q[S]$, and O is a all-zero matrix. Note that $T^{-1} = T$, if $\mathbb{F}_q$ is of characteristic 2. In addition, I^{11} and I^{22} are identity matrices over $\mathcal{R}$ of size $v \times v$ and $o \times o$ respectively.

Public map is defined as $\mathcal{P} := \mathcal{F} \circ \mathcal{T} = [\mathcal{P}_1 = \mathcal{F}_1 \circ \mathcal{T}, \ldots, \mathcal{P}_m = \mathcal{F}_m \circ \mathcal{T}]$. Set $U = (U_1, \ldots, U_n)^t \in \mathcal{R}^n$, then

$$\mathcal{P}_k(U) = \sum_{\alpha=1}^{l^2} A_\alpha (TU)^t \Lambda_{Q_{\alpha 1}} F_k \Lambda_{Q_{\alpha 2}} (TU) \cdot B_\alpha \tag{1}$$

for any $k \in [m]$. Moreover, $\mathcal{P}_k(U)$ can be written as $\mathcal{P}_k(U) = \mathcal{F}_k(\mathcal{T}(U)) = \sum_{\alpha=1}^{l^2} \sum_{i=1}^{n} \sum_{j=1}^{n} A_\alpha \cdot U_i^t (Q_{\alpha 1} P_{k,ij} Q_{\alpha 2}) U_j \cdot B_\alpha$, where $P_{k,ij} = \sum_{(s,t)\in\Omega} T_{is} F_{k,st} T_{tj}$, by the commutativity of $\mathbb{F}_q[S]$ and that the elements in $\mathbb{F}_q[S]$ are symmetric. Set $P_k = [P_{k,ij}]_{(i,j)\in[n]\times[n]}$ for each $k \in [m]$.

The SNOVA signature scheme [28] consists of a triple of algorithms (`KeyGen`, `Sign`, `Verify`). Moreover, a SNOVA parameter set is given by values for v, o, l, λ.

The `KeyGen` function runs a probabilistic algorithm, and outputs a SNOVA key pair (`sk`, `pk`).

The public key `pk` is a representation of $\mathcal{P}$. A full public key consists of the list of matrices $\left\{P_k = \begin{bmatrix} P_k^{11} & P_k^{12} \\ P_k^{21} & P_k^{22} \end{bmatrix} : k \in [m]\right\}$ and the list of matrices $\left\{A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2} : \alpha \in [l^2]\right\}$. However, it is enough to store a tuple of the form $(\texttt{Spublic}, \{P_k^{22}\}_{k\in[m]})$, where `Spublic` is public seed, which is used to regenerate P_k^{11}, P_k^{12} and P_k^{21} for $k \in [m]$, and $A_\alpha, B_\alpha, Q_{\alpha 1}$ and $Q_{\alpha 2}$ for $\alpha \in \{1, \ldots, l^2\}$. Therefore, the public key size is $m \cdot m^2 \cdot l^2$ field elements plus the size of the public seed `Spublic`.

The private key `sk` is a representation of $(\mathcal{F}, \mathcal{T})$. A full private key consists of a matrix T^{12} and the list of matrices $\left\{F_k : k \in [m]\right\}$. In practice, a private seed `Sprivate` is used to generate T^{12}, and the matrices $\{F_k\}_{k\in[m]}$ are computed by exploiting the relation between F_k, P_k along with T^{12}.

Table 2 summarizes current SNOVA parameters, and public and private keys sizes, as well as, signature sizes for each security level defined by λ as it is usually the security parameter.

Table 2: Parameters for SNOVA [18].

Sec. Level	(v, o, q, l, λ)	Public key (B)	Signature (B)	Private key (B)
I	$(37, 17, 16, 2, 128)$	9826(+16)	108(+16)	60008(+48)
	$(25, 8, 16, 3, 128)$	2304(+16)	148.5(+16)	37962(+48)
	$(24, 5, 16, 4, 128)$	1000(+16)	232(+16)	34112(+48)
III	$(56, 25, 16, 2, 192)$	31250(+16)	162(+16)	202132(+48)
	$(49, 11, 16, 3, 192)$	5990(+16)	270(+16)	174798(+48)
	$(37, 8, 16, 4, 192)$	4096(+16)	360(+16)	128384(+48)
V	$(75, 33, 16, 2, 256)$	71874(+16)	216(+16)	515360(+48)
	$(66, 15, 16, 3, 256)$	15188(+16)	364.5(+16)	432297(+48)
	$(60, 10, 16, 4, 256)$	8000(+16)	560(+16)	389312(+48)

The `Sign` function runs a UOV-like signing procedure as shown by Algorithm 1. It digitally signs a message `M` under the private key `sk`. It first samples a `salt` from $\{0,1\}^{2\lambda}$, then sets $Y \leftarrow \mathcal{H}_1(\texttt{Spublic}||\mathcal{H}_0(\texttt{M})||\texttt{salt})$ where $Y \in \mathcal{R}^m$. The algorithm then chooses random values $V_1, \ldots, V_v \in \mathcal{R}$ as the vinegar variables. Then, it attempts to find the values $(V_{v+1}, \ldots, V_n)$ by solving the equation $\mathcal{F}(V_1, \ldots, V_v, V_{v+1}, \ldots, V_n) = Y$. If no solution to the equation is found, the algorithm will choose random values $V_1', \ldots, V_v' \in \mathcal{R}$ and repeat the procedure until it finds a solution to the equation. Let $X = (V_1, \ldots, V_v, V_{v+1}, \ldots, V_n)^t$ be the solution to the equation. This algorithm then computes the signature as $\mathtt{S} = \mathcal{T}^{-1}(X)$ and outputs $(\mathtt{S}, \mathtt{salt})$.

The `Verify` function runs a deterministic algorithm. It simply verifies if a signature $(\mathtt{S}, \mathtt{salt})$ for `M` is valid under the public key `pk`. If $\mathcal{H}_1(\mathtt{Spublic}||\mathcal{H}_0(\mathtt{M})||\mathtt{salt}) = \mathcal{P}(\mathtt{S})$, then the signature is accepted, otherwise it is rejected. Further details about this function are provided in Appendix A.

2.1 Reconciliation attack.

Ikematsu, and Akiyama [13], Li and Ding [17], and Nakamura, Tani, and Furue [21] analyzed the security of SNOVA against key-recovery attacks, unveiling all known key-recovery attacks for an instance of SNOVA can be seen as key-recovery attacks to instances of an equivalent UOV signature scheme. Particularly, [13] and [17] concluded that all known key-recovery attacks for SNOVA with parameters (v, o, l, q) can be seen as attacks to a UOV signature scheme with lo^2 equations and $l(v + o)$ variables over $\mathbb{F}_q$. In particular, for the *reconciliation* attack, the attacker must find a specific solution $\mathbf{u}_0 \in \mathbb{F}_q^{ln}$ from among many solutions of a quadratic polynomial system of the form

$$\mathbf{u}_0^t(\Lambda_{S^i} P_k \Lambda_{S^j})\mathbf{u}_0 = 0 \in \mathbb{F}_q, \tag{2}$$

for $k \in [m], i, j \in \{0, 1, \dots, l-1\}$.

Once $\mathbf{u}_0$ is found, any $\mathbf{u}$ in the linearly independent set $\{\Lambda_{S^j}\mathbf{u}_0 : 0 \le j \le l-1\}$ will also satisfy Eq. (2). Additionally, the remaining vectors in the secret space $\mathcal{O}$ can be determined by leveraging the fact that for any $\mathbf{u}, \mathbf{v} \in \mathcal{O}$, it holds

$$\mathbf{v}^t(\Lambda_{S^i} P_k \Lambda_{S^j})\mathbf{u} = 0 \in \mathbb{F}_q \text{ for } k \in [m], 0 \le i, j \le l-1. \tag{3}$$

Finally, for any $U \in \mathcal{K}$, it holds $\mathcal{P}_k(U) = 0$ for all $k \in [m]$, where

$$\mathcal{K} := \mathcal{O} \otimes \mathbb{F}_q^l = \{\mathbf{u} \otimes \mathbf{e}^t \in \mathcal{R}^n : \mathbf{u} \in \mathcal{O}, \mathbf{e} \in \mathbb{F}_q^l\}. \tag{4}$$

Thus, the complexity of the *reconciliation* attack is dominated by finding a solution to the quadratic system in Eq. (2). A recent paper [9] introduces a new algorithm that exploits the stability of the quadratic system in Eq. (2) under the action of a commutative group of matrices, reducing the complexity of solving SNOVA systems, over generic ones. In particular, they show how their new algorithm decreases the complexity of solving such a system. On the other hand, we here explore other directions by introducing a new fault-assisted reconciliation attack in Section 3.6. This attack leverages induced transient faults to recover the secret key space by solving the system as mentioned earlier.

3 Fault analysis on SNOVA

3.1 Adversarial Model

We adopt an adversarial model similar to [16], focused on UOV and RAINBOW. Here, the attacker targets the signature generation process by inducing transient or permanent faults in Algorithm 1, manipulating values during execution. The attacker may not know the exact number or content of the manipulated values. By invoking the faulty Algorithm 1 multiple times to collect message-signature pairs, the attacker aims to analyze these pairs to extract partial private key information.

3.2 Attack strategy by fixing field elements of the central map

1. The attacker causes a single permanent fault, which affects line 4 of Algorithm 1, such that some $\mathbb{F}_q$ elements in $F_i \in \mathbb{F}_q^{ln\times ln}$, for $i \in I \subseteq [m]$ and $|I| \geq 1$, are fixed and unknown. In particular, for F_i, there is a fixed non-empty subset $J_i \subseteq [nl] \times [nl]$, such that $\bar{F}_{i,(r_0,r_1)} \in \mathbb{F}_q$, $(r_0, r_1) \in J_i$ is fixed and unknown. We remark the line 4 in practice is an expansion of a private seed along with other operations to compute the list of matrices $\{F_k\}_{k\in[m]}$.
2. For each $\omega \in [N_{msg}]$, the attacker calls Algorithm 1 for the randomly chosen message $\mathtt{M}^{(\omega)} \in \mathcal{R}^m$ and receives the signatures $(\mathtt{S}^{(\omega)}, \mathtt{salt}^{(\omega)})$.

Let $\bar{\mathcal{F}}$ be the faulty central map and $\bar{\mathcal{P}} = \bar{\mathcal{F}} \circ \mathcal{T}$ be the faulty public map. Recall that the SNOVA public map is defined as $\mathcal{P}_k(\mathtt{S}^{(\omega)}) = \sum_{\alpha=1}^{l^2} A_\alpha (T\mathtt{S}^{(\omega)})^t \Lambda_{Q_{\alpha1}} F_k \Lambda_{Q_{\alpha2}} (T\mathtt{S}^{(\omega)}) \cdot B_\alpha$ for any $k \in [m]$. Therefore, it holds

$$\bar{\mathcal{P}}_k(\mathtt{S}^{(\omega)}) - \mathcal{P}_k(\mathtt{S}^{(\omega)}) = \sum_{\alpha=1}^{l^2} A_\alpha (\mathtt{V}^\omega)^t \Lambda_{Q_{\alpha1}} (\bar{F}_k - F_k) \Lambda_{Q_{\alpha2}} \mathtt{V}^\omega \cdot B_\alpha \tag{5}$$

where $T\mathtt{S}^{(\omega)} = \mathtt{V}^{(\omega)}$ with $\mathtt{V}^{(\omega)} = (\mathtt{V}_1^{(\omega)}, \ldots, \mathtt{V}_n^{(\omega)})^t$, by the line 23 of Algorithm 1.

We remark the attacker can compute the left hand of Eq. (5), since $\bar{\mathcal{P}}_k(\mathtt{S}^{(\omega)}) = \mathcal{H}_1(\mathtt{Spublic}||\mathcal{H}_0(\mathtt{M}^{(\omega)})||\mathtt{salt}^{(\omega)})_k$ and $\mathcal{P}$ is public.

Moreover, for any $i \in [m] \setminus I$, both sides of Eq. (5) vanish. However, for any $i \in I$, the left hand of Eq. (5) is expected to be a non-zero element in $\mathcal{R}$, and $\tilde{F}_i = \bar{F}_i - F_i \in \mathbb{F}_q^{ln\times ln}$, on the right side of Eq. (5), is expected to become a sparse matrix, since the entries $\tilde{F}_{i,st} \in \mathbb{F}_q$ with $(s,t) \in J_i$ are the only ones expected to be non-zero.

However, we note that these observations may not be easily exploitable for the attacker to gain information on T, since the attacker does not know $I, \tilde{F}_i, J_i, \mathtt{V}^{(\omega)}$ and $\bar{P}_i$. Therefore, our next scenario focuses on inducing a permanent fault affecting the line 13 of Algorithm 1 to further exploit the relation $T\mathtt{S}^{(\omega)} = \mathtt{V}^{(\omega)}$.

3.3 Attack strategy by fixing field elements of vinegar variables

1. The attacker introduces a single permanent fault, which affects line 13 of Algorithm 1, causing certain $\mathbb{F}_q$ elements in $\mathtt{V}_i \in \mathcal{R}$, for $i \in I \subseteq [v]$ with $|I| \geq 1$, to be fixed and unknown. Specifically, for each variable $\mathtt{V}_i$, there is a fixed non-empty subset $J_i \subseteq [l] \times [l]$, such that $\bar{\mathtt{V}}_{i,(r_0,r_1)} \in \mathbb{F}_q$ for $(r_0, r_1) \in J_i$ is fixed and unknown.

Algorithm 1: Signs message `M`

```
   Input: $v, o, l, \lambda, \mathtt{sk}, \mathtt{spublic}, \mathtt{M}$
   Output: $(\mathtt{S}, \mathtt{salt})$
1  Function sign($v, o, l, \lambda, sk, spublic, M$):
2      $m \leftarrow o$;
3      $n \leftarrow o + v$;
4      $(\{F_k^{11}\}_{k\in[m]}, \{F_k^{12}\}_{k\in[m]}, \{F_k^{21}\}_{k\in[m]}, T^{12}) \leftarrow \mathtt{sk}$;
5      $\{A_\alpha\}_{\alpha\in[l^2]}, \{B_\alpha\}_{\alpha\in[l^2]}, \{Q_{\alpha 1}\}_{\alpha\in[l^2]}, \{Q_{\alpha 2}\}_{\alpha\in[l^2]} \leftarrow PRG(\mathtt{spublic})$;
6      $\mathtt{digest} \leftarrow \mathcal{H}_0(\mathtt{M})$;
7      $\mathtt{salt} \xleftarrow{R} \{0,1\}^\lambda$;
8      $\mathtt{is\_done} \leftarrow \mathtt{False}$;
9      $cont \leftarrow 0$;
10     $[Y_1, Y_2, \ldots, Y_m] \leftarrow \mathcal{H}_1(\mathtt{spublic}||\mathtt{digest}||\mathtt{salt})$;
11
       $$F_k \leftarrow \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{i=1}^{v}\sum_{j=1}^{v} X_i^t (Q_{\alpha 1} F_{k,ij}^{11} Q_{\alpha 2}) X_j\right) \cdot B_\alpha
         + \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{i=1}^{v}\sum_{j=1}^{m} X_i^t (Q_{\alpha 1} F_{k,ij}^{12} Q_{\alpha 2}) X_j\right) \cdot B_\alpha
         + \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{j=1}^{m}\sum_{i=1}^{v} X_j^t (Q_{\alpha 1} F_{k,ji}^{21} Q_{\alpha 2}) X_i\right) \cdot B_\alpha.$$
12     while not is_done do
13         $[\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v] \leftarrow PRG(\mathtt{Sprivate}||\mathtt{digest}||\mathtt{salt}||cont)$;
14         Compute $F_{k,VV} \leftarrow \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{i=1}^{v}\sum_{j=1}^{v} \mathtt{V}_i^t (Q_{\alpha 1} F_{k,ij}^{11} Q_{\alpha 2}) \mathtt{V}_j\right) \cdot B_\alpha$ for all
            $k \in [m]$;
15         Express
           $$Y_k - F_{k,VV} = \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{i=1}^{v}\sum_{j=1}^{m} \mathtt{V}_i^t (Q_{\alpha 1} F_{k,ij}^{12} Q_{\alpha 2}) X_j\right) \cdot B_\alpha
             + \sum_{\alpha=1}^{l^2} A_\alpha \left(\sum_{j=1}^{m}\sum_{i=1}^{v} X_j^t (Q_{\alpha 1} F_{k,ji}^{21} Q_{\alpha 2}) \mathtt{V}_i\right) \cdot B_\alpha.$$
            for all $k \in [m]$ as an equation system on the oil variables $\overrightarrow{X_1}, \overrightarrow{X_2}, \ldots, \overrightarrow{X_m}$;
16
           $$M_{1,1}\overrightarrow{X_1} + M_{1,2}\overrightarrow{X_1} + \ldots + M_{1,m}\overrightarrow{X_m} = \overrightarrow{Y_1} - \overrightarrow{F_{1,VV}}$$
           $$M_{2,1}\overrightarrow{X_1} + M_{2,2}\overrightarrow{X_2} + \ldots + M_{2,m}\overrightarrow{X_m} = \overrightarrow{Y_2} - \overrightarrow{F_{2,VV}}$$
           $$\vdots$$
           $$M_{m,1}\overrightarrow{X_1} + M_{m,2}\overrightarrow{X_2} + \ldots + M_{m,m}\overrightarrow{X_m} = \overrightarrow{Y_m} - \overrightarrow{F_{m,VV}}$$
17         Represent this equation system as an $(ml^2) \times (ml^2 + 1)$ matrix $\mathbf{A}$ over $\mathbb{F}_{16}$;
18         $L_O, \mathtt{output} \leftarrow \mathtt{Gauss}(\mathbf{A})$;
19         if output then
20             $[\mathtt{V}_{v+1}, \mathtt{V}_{v+2}, \ldots, \mathtt{V}_n] \leftarrow L_O$;
21             $\mathtt{V} \leftarrow [\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v, \mathtt{V}_{v+1}, \ldots, \mathtt{V}_n]$;
22             $T \leftarrow \begin{bmatrix} I^{11} & T^{12} \\ O & I^{22} \end{bmatrix}$;
23             $\mathtt{S} \leftarrow T \cdot \mathtt{V}^t$;
24             $\mathtt{is\_done} \leftarrow \mathtt{True}$;
25         end
26         else
27             $cont \leftarrow cont + 1$;
28         end
29     end
30     return $(S, salt)$;
```

2. For each $\omega \in [N_{msg}]$, the attacker calls Algorithm 1 for the randomly chosen message $\mathtt{M}^{(\omega)} \in \mathcal{R}^m$ and receives the signature $(\mathtt{S}^{(\omega)}, \mathtt{salt}^{(\omega)})$.
3. The attacker then calls Algorithm 2 with parameters $v, o, l, [\mathtt{S}^{(1)}, \dots, \mathtt{S}^{(N_{msg})}]$ to obtain $\mathtt{dic}$, a dictionary-like data structure indexed by $[v] \times [l]^2$. When $N_{msg} > lo + 1$, Algorithm 2 will produce $\mathtt{dic}$ such that $\mathtt{dic}[(i, r_0, r_1)] = [T^{12}_{i1}, \dots, T^{12}_{io}]$ for $i \in I$ and $(r_0, r_1) \in J_i$, and $\mathtt{dic}[(i, r_0, r_1)] = \mathtt{None}$ otherwise.

Algorithm 2: Partially recovers T^{12} from fixed field elements.

```
Function recover_F16($v, o, l, [\mathtt{S}^{(1)}, \dots, \mathtt{S}^{(N_{msg})}]$):
    $S \leftarrow$ getS($l$);
    dic $\leftarrow$ {};
    for $(i, r_0, r_1) \in [v] \times [l] \times [l]$ do
        $\mathbf{A} \leftarrow \mathbb{F}_{16}^{(N_{msg}-1)\times ol}$;
        $\mathbf{Y} \leftarrow \mathbb{F}_{16}^{(N_{msg}-1)\times 1}$;
        for $\omega \leftarrow 2$ to $N_{msg}$ do
            $\mathtt{S}_i^{(\omega,1)} \leftarrow \mathtt{S}_i^{(1)} - \mathtt{S}_i^{(\omega)}$;
            for $(j, j_1) \in [o] \times [l]$ do
                $\mathtt{S}_{v+j}^{(j_1,k,1)} \leftarrow S^{j_1-1}(\mathtt{S}_{v+j}^{(\omega)} - \mathtt{S}_{v+j}^{(1)})$;
                $\mathbf{A}_{(\omega-1),j\cdot l+j_1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1)}^{(j_1,k,1)}$;
            end
            $\mathbf{Y}_{(\omega-1),0} \leftarrow \mathtt{S}_{i,(r_0,r_1)}^{(\omega,1)}$;
        end
        (X, output) $\leftarrow$ Gauss($\mathbf{A}, \mathbf{Y}$);
        if output then
            $[T^{12}_{i1}, \dots, T^{12}_{io}] \leftarrow$ get_elements_in_FqS(X, $l$);
            dic$[(i, r_0, r_1)] \leftarrow [T^{12}_{i1}, \dots, T^{12}_{io}]$;
        end
        else
            dic$[(i, r_0, r_1)] \leftarrow$ None;
        end
    end
    return dic;
```

Why is Step 3 of the attack strategy expected to work correctly? As seen in Section 2, the invertible linear map $\mathcal{T}$ for the SNOVA scheme is given by the matrix

$$T = \begin{bmatrix} I^{11} & T^{12} \\ O & I^{22} \end{bmatrix},$$

where T^{12} is a $v \times o$ matrix with nonzero entries T^{12}_{ij} chosen randomly from $\mathbb{F}_q[S]$. From the line 23 of Algorithm 1, it holds $T\mathtt{S}^{(\omega)} = \mathtt{V}^{(\omega)}$, that is,

$$\begin{bmatrix} 1 & 0 & \dots & 0 & T^{12}_{11} & \dots & T^{12}_{1o} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & T^{12}_{v1} & \dots & T^{12}_{vo} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} \mathtt{S}_1^{(\omega)} \\ \vdots \\ \mathtt{S}_v^{(\omega)} \\ \vdots \\ \mathtt{S}_n^{(\omega)} \end{bmatrix} = \begin{bmatrix} \mathtt{V}_1^{(\omega)} \\ \vdots \\ \mathtt{V}_v^{(\omega)} \\ \vdots \\ \mathtt{V}_n^{(\omega)} \end{bmatrix}.$$

Since $T^{12}_{ij} \in \mathbb{F}_q[S]$, it holds $T^{12}_{ij} = \sum_{j_1=1}^{l} t^{12}_{ij,j_1} S^{j_1-1}$, where $t^{12}_{ij,j_1} \in \mathbb{F}_q$, and

$$\mathtt{S}_i^{(\omega)} + \sum_{j=1}^{o} T_{ij}^{12} \mathtt{S}_{v+j}^{(\omega)} = \mathtt{S}_i^{(\omega)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} S^{j_1-1} \mathtt{S}_{v+j}^{(\omega)} \quad = \mathtt{V}_i^{(\omega)},\ i \in [v]$$

$$\mathtt{S}_i^{(\omega)} = \mathtt{V}_i^{(\omega)},\ v+1 \le i \le n$$

For $2 \le \omega \le N_{msg}$, we can write

$$\mathtt{S}_i^{(\omega)} - \mathtt{S}_i^{(1)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} S^{j_1-1}(\mathtt{S}_{v+j}^{(\omega)} - \mathtt{S}_{v+j}^{(1)}) = \mathtt{V}_i^{(\omega)} - \mathtt{V}_i^{(1)},\ i \in [v]$$

$$\mathtt{S}_i^{(\omega)} - \mathtt{S}_i^{(1)} = \mathtt{V}_i^{(\omega)} - \mathtt{V}_i^{(1)},\ v+1 \le i \le n$$

Let us fix $2 \le \omega \le N_{msg}$ and $i \in [v]$. Also, let us set $\mathtt{S}_i^{(\omega,1)} = \mathtt{S}_i^{(\omega)} - \mathtt{S}_i^{(1)}$, $\mathtt{S}_{v+j}^{(j_1,\omega,1)} = S^{j_1-1}(\mathtt{S}_{v+j}^{(\omega)} - \mathtt{S}_{v+j}^{(1)})$ and $\mathtt{V}_i^{(\omega,1)} = \mathtt{V}_i^{(\omega)} - \mathtt{V}_i^{(1)}$. Consider

$$\mathtt{S}_i^{(\omega,1)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} \mathtt{S}_{v+j}^{(j_1,\omega,1)} = \mathtt{V}_i^{(\omega,1)}. \tag{6}$$

Since Eq. (6) is defined over $\mathcal{R}$, it is equivalent to l^2 equations on lo unknowns over $\mathbb{F}_q$. Therefore,

$$\mathtt{S}_{i,(r_0,r_1)}^{(\omega,1)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} \mathtt{S}_{v+j,(r_0,r_1)}^{(j_1,\omega,1)} = \mathtt{V}_{i,(r_0,r_1)}^{(\omega,1)},\ \text{for } r_0, r_1 \in [l]. \tag{7}$$

The attacker can compute $\mathtt{S}^{(\omega,1)}$ and $\mathtt{S}^{(j_1,\omega,1)}$ on the left hand of Eq. (6). Additionally, for a fixed $i \in I$ and for $2 \le \omega \le N_{msg}$, a linear system of $(N_{msg} - 1)|J_i|$ equations and lo unknowns over $\mathbb{F}_q$ can be obtained and is given by

$$\left\{\mathtt{S}_{i,(r_0,r_1)}^{(\omega,1)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} \mathtt{S}_{v+j,(r_0,r_1)}^{(j_1,\omega,1)} = 0,\ \text{for } (r_0,r_1) \in J_i\right\}_{2 \le \omega \le N_{\mathrm{msg}}} \tag{8}$$

Furthermore, if J_i is known by the attacker for such a i, collecting $N_{msg} > \frac{o \cdot l}{|J_i|} + 1$ would be enough to guarantee an unique solution to the linear system of Eq. (8) and recover the i-th row of T^{12}.

However, if the attacker does not know either I or J_i for $i \in I$, the attacker may still gain knowledge of I and J_i for $i \in I$, and recover $T_{i,j}^{12}$ for $i \in I, j \in [o]$, by trying to solve $v \cdot l^2$ linear systems separately, i.e. one for $i \in [v]$ and $(r_0, r_1) \in [l]^2$,

$$\{\mathtt{S}_{i,(r_0,r_1)}^{(\omega,1)} + \sum_{j=1}^{o}\sum_{j_1=1}^{l} t_{ij,j_1}^{12} \mathtt{S}_{v+j,(r_0,r_1)}^{(j_1,\omega,1)} = 0\}_{2 \le \omega \le N_{msg}}, \tag{9}$$

where each has $N_{msg} - 1$ equations and lo unknowns. If $N_{msg} > l \cdot o + 1$, then the linear systems for $(r_0, r_1) \in J_i, i \in I$ will have a unique solution, while the other linear systems are expected to have no solution. Therefore, when $N_{msg} > lo + 1$, Algorithm 2 will output `dic` such that $\texttt{dic}[(i, r_0, r_1)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I, (r_0, r_1) \in J_i$ and $\texttt{dic}[(i, r_0, r_1)] = \texttt{None}$ otherwise.

Furthermore, if the attacker is able to fix at least an entry of each vinegar variable (i.e. $I = [v]$ and so $|J_i| \geq 1$) and collect at least $lo + 2$ signatures, Algorithm 2 will recover the entire matrix T^{12}.

3.4 What if the attacker only can fix some bits of V_i for $i \in I$?

In this section, we assume a variable $\texttt{V}_i$ is represented as a bit-string of length $N_{bits} \cdot l^2$, where $q = 2^{N_{bits}}$ as it is the case for SNOVA. The attack strategy is as follows.

1. The attacker causes a single permanent fault, which affects the line 13 of Algorithm 1, such that some bits of $\texttt{V}_i \in \mathcal{R}$, with $i \in I \subseteq [v]$ and $|I| \geq 1$, are fixed and unknown. In particular, for the variable $\texttt{V}_i$, there is a fixed non-empty subset $B_i \subseteq [l] \times [l] \times [N_{bits}]$, such that $\bar{\texttt{V}}_{i,(r_0,r_1,b)} \in \mathbb{F}_2$, $(r_0, r_1, b) \in B_i$ is fixed and unknown.
2. For each $\omega \in [N_{msg}]$, the attacker calls Algorithm 1 for the randomly chosen message $\texttt{M}^{(\omega)} \in \mathcal{R}^m$ and receives the signature $(\texttt{S}^{(\omega)}, \texttt{salt}^{(\omega)})$.
3. The attacker calls Algorithm 3 with parameters $v, o, l, [\texttt{S}^{(1)}, \ldots, \texttt{S}^{(N_{msg})}]$ to get `dic`, a dictionary-like data structure indexed by $[v] \times [l]^2 \times [N_{bits}]$. When $N_{msg} > N_{bits} \cdot lo + 1$, Algorithm 3 will output `dic` such that $\texttt{dic}[(i, r_0, r_1, b)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I, (r_0, r_1, b) \in B_i$ and $\texttt{dic}[(i, r_0, r_1, b)] = \texttt{None}$ otherwise.

Why is Step 3 of the attack strategy expected to work correctly? Since $\mathbb{F}_q$ can be seen as a vector space of dimension N_{bits} over $\mathbb{F}_2$, we can obtain similar equations to those of Eq. (8). That is, for $i \in I$, we have

$$\left\{ \texttt{S}^{(\omega,1)}_{i,(r_0,r_1,b)} + \sum_{j=1}^{o} \sum_{j_1=1}^{l} \kappa^{i,j,j_1,\omega,1}_{(r_0,r_1,b)} = 0, \text{ for } (r_0, r_1, b) \in B_i \right\}_{2 \leq \omega \leq N_{\text{msg}}} \tag{10}$$

where $\kappa^{i,j,j_1,\omega,1}_{(r_0,r_1)} = t^{12}_{ij,j_1} \texttt{S}^{(j_1,\omega,1)}_{v+j,(r_0,r_1)}$. Eq. (10) represents a linear system with $|B_i| \cdot (N_{msg} - 1)$ equations and $N_{bits} \cdot l \cdot o$ unknowns over $\mathbb{F}_2$. However, the attacker does not know either I or B_i. For the attacker to gain knowledge of I and B_i for $i \in I$, and recover $T^{12}_{i,j}$ for $i \in I, j \in [o]$, the attacker may try to solve $N_{bits} \cdot v \cdot l^2$ linear systems separately, i.e. one for $i \in [v]$ and $(r_0, r_1, b) \in [l] \times [l] \times [N_{bits}]$,

$$\{\texttt{S}^{(\omega,1)}_{i,(r_0,r_1,b)} + \sum_{j=1}^{o} \sum_{j_1=1}^{l} \kappa^{i,j,j_1,\omega,1}_{(r_0,r_1,b)} = 0\}_{2 \leq \omega \leq N_{msg}}, \tag{11}$$

Algorithm 3: Partially Recovers T^{12} from Fixed Bits

```
recover_F2($v, o, l, [\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(N_{msg})}]$)
$S \leftarrow$ getS($l$)
dic $\leftarrow \{\}$
for $(i, r_0, r_1, b) \in [v] \times [l] \times [l] \times [4]$ do
    $\mathbf{A} \leftarrow \mathbb{F}_2^{(N_{msg}-1)\times 4\cdot ol}$
    $\mathbf{Y} \leftarrow \mathbb{F}_2^{(N_{msg}-1)\times 1}$
    for $\omega \leftarrow 2$ to $N_{msg}$ do
        $\mathtt{S}_i^{(\omega,1)} \leftarrow \mathtt{S}_i^{(1)} - \mathtt{S}_i^{(\omega)}$
        for $(j, j_1) \in [o] \times [l]$ do
            $\mathtt{S}_{v+j}^{(j_1,k,1)} \leftarrow S^{j_1-1}(\mathtt{S}_{v+j}^{(\omega)} - \mathtt{S}_{v+j}^{(1)})$
            if $b = 1$ then
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,1)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+2} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+3} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+4} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
            else if $b = 2$ then
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+2} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,1)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+3} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+4} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
            else if $b = 3$ then
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+2} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+3} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,1)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+4} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
            else
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+2} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+3} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+4} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,1)}^{(j_1,k,1)}$
        $\mathbf{Y}_{(\omega-1),0} \leftarrow \mathtt{S}_{i,(r_0,r_1,b)}^{(\omega,1)}$
    (X, output) $\leftarrow$ Gauss($\mathbf{A}, \mathbf{Y}$)
    if output then
        $[T_{i1}^{12}, \ldots, T_{io}^{12}] \leftarrow$ get_elements_in_FqS_from_bits(X, $l$)
        dic$[(i, r_0, r_1, b)] \leftarrow [T_{i1}^{12}, \ldots, T_{io}^{12}]$
    else
        dic$[(i, r_0, r_1, b)] \leftarrow$ None
return dic
```

where each has $N_{msg} - 1$ equations and $N_{bits} \cdot l \cdot o$ unknowns over $\mathbb{F}_2$. If $N_{msg} > N_{bits} \cdot l \cdot o + 1$, then the linear systems for $(r_0, r_1, b) \in B_i, i \in I$ will have a unique solution, while the other linear systems are expected to have no solution. Algorithm 3 details the recovery strategy by the attacker and exploits the fact that $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$ for SNOVA.

Therefore, when $N_{msg} > N_{bits} \cdot lo + 1$, Algorithm 3 will output `dic` such that $\texttt{dic}[(i, r_0, r_1, b)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I, (r_0, r_1, b) \in B_i$ and $\texttt{dic}[(i, r_0, r_1, b)] =$ `None` otherwise.

We remark that even if a permanent fault fixes all bits for some i and the attacker knows it, they can adjust Algorithm 3 to reduce the required signatures to retrieve the row i of T^{12} to $N_{\text{msg}} > \frac{o}{l} + 1$. However, even in the best-case scenario, more than two signatures are needed for recovery.

3.5 How can the attacker recover $T^{12}_{i,j}$ for a fixed $i \in [v] \setminus I, j \in [o]$?

For $1 \le \omega \le N_{msg}$, we have

$$\mathtt{S}_i^{(\omega)} + \sum_{j=1}^{o} T^{12}_{ij} \mathtt{S}^{(\omega)}_{v+j} = \mathtt{S}_i^{(\omega)} + \sum_{j=1}^{o} \sum_{j_1=1}^{l} t^{12}_{ij,j_1} S^{j_1-1} \mathtt{S}^{(\omega)}_{v+j} \quad = \mathtt{V}_i^{(\omega)}, i \in [v] \setminus I,$$

Note that the previous equations can always be arranged as

$$\mathbf{S}_i + \sum_{j=1}^{o} \sum_{j_1=1}^{l} t^{12}_{ij,j_1} \mathbf{S}_{j_1,j} = \mathbf{V}_i$$

with $\mathbf{S}_i = \begin{bmatrix} \mathtt{S}_i^{(1)} \\ \vdots \\ \mathtt{S}_i^{(N_{msg})} \end{bmatrix}^t$, $\mathbf{S}_{j_1,j} = \begin{bmatrix} S^{j_1-1}\mathtt{S}_{v+j}^{(1)} \\ \vdots \\ S^{j_1-1}\mathtt{S}_{v+j}^{(N_{msg})} \end{bmatrix}^t$, $\mathbf{V}_i = \begin{bmatrix} \mathtt{V}_i^{(1)} \\ \vdots \\ \mathtt{V}_i^{(N_{msg})} \end{bmatrix}^t \in \mathcal{R}^{1 \times N_{msg}}$.
This indeed induces an instance of the MinRank problem [8] over $\mathbb{F}_q$. Note that by setting $\mathbf{M} = (\mathbf{S}_i, \mathbf{S}_{1,1}, \ldots, \mathbf{S}_{l,o}) \in (\mathbb{F}_q^{l \times (N_{msg} \cdot l)})^{l \cdot o + 1}$, there exists a $(t_{i1,1}, \ldots, t_{io,l}) \in \mathbb{F}_q^{ol}$ and a matrix $\mathfrak{M} \in \mathbb{F}_q^{l \times (N_{msg}-1) \cdot l}$ such that

$$(\mathbf{S}_i + \sum_{j=1}^{o} \sum_{j_1=1}^{l} t^{12}_{ij,j_1} \mathbf{S}_{j_1,j}) \begin{bmatrix} \mathfrak{I} \\ -\mathfrak{M} \end{bmatrix} = 0$$

where $\mathfrak{I} \in \mathbb{F}_q^{(N_{msg}-1) \cdot l \times (N_{msg}-1) \cdot l}$ is a non-singular matrix.

Discussion of previous scenarios The scenarios described in Sections 3.3 and 3.4 are examples of related randomness attacks [24]. In these attacks, the adversary injects a permanent fault to force the reuse of fixed sub-bitstrings within the $N_{bits} v l^2$ bitstring $\mathtt{V}_{bs}$ representing the vinegar values $\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v$. As a result, partial recovery of the private linear map $\mathcal{T}$ becomes feasible using the techniques in Sections 3.3 and 3.4, provided the attacker can find other methods to fix sub-bitstrings within $\mathtt{V}_{bs}$ and collects enough valid signatures generated using these fixed values.

Additionally, if the attacker can identify a fixed bit in each $\mathtt{V}_i$ during signature generation, they can use Algorithm 3 to recover T with sufficient signatures. Define $\mathcal{J} := [v] \times [l]^2 \times [N_{bits}]$, $\mathcal{G}$, and $\mathcal{O}_{\mathcal{I}}$ as in Algorithm 4. The attacker, given $\mathcal{I} \leftarrow \mathcal{G}()$ and oracle $\mathcal{O}_{\mathcal{I}}$, can proceed with the recovery.

Algorithm 4: Definition of functions $\mathcal{G}$ and $\mathcal{O}_{\mathcal{I}}$.

```
Function $\mathcal{G}()$
    $\mathcal{I} \leftarrow \emptyset$;
    for ($i \leftarrow 1$ to $v$) do
        $(r_0, r_1, b) \xleftarrow{\$} [l] \times [l] \times [N_{bits}]$;
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{(i, r_0, r_1, b)\}$;
    end
    return $\mathcal{I}$;
Function $\mathcal{O}_{\mathcal{I}}(\iota \in \mathcal{J}, b \in \mathbb{F}_2)$
    if $\iota \in \mathcal{J} \setminus \mathcal{I}$ then
        $c \xleftarrow{\$} \mathbb{F}_2$;
        return $c$;
    end
    Let $\mathtt{V}_\iota$ be the random bit chosen by line 13 of Algorithm 1 in the most recent call.;
    if $b = \mathtt{V}_\iota$ then
        return 1;
    end
    return 0;
```

This adversary can leverage his knowledge of $\mathcal{I}$, his access to $\mathcal{O}_{\mathcal{I}}$ and Algorithm 3 to fully recover the private linear transformation $\mathcal{T}$ as follows.

1. The attacker sets $\mathcal{S} = []$, creates the lists $\mathtt{L}_\iota = []$ and sets $b_\iota \xleftarrow{\$} \mathbb{F}_2$ for all $\iota \in \mathcal{I}$.
2. The attacker calls Algorithm 1 for the random message $\mathtt{M}^{(j)}$, which outputs $(\mathtt{S}^{(j)}, \mathtt{salt}^{(j)})$, and then updates $\mathcal{S}.\mathtt{append}((\mathtt{S}^{(j)}, \mathtt{salt}^{(j)}))$.
   Additionally, the attacker updates its lists $\mathtt{L}_\iota$ for all $\iota \in \mathcal{I}$ as follows.
   (a) For each $\iota \in \mathcal{I}$, $\mathtt{L}_\iota.\mathtt{append}(\mathcal{O}_{\mathcal{I}}(\iota, b_\iota))$.
3. After collecting sufficient signatures, N_{msg}, the attacker stops. In particular, once $\sum_{i=1}^{N_{msg}} \mathtt{L}_\iota[i] > N_{bits} \cdot l \cdot o + 1$ for all $\iota \in \mathcal{I}$, it will stop.
4. The attacker then uses the collected signatures and calls Algorithm 3 $|\mathcal{I}|$ times to recover the matrix T.
   (a) For each $\iota = (i, r_0, r_1, b) \in \mathcal{I}$, the attacker creates $\mathcal{S}_\iota = [\mathcal{S}[j]$ for $j \in [N_{msg}]$ if $\mathtt{L}_\iota[j] = 1]$ and calls Algorithm 3 with parameters v, o, l and $\mathcal{S}_\iota$. From Section 3.4, it follows each call of Algorithm 3 with parameters v, o, l and $\mathcal{S}_\iota$ will return $\mathtt{dic}[\iota] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$.

We remark that the previous example scenario is yet another case of related randomness attacks, since the attacker at step 2a marks what signatures share the bit b_ι in $\mathtt{V}_\iota$. However, we stress that we do not know how to instantiate this oracle $\mathcal{O}_{\mathcal{I}}$ in a real scenario effectively, and therefore this question remains open.

3.6 Fault-assisted reconciliation attack

As seen in Section 2.1, for the *reconciliation* attack, the attacker must find a specific solution $\mathbf{u}_0 \in \mathbb{F}_q^{ln}$ from among many solutions to the quadratic system of the form

$$\mathbf{u}_0^t(\Lambda_{S^i} P_k \Lambda_{S^j})\mathbf{u}_0 = 0 \in \mathbb{F}_q, \tag{12}$$

for $k \in [m], i, j \in \{0, 1, \ldots, l-1\}$. Furthermore, for any valid signature $(\mathtt{S}, \mathtt{salt})$, it holds $\mathtt{S} = T^{-1}\mathtt{V}$, with $\mathtt{V} = (\mathtt{V}_1, \ldots, \mathtt{V}_v, \mathtt{O}_1, \ldots, \mathtt{O}_o)^t$ and $T^{-1} = T$. Consequently, for any $\beta \in [l]$, we have

$$\mathtt{S}_{:\beta} = \begin{bmatrix} \mathtt{S}_{1,:\beta} \\ \vdots \\ \mathtt{S}_{v,:\beta} \\ \vdots \\ \mathtt{S}_{n,:\beta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots & 0 & T_{1,1}^{12} & \ldots & T_{1,o}^{12} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & T_{v,1}^{12} & \ldots & T_{v,o}^{12} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \ldots & 0 & \ldots & 1 \end{bmatrix} \begin{bmatrix} \mathtt{V}_{1,:\beta} \\ \vdots \\ \mathtt{V}_{v,:\beta} \\ \mathtt{O}_{1,:\beta} \\ \vdots \\ \mathtt{O}_{o,:\beta} \end{bmatrix}$$

where $\mathtt{S}_{:\beta}$ denotes the β-th column of $\mathtt{S}$. If an attacker knows $\mathtt{V}_{1,:\beta}, \ldots, \mathtt{V}_{v,:\beta}$, then the attacker can set

$$\mathbf{u}_0 = \begin{bmatrix} \mathtt{V}_{1,:\beta} - \sum_{j=1}^{o} T_{1j}^{12}\mathtt{O}_{j,:\beta} \\ \vdots \\ \mathtt{V}_{v,:\beta} - \sum_{j=1}^{o} T_{vj}^{12}\mathtt{O}_{j,:\beta} \\ \mathtt{O}_{1,:\beta} \\ \vdots \\ \mathtt{O}_{o,:\beta} \end{bmatrix}$$

which will satisfy Eq. (12). Therefore, the main task of the attacker is to find $\mathtt{V}_{1,:\beta}, \ldots, \mathtt{V}_{v,:\beta}$ for a valid signature $(\mathtt{S}, \mathtt{salt})$ and some $\beta \in [l]$.

Our fault-assisted reconciliation attack is as follows.

1. The attacker injects transient faults at line 13 of Algorithm 1, targeting $\mathtt{V}_{i,j\beta}$ for all $i \in [v], j \in [l]$, and $\beta \in \mathcal{C} \subseteq [l]$. For each $\mathtt{V}_i$, there is a fixed non-empty subset $J_i \subseteq [l] \times \mathcal{C}$ such that $\bar{\mathtt{V}}_{i,(r_0,r_1)} = \omega$ for $(r_0, r_1) \in J_i$, where $\omega \in \mathbb{F}_q$ is an unknown fixed value.
2. The attacker calls Algorithm 1 with a random message $\mathtt{M} \in \mathcal{R}^m$ to obtain the signature $(\mathtt{S}, \mathtt{salt})$.
3. If Step 2 succeeds, Algorithm 5 is executed with parameters $v, o, l, S, \mathcal{C}, \Gamma_\beta$ for $\beta \in \mathcal{C}$[1], where $\Gamma_\beta \subseteq [lv]$. Here, F_γ represents all subsets of γ integers from $[lv]$, and $A^c = [lv] \setminus A$ for $A \in F_\gamma$.

By selecting appropriate Γ_β, the attacker ensures the quadratic systems at line 9 of Algorithm 5 have ol^2 equations and $lv - \gamma < ol^2$ unknowns, typically yielding no or few solutions.

[1] If $\mathcal{C}$ is unknown, set $\mathcal{C} = [l]$.

Let $\mathtt{V} = (\mathtt{V}_1^t, \ldots, \mathtt{V}_v^t)^t \in \mathbb{F}_q^{vl^2}$. If Steps 1 and 2 result in $\mathtt{V}_{i\beta} = \omega$ for $i \in A$, with $A \in F_\gamma$ and $\gamma \in \Gamma_\beta$, Algorithm 5 will find $\mathtt{U}$ satisfying Eq. (12) and the secret space $\mathcal{O}$. Otherwise, the attack restarts. Runtime complexity is analyzed in Appendix C.

Algorithm 5: Attempts to find $\mathcal{O}$ after having run the attack strategy.

```
    Input: $v, o, l, \mathtt{S}, \mathcal{C}, \Gamma_\beta$ for $\beta \in \mathcal{C}$
    Output: $\mathcal{O}$ or $\perp$
 1  Function fault_assisted_reconcilation_attack($v, o, l, \mathcal{S}, \mathcal{C}, \Gamma_\beta$):
 2      for $\beta \in \mathcal{C}$ do
 3          for $\gamma \in \Gamma_\beta$ do
 4              for $A \in F_\gamma$ do
 5                  for $\omega \in \mathbb{F}_q$ do
 6                      Set $\Omega \leftarrow (x_1, \ldots, x_{lv}, 0, \ldots, 0)^t \in \mathbb{F}_q^{ln}$;
 7                      Set $\Omega_i \leftarrow w$ for $i \in A$;
 8                      $\mathtt{X} \leftarrow \mathtt{S}_{:\beta} - \Omega$;
 9                      Attempt to solve the quadratic system:

                                $\mathtt{X}^t(\Lambda_{\mathcal{S}^i} P_k \Lambda_{\mathcal{S}^j})\mathtt{X} = 0 \in \mathbb{F}_q,$

                         for $k \in [m], i, j \in \{0, 1, \ldots, l-1\}$. This system has $ml^2$ equations
                         and $lv - \gamma$ unknowns, namely $x_i$ for $i \in A^c$;
10                      if $x_i$ for $i \in A^c$ are found then
11                          Set $\mathtt{U}$ as the solution;
12                          Recover $\mathcal{O}$ from the linearly independent set
                              $\{\Lambda_{\mathcal{S}^j}\mathtt{U} : 0 \leq j \leq l-1\}$;
13                          return $\mathcal{O}$;
14                      end
15                  end
16              end
17          end
18      end
19      return $\perp$;
```

Success Probability of our Attack Strategy. Let $X_{ij} \in \{0, 1\}$ be a Bernoulli random variable indicating whether $\mathtt{V}_{ij}$ is fixed to ω due to a transient fault, with $\Pr(X_{ij} = 1) = p_{ij}$. Define $Y_\beta = \sum_{i=1}^{lv} X_{i\beta}$. The probability of γ successful fixes in the β-th column of $\mathtt{V}$ is:

$$\Pr(Y_\beta = \gamma) = \sum_{A \in F_\gamma} \prod_{i \in A} p_{i\beta} \prod_{j \in A^c} (1 - p_{j\beta}).$$

Let $\rho_\beta = \sum_{\gamma \in \Gamma_\beta} \Pr(Y_\beta = \gamma)$, and $\rho = \max\{\rho_\beta : \beta \in \mathcal{C}\}$, representing the success probability of Algorithm 5. The overall success probability of the attack strategy is $\rho(1 - \delta)$, where δ is the failure probability of Step 2. If p_{ij} remains constant, the attacker expects to run the strategy $1/\rho(1 - \delta)$ times.

If Step 1 is implemented via a single transient fault that targets $\mathtt{V}_{i\beta}$ for all $i \in [lv], \beta \in [l]$, the attacker runs the strategy $1/(1-\delta)$ times if $p_{i\beta} = 1$. However, if $\epsilon \leq p_{i\beta} \leq 1$, choosing a proper Γ_ϵ yields

$$(1-\delta)\sum_{\gamma\in\Gamma_\epsilon}\binom{lv}{\gamma}\epsilon^\gamma(1-\epsilon)^{lv-\gamma} \le (1-\delta)\rho \le (1-\delta).$$

Targeting $\mathtt{V}_{i\beta}$ for all $i \in [lv]$ and $\beta \in \mathcal{C}$ with $|\mathcal{C}| = 1$ can further improve success probability, since, in such cases, δ is expected to be very low. If $\epsilon \le p_{i\beta} \le 1$, setting Γ_ϵ such that $\sum_{\gamma\in\Gamma_\beta}\binom{lv}{\gamma}\epsilon^\gamma(1-\epsilon)^{lv-\gamma} \approx 1$ may allow the attacker to run the strategy[2] once. Simulations in Section 4.2 analyze these scenarios.

3.7 Alternative Versions of SNOVA

The SNOVA team released a preprint [27] that proposes two new versions of SNOVA to counteract Buellens' attack [6].

The first alternative version of SNOVA chooses random matrices $A_{k,\alpha}, B_{k,\alpha} \in \mathcal{R}$ and $Q_{k,\alpha 1}, Q_{k,\alpha 2} \in \mathbb{F}_q[S]$, for $k \in [o]$ and $\alpha \in [l^2]$, and define the k-th coordinate of the public map $\mathcal{P}(U)$ as

$$\mathcal{P}_k(U_1,\ldots,U_n) = \sum_{\alpha=1}^{l^2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{k,\alpha}\cdot U_i^t(Q_{k,\alpha 1}P_{k,i,j}Q_{k,\alpha 2})U_j\cdot B_{k,\alpha}.$$

The second alternative version of SNOVA defines the k-th coordinate of the public map $\mathcal{P}(U)$ as follows

$$\mathcal{P}_k(U) = \sum_{\alpha=1}^{l^4}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{\alpha}\cdot U_i^t(Q_{\alpha 1}P_{k,i,j}Q_{\alpha 2})U_j\cdot B_{\alpha},$$

where the matrices $A_\alpha, B_\alpha \in \mathcal{R}$, and $Q_{\alpha 1}, Q_{\alpha 2} \in \mathbb{F}_q[S]$, for $\alpha \in [l^4]$, are determined by fixed matrices $\tilde{E}_{i,j} \in \mathbb{F}_q^{l^2\times l^2}$, for $i, j \in [l]$, specified in [27].

We remark that either alternative version would be *still vulnerable to our fault strategies* described in Sections 3.3 and 3.4, since these strategies exploit the related randomness present in the vinegar variables $\mathtt{V}^{(\omega)} = (\mathtt{V}_1^{(\omega)},\ldots,\mathtt{V}_n^{(\omega)})^t$ when a permanent fault has been established and the relation $\mathtt{S}^{(\omega)} = T^{-1}\mathtt{V}^{(\omega)}$. Additionally, the proposed alternatives do not affect the reconciliation attack. However, we remark our experiments reported in Section 4 were carried out on the Round 1 SNOVA reference implementation.

4 Experiments of our fault attacks

We conducted experiments to validate our fault attack, detailing the procedure and results. We implemented the SNOVA signature scheme in SAGE, following its specification [18], and used the latest SNOVA code[3] to generate signatures. Faults were introduced by fixing specific values in the vinegar variables, replicating the fault injection process in Section 3.1.

[2] Runtime depends on Γ_ϵ.

[3] `https://github.com/PQCLAB-SNOVA/SNOVA` (commit 3d7e8c7cebdd57293d74dc6c2608656697b99597)

4.1 Simulating the Fault Attack from Sections 3.3 and 3.4

We simulate the attack in two scenarios.

In *Scenario I*, we replace line 13 of Algorithm 1 with Algorithm 6. This function uses a list `L` of random $\mathbb{F}_{16}$ elements and a binary string `x` of size l^2v to determine which elements of $\mathtt{V}_i$ are fixed or randomly generated, ensuring consistent fixed values across executions.

Algorithm 6: Simulates a fault by fixing $\mathbb{F}_{16}$ elements in the vinegar variables.

```
Function assign_values_to_vinegar_variables_fault_F16($v, o, l, x, L$):
    V ← [];
    for $i \leftarrow 1$ to $v$ do
        $\mathtt{V}_i \leftarrow [0]^{l \times l}$;
        for $r_0 \leftarrow 1$ to $l$ do
            for $r_1 \leftarrow 1$ to $l$ do
                if $x_{i \cdot l^2 + r_0 \cdot l + r_1} = 1$ then
                    $\mathtt{V}_i[r_0, r_1] \leftarrow \mathtt{L}_{i \cdot l^2 + r_0 \cdot l + r_1}$;
                end
                else
                    $\mathtt{V}_i[r_0, r_1] \xleftarrow{\$} \mathbb{F}_{16}$;
                end
            end
        end
        V.append($\mathtt{V}_i$);
    end
    return $V$;
```

In *Scenario II*, we replace line 13 of Algorithm 1 with Algorithm 7. This function uses a binary string `x` of size $4l^2v$ and a list `L` of random $\mathbb{F}_2$ elements to ensure the same bits in the binary representation of each $\mathtt{V}_i$ remain fixed across executions.

Algorithm 7: Simulates a fault by fixing $\mathbb{F}_2$ elements in the vinegar variables.

```
Function assign_values_to_vinegar_variables_fault_F2($v, o, l, x, L$):
    V ← [];
    for $i \leftarrow 1$ to $v$ do
        $\mathtt{V}_i \leftarrow [0]^{l \times l}$;
        for $r_0 \leftarrow 1$ to $l$ do
            for $r_1 \leftarrow 1$ to $l$ do
                e ← $[0]^4$;
                for $r_2 \leftarrow 1$ to $4$ do
                    if $x_{i \cdot l^2 + r_0 \cdot l + 4 \cdot r_1 + r_2} = 1$ then
                        $\mathtt{e}[r_2] \leftarrow \mathtt{L}_{i \cdot l^2 + r_0 \cdot l + 4 \cdot r_1 + r_2}$;
                    end
                    else
                        $\mathtt{e}[r_2] \xleftarrow{\$} \mathbb{F}_2$;
                    end
                end
                $\mathtt{V}_i[r_0, r_1] \leftarrow \mathtt{e}$;
            end
        end
        V.append($\mathtt{V}_i$);
    end
    return $V$;
```

Our *test experiments* follow this procedure:

1. Select a SNOVA parameter set and generate a key pair ($\mathtt{sk}, \mathtt{pk}$) using the SNOVA key generation algorithm.
2. Create a bitstring $\mathtt{x}$ by performing l^2v (Scenario I) or $4l^2v$ (Scenario II) Bernoulli trials with probability $0 < \rho < 1$. Generate a list $\mathtt{L}$ of random field elements ($\mathbb{F}_{16}$ for Scenario I, $\mathbb{F}_2$ for Scenario II) of size $|\mathtt{x}|$.
3. Collect N_{msg} signatures using the modified Algorithm 1. Set $N_{msg} = o \cdot l + 2$ for Scenario I and $N_{msg} = 4 \cdot o \cdot l + 2$ for Scenario II.
4. Call the corresponding recovery algorithm: Algorithm 2 for Scenario I or Algorithm 3 for Scenario II.
5. Compare the recovered part of T with the actual T to verify the recovery algorithms' effectiveness.

Our *experimental results* confirm that the recovery algorithms perform as expected, successfully recovering the correct components of T with the required number of faulty signatures, as detailed in Sections 3.3 and 3.4. Table 3 summarizes the minimum number of faulty signatures needed for each SNOVA parameter set.

Table 3: Minimum number of signatures per SNOVA parameter set

Security Level	(v, o, q, l, λ)	Recovery from fixed field elements by Algorithm 2	Recovery from fixed bits by Algorithm 3
I	$(37, 17, 16, 2, 128)$	36	138
	$(25, 8, 16, 3, 128)$	26	98
	$(24, 5, 16, 4, 128)$	22	82
III	$(56, 25, 16, 2, 192)$	52	202
	$(49, 11, 16, 3, 192)$	35	134
	$(37, 8, 16, 4, 192)$	34	130
V	$(75, 33, 16, 2, 256)$	68	266
	$(66, 15, 16, 3, 256)$	47	182
	$(60, 10, 16, 4, 256)$	42	162

In addition to the SAGE implementation, we use the C version of SNOVA to generate faulty signatures by integrating Algorithm 6 into the code. Vinegar values are generated in the `sign_digest_core_ref` function (in `snova_kernel.h`) using Keccak from the XKCP library[4]. Listing 1.2 shows how these values are assigned to the matrix X_IN_GF16MATRIX.

To create faulty signatures, we modify the `get_F16` function, which generates random $\mathbb{F}_{16}$ elements and a binomial random variable array x. This array determines which entries in X_IN_GF16MATRIX are assigned random values instead of hash-derived values, as shown in Listing 1.3.

[4] https://github.com/XKCP/XKCP

Using the faulty X_IN_GF16MATRIX, we generate signatures by multiplying parts of it with the private key matrix 'T12', as detailed in Listing 1.4. The results from SAGE and the C code for generating faulty signatures and executing recovery algorithms are consistent.

4.2 Simulating the fault-assisted reconciliation attack

We first simulated our fault attack described in Section 3.6 using our SAGE implementation and the C version.

Let P be a matrix of size $v \times l \times l$, where the elements P_{ijk} represent the probabilities $p_{i,jk}$ as described in Section 3.6. We replace line 13 of Algorithm 1 with a function that takes a set of SNOVA parameters and the matrix P, randomly selects $\omega \in \mathbb{F}_q$ and returns the vinegar variables $\mathtt{V}_i$, where each $\mathtt{V}_{i,jk}$ is equal to ω with probability P_{ijk}.

We conducted experiments, each consisting of 100 runs of SNOVA and our algorithms. In each trial, probabilities for $P_{i,jk}$ are set, and the modified version of Algorithm 1 is run. A "failure" in Step 2 occurs if the modified algorithm cannot compute a signature after one iteration. A success in Algorithm 5 occurs if the secret space can be computed after Step 2 has completed successfully. Therefore, the success rate of Algorithm 5 is the number of successful computations divided by the number of trials, excluding those that failed in Step 2. Finally, the overall success rate of the attack strategy is the number of successes in Algorithm 5 divided by the total number of trials.

In our experiments we set $\epsilon \in \{1, 0.97, 0.95, 0.93\}$ and $\Gamma_{\epsilon,r} = \{\lfloor \mu + r\sigma \rfloor, \ldots, \lfloor \mu - r\sigma \rfloor\}$, where $\mu = lv\epsilon$, $\sigma = \sqrt{lv\epsilon(1-\epsilon)}$ and $r \in \{1, 2\}$. Table 4 shows our results for different assignment for P and the SNOVA parameter $(37, 17, 16, 2, 128)$. Algorithm 5's runtime was computed by using Eq. (13) and the Cryptographic Estimators library [10].

Implementing the fault-assisted reconciliation attack We used a ChipWhisperer-Lite (ARM-based) for clock-glitching attacks on the SNOVA signing process. The attack targets pseudo-random data generation to bypass a hash function call (line 5 of Listing 1.1 corresponding to line 9 of Listing 1.2) by skipping a key instruction with a clock glitch.

Across 200 trials, we observed that in 3 instances, the vinegar matrix $\mathtt{V}$ resulted in all its entries defaulting to zero or another fixed value. In 5 of those trials, it resulted in at least one column having a value repeated $v_l - 4$ times.

In our attack, we either forced the system to skip invoking the hash function entirely or induced it to omit critical instructions within the function, thereby fixing the output to a predetermined value. This aligns with our threat model and key recovery results in Table 4.

This mirrors the Keccak function-skipping method in [14], confirming that disrupting critical code segments enables key recovery. We posit that other fault injection techniques (e.g., voltage glitching) could similarly instantiate our threat model.

Table 4: Table with the results of our experiments for the SNOVA parameter $(37, 17, 16, 2, 128)$, where $C_\beta := \{(i, j, \beta) : i \in [v], j \in [l]\}$ for some $\beta \in [l]$.

Assignments for P	Failure Rate Step 2	Algorithm 5 Success Rate		Attack Strategy Success Rate		Algorithm 5 Runtime (bits)	
		$r = 1$	$r = 2$	$r = 1$	$r = 2$	$r = 1$	$r = 2$
$P_\iota = 1$ for $\iota \in [v] \times [l]^2$	6%	100%	100%	94%	94%	6	7
$0.97 \leq P_\iota \leq 1$ for $\iota \in [v] \times [l]^2$	6%	44%	100%	41%	94%	42	52
$0.95 \leq P_\iota \leq 1$ for $\iota \in [v] \times [l]^2$	7%	33%	100%	31%	93%	52	60
$0.93 \leq P_\iota \leq 1$ for $\iota \in [v] \times [l]^2$	5%	19%	100%	18%	95%	60	67
$P_\iota = 1$ for $\iota \in C_\beta$ $P_\iota = 1/q$ for $\iota \in [v] \times [l]^2 \setminus C_\beta$	2%	100%	100%	98%	98%	5	6
$0.97 \leq P_\iota \leq 1$ for $\iota \in C_\beta$ $P_\iota = 1/q$ for $\iota \in [v] \times [l]^2 \setminus C_\beta$	8%	41%	100%	38%	92%	41	51
$0.95 \leq P_\iota \leq 1$ for $\iota \in C_\beta$ $P_\iota = 1/q$ for $\iota \in [v] \times [l]^2 \setminus C_\beta$	5%	37%	100%	35%	95%	51	59
$0.93 \leq P_\iota \leq 1$ for $\iota \in C_\beta$ $P_\iota = 1/q$ for $\iota \in [v] \times [l]^2 \setminus C_\beta$	4%	40%	93%	38%	89%	59	66

```
8000416:     f000 fb23     bl     8000a60 <trigger_high>
800041a:     f107 0310     add.w     r3, r7, #16
800041e:     2100          movs     r1, #0
8000420:     4618          mov     r0, r3
8000422:     f002 fa45     bl     80028b0 <_etext+0x80>
8000426:     f107 01f0     add.w     r1, r7, #240     @ 0xf0
800042a:     f107 0310     add.w     r3, r7, #16
800042e:     f44f 62c0     mov.w     r2, #1536     @ 0x600
8000432:     4618          mov     r0, r3
8000434:     f002 fa2c     bl     8002890 <_etext+0x60>
8000438:     f000 fb19     bl     8000a6e <trigger_low>
```

Listing 1.1: Assembly code of the attack.

5 Countermeasure

The countermeasure adapts a general strategy designed to defend against fault attacks targeting multivariate public key cryptosystems. This strategy was initially proposed in [12] and later extended and tailored for the UOV and Rainbow schemes in [16].

Specifically, Algorithm 8 implements this countermeasure for SNOVA and should be invoked by Algorithm 1 immediately after executing line 13.

Algorithm 8 accepts three positive integers, $\mathit{\Gamma}$ and $\mathit{\Lambda}$, with the condition that $\mathit{\Gamma} < \mathit{\Lambda}$, and $\mathit{\Upsilon}$, as well as a tuple of finite field elements $(\alpha_1, \ldots, \alpha_{l^2 v})$ of size $l^2 v$. This tuple represents the SNOVA vinegar values $[\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v]$ generated at line 13 of Algorithm 1. Furthermore, the function `compare`, called by Algorithm 8 at line 14, takes two tuples of size $l^2 v$: $(\alpha_1, \ldots, \alpha_{l^2 v})$ and $(\beta_1, \ldots, \beta_{l^2 v})$. It returns a tuple of size $l^2 v$ where the j-th entry is 1 if $\alpha_j \neq \beta_j$ and 0 otherwise. Finally, the function `checkColumn` takes $(\alpha_1, \ldots, \alpha_{l^2 v}), x, \beta, \mathit{\Upsilon}$ and checks if there are at least $\mathit{\Upsilon}$ occurrences of x in the sequence $\mathtt{V}_{1,1\beta}, \ldots, \mathtt{V}_{1,l\beta}, \ldots, \mathtt{V}_{v,l\beta}$.

Algorithm 8: Countermeasure by checking and storing vinegar values

```
   Input: $\varGamma, \varLambda, \varUpsilon, (\alpha_1, \dots, \alpha_{l^2 v})$
   Output: success or fail
1  Function countermesure($\varGamma, \varLambda, \varUpsilon, (\alpha_1, \dots, \alpha_{l^2 v})$):
2      for $x \in \mathbb{F}_q$ do
3          for $\beta \in \{1, 2, \dots, l\}$ do
4              if checkColumn($(\alpha_1, \dots, \alpha_{l^2 v}), x, \beta, \varUpsilon$) then
5                  return fail;
6              end
7          end
8      end
9      if L has not been created then
10         L $\leftarrow$ [];
11     end
12     count $\leftarrow [0]^{vl^2}$;
13     for $i \leftarrow 0$ to $|L| - 1$ do
14         count $\leftarrow$ count + compare(L[$i$], $(\alpha_1, \dots, \alpha_{l^2 v})$);
15     end
16     if count[$j$] $> \varGamma$ for some $j \in [l^2 v]$ then
17         return fail ;
18     end
19     if $|L| = \varLambda$ then
20         L.removeEntryAtIndex(0);
21     end
22     L.append($(\alpha_1, \dots, \alpha_{l^2 v})$);
23     return success;
```

Why does this countermeasure work? Assume the countermeasure is implemented in the signing algorithm with $\varLambda = l \cdot o$ and $\varGamma < \varLambda$. The check between lines 2 and 7 targets the attack in Section 3.6. Let $\mathtt{V} = (\mathtt{V}_1^t, \dots, \mathtt{V}_v^t)^t$, and let Z_β count occurrences of $x \in \mathbb{F}_q$ in the β-th column of V. Z_β follows a binomial distribution with $p = 1/q$ in the absence of faults. We set $\varUpsilon$ such that $\Pr(Z_\beta \geq \varUpsilon)$, the probability of `checkColumn` returning `True` at line 4, is negligible. For SNOVA parameters, $\varUpsilon = \lfloor l \cdot v \cdot p + r\sqrt{l \cdot v \cdot p(1-p)} \rfloor$, where $r \geq 6$. If faults fix at least $\varUpsilon$ entries in a column to x, `checkColumn` returns `True`; otherwise, it returns `False`, making the attack runtime prohibitive (see Appendix C).

Next, we analyze the countermeasure against the attack in Section 3.3. Assume $|\mathtt{L}| = \varLambda$ and the `countermeasure` function is called with $\varGamma$, $\varLambda$, and $(\alpha_1, \dots, \alpha_{l^2 v})$. Let E be the event where line 17 of Algorithm 8 returns `fail` without fault injection. For E to occur, there must exist $j \in [l^2 v]$ such that `count`$[j] > \varGamma$, meaning α_j appears more than $\varGamma$ times in $(\mathtt{L}[i][j])_{i=0}^{\varLambda-1}$. Each X_j (counting α_j occurrences) follows a binomial distribution with $p_j = \frac{1}{|\mathbb{F}_q|}$, and the variables are independent. Thus, the probability of reaching line 17 is:

$$p_{\text{fail}} = 1 - \Pr(X_j \leq \varGamma \text{ for all } j \in [l^2 v]).$$

Under permanent fault injection, the signing algorithm aborts after $\varGamma$ faulty signatures. The attacker constructs $v^2 l$ under-determined linear systems (each with $\varGamma - 1$ equations and lo unknowns). Specializing $lo - \varGamma + 1$ variables and solving the system yields a correct solution with probability $p_j^{\varGamma - lo - 1}$, but the attacker cannot verify correctness. Thus, $\varGamma$ must ensure both p_{fail} and the attack success probability are negligible. Table 5 provides suitable Γ and Λ values for each SNOVA parameter set.

Table 5: Suggested values for Γ and Λ.

Security level	(v, o, q, l, λ)	Γ	Λ
I	$(37, 17, 16, 2, 128)$	10	34
	$(25, 8, 16, 3, 128)$	10	24
	$(24, 5, 16, 4, 128)$	6	20
III	$(56, 25, 16, 2, 192)$	14	50
	$(49, 11, 16, 3, 192)$	9	33
	$(37, 8, 16, 4, 192)$	8	32
V	$(75, 33, 16, 2, 256)$	15	66
	$(66, 15, 16, 3, 256)$	14	45
	$(60, 10, 16, 4, 256)$	9	40

6 Conclusion

In this paper, we presented multiple fault attack strategies against the SNOVA cryptographic scheme. We introduced two methods for executing fault attacks, demonstrating that our novel key recovery algorithm can recover the secret key with as few as 22 to 34 faulty signatures at the lowest security level, and up to 42 or 68 signatures at the highest level. Experiments implemented in SAGE and C confirmed the efficiency of our algorithm under various fault conditions. Additionally, we proposed a new fault-assisted reconciliation attack in Section 3.6, which exploits transient faults to recover the secret key space by solving a quadratic system. Evaluations using the lowest security parameter set for SNOVA showed a high success rate under specific fault probability conditions, highlighting the attack's potential to compromise SNOVA's security.

To address these vulnerabilities, we proposed a lightweight countermeasure that reduces the likelihood of successful key recovery without significantly affecting SNOVA's performance. This scalable solution is adaptable to various SNOVA parameter sets. Our findings emphasize the critical need for robust fault-resistant implementations in post-quantum cryptographic schemes like SNOVA. Future research could focus on optimizing countermeasures and investigating the impact of these attacks on other cryptographic systems.

References

1. Subidh Ali, Xiaofei Guo, Ramesh Karri, and Debdeep Mukhopadhyay. *Fault Attacks on AES and Their Countermeasures*, pages 163–208. Springer International Publishing, Cham, 2016.
2. Thomas Aulbach, Fabio Campos, and Juliane Krämer. SoK: On the physical security of UOV-based signature schemes. Cryptology ePrint Archive, Paper 2024/1818, 2024.
3. Thomas Aulbach, Tobias Kovats, Juliane Krämer, and Soundes Marzougui. Recovering rainbow's secret key with a first-order fault attack. In *AFRICACRYPT 2022*, pages 348–368, 2022.
4. Thomas Aulbach, Soundes Marzougui, Jean-Pierre Seifert, and Vincent Quentin Ulitzsch. Mayo or MAY-not: Exploring implementation security of the post-quantum signature scheme MAYO against physical attacks. In *2024 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 28–33, 2024.
5. Ward Beullens. MAYO: practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography - 28th International Conference, SAC 2021, Virtual Event, September 29 - October 1, 2021, Revised Selected Papers*, volume 13203 of *Lecture Notes in Computer Science*, pages 355–376. Springer, 2021.
6. Ward Beullens. Improved cryptanalysis of SNOVA. Cryptology ePrint Archive, Paper 2024/1297, 2024.
7. Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO, June 2023. Available at https://pqmayo.org/assets/specs/mayo.pdf.
8. Jonathan F Buss, Gudmund S Frandsen, and Jeffrey O Shallit. The computational complexity of some problems of linear algebra. *Journal of Computer and System Sciences*, 58(3):572–596, 1999.
9. Daniel Cabarcas, Peigen Li, Javier Verbel, and Ricardo Villanueva-Polanco. Improved attacks for SNOVA by exploiting stability under a group action. Cryptology ePrint Archive, Paper 2024/1770, 2024.
10. Andre Esser, Javier Verbel, Floyd Zweydinger, and Emanuele Bellini. Sok: CryptographicEstimators – a software library for cryptographic hardness estimation. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, page 560–574, New York, NY, USA, 2024. Association for Computing Machinery.
11. Hiroshi Furue, Yuichi Kiyomura, and Takashi Takagi. A new fault attack on UOV multivariate signature scheme. In *Post-Quantum Cryptography - PQCrypto 2022*, pages 124–143, 2022.
12. Yasufumi Hashimoto, Tsuyoshi Takagi, and Kouichi Sakurai. General fault attacks on multivariate public key cryptosystems. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 1–18, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
13. Yasuhiko Ikematsu and Rika Akiyama. Revisiting the security analysis of SNOVA. Cryptology ePrint Archive, Paper 2024/096, 2024. https://eprint.iacr.org/2024/096.
14. Sönke Jendral and Elena Dubrova. MAYO key recovery by fixing vinegar seeds. *IACR Communications in Cryptology*, 1(4), 2025.
15. Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. *SIGARCH Comput. Archit. News*, 42(3):361–372, June 2014.

16. Juliane Krämer and Mirjam Loiero. Fault attacks on UOV and Rainbow. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 193–214, Cham, 2019. Springer International Publishing.
17. Peigen Li and Jintai Ding. Cryptanalysis of the SNOVA signature scheme. Cryptology ePrint Archive, Paper 2024/110, 2024. https://eprint.iacr.org/2024/110.
18. Chun-Yen Chou Lih-Chung Wang, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. Snova: Proposal for nistpqc: Digital signature schemes project. Proposal for NISTPQC: Digital Signature Schemes project, 2023. https://snova.pqclab.org/.
19. Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Advances in Cryptology — EUROCRYPT '88*, pages 419–453, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.
20. Koksal Mus, Saad Islam, and Berk Sunar. Quantumhammer: A practical hybrid attack on the LUOV signature scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1071–1084, New York, NY, USA, 2020. Association for Computing Machinery.
21. Shuhei Nakamura, Yusuke Tani, and Hiroki Furue. Lifting approach against the SNOVA scheme. Cryptology ePrint Archive, Paper 2024/1374, 2024.
22. Jacques Patarin. Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. In *CRYPTO '95, 15th*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1995.
23. Jacques Patarin. The oil and vinegar algorithm for signatures. *Dagstuhl Workshop on Cryptography*, 1997. https://cir.nii.ac.jp/crid/1572543024892110208.
24. Kenneth G. Paterson, Jacob C. N. Schuldt, and Dale L. Sibborn. Related randomness attacks for public key encryption. Cryptology ePrint Archive, Paper 2014/337, 2014. https://eprint.iacr.org/2014/337.
25. Oussama Sayari, Soundes Marzougui, Thomas Aulbach, Juliane Krämer, and Jean-Pierre Seifert. Hamayo: A fault-tolerant reconfigurable hardware implementation of the MAYO signature scheme. In *Constructive Side-Channel Analysis and Secure Design*, pages 240–259, Cham, 2024. Springer Nature Switzerland.
26. Kyung-Ah Shim and Namhun Koo. Algebraic fault analysis of UOV and Rainbow with the leakage of random vinegar values. *IEEE Transactions on Information Forensics and Security*, 15:2429–2439, 2020.
27. Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Jan Adriaan Leegwater, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. A note on the SNOVA security. Cryptology ePrint Archive, Paper 2024/1517, 2024.
28. Lih-Chung Wang, Po-En Tseng, Yen-Liang Kuan, and Chun-Yen Chou. A simple noncommutative UOV scheme. Cryptology ePrint Archive, Paper 2022/1742, 2022.

A SNOVA algorithms

Algorithm 9 presents the signature verification process in the SNOVA cryptosystem. The algorithm uses the public key, document digest, and associated parameters to validate a signature. It begins by generating auxiliary parameters and random components required for evaluation. A hash value $\mathbf{hash}_s$ is computed from the public key, document digest, and salt, and is then compared against $\mathbf{hash}_d$, which is derived using Algorithm 12. The signature is accepted if the two hash values match; otherwise, it is rejected.

Algorithm 9: SNOVA Signature Verification

```
Input: SNOVA parameters $(v, o, l)$,
Public key $(\mathbf{s}_{\text{public}}, P_i^{22}$ for $0 \le i < m)$,
Document digest $\mathbf{digest} = \text{Hash}(D)$,
Digest length $|\mathbf{digest}|$
Output: Accept or Reject
1  Function VerifySignature()
2      Generate $A_\alpha$, $B_\alpha$, $Q_{\alpha 1}$, and $Q_{\alpha 2}$ for $0 \le \alpha < l^2$ using Algorithm 11;
3      $m \leftarrow o$;
4      Generate $(P_1^{11}, P_1^{12}, P_i^{21}$ for $0 \le i < m)$ using Algorithm 11;
5      $\mathbf{hash}_s \leftarrow \text{Hash}_{\text{SHAKE256}}(\mathbf{s}_{\text{public}} \,||\, \mathbf{digest} \,||\, \mathbf{salt})$;
6      Compute $\mathbf{hash}_d$ using Algorithm 12;
7      if $\mathbf{hash}_s == \mathbf{hash}_d$ then
8          return Accept;
9      else
10         return Reject;
```

Algorithm 10: Generate Invertible Matrix

```
Input: A $l \times l$ matrix $M$
Output: Invertible matrix $M$
1  if $\det(M) = 0$ then
2      for $a \leftarrow 1$ to 15 do
3          if $\det(M + aS) \neq 0$ then
4              $M \leftarrow M + aS$;
5              break;
6  return $M$;
```

Algorithm 11: Generate the random part of the public key

```
Input: SNOVA parameters $(v, o, l)$,
       public seed $\mathtt{s}_{\mathtt{public}}$
Output: Matrices $(A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2}$ for $0 \le \alpha < l^2)$,
        $(P_i^{11}, P_i^{12}, P_i^{21}, P_i^{22}$ for $0 \le i < m)$
1  Compute $\mathtt{Hash}_{\mathtt{AES128}}(\mathtt{s}_{\mathtt{public}})$ ;        // Initialize as AES128 throughout
2  for $\alpha$ from 0 to $l^2 - 1$ do
3      Let $A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2}$ be invertible using Algorithm 10;
4  return $(A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2}$ for $0 \le \alpha < l^2)$ and $(P_i^{11}, P_i^{12}, P_i^{21}, P_i^{22}$ for $0 \le i < m)$;
```

Algorithm 12: Evaluate the public map

```
Input: SNOVA parameters $(v, o, l)$,
       public key $(A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2}$ for $0 \le \alpha < l^2)$,
       public map $(P_i^{11}, P_i^{12}, P_i^{21}, P_i^{22}$ for $0 \le i < m)$,
       the signature sig
Output: The evaluation hashes of $P$ at sig
1  $m \leftarrow o$;
2  for $\alpha$ from 0 to $m-1$ do
3      for $j$ from 0 to $n-1$ do
4          $\texttt{Left}_\alpha[j] \leftarrow A_\alpha \cdot (\texttt{sig}[j])^t \cdot Q_{\alpha 1}$ ;        // The left term of $P_{i,d_\alpha,d_k}$
5          $\texttt{Right}_\alpha[j] \leftarrow Q_{\alpha 2} \cdot \texttt{sig}[j] \cdot B_\alpha$ ;        // The right term of $P_{i,d_\alpha,d_k}$
6  for $i$ from 0 to $m-1$ do
7      $\texttt{hash}_s[i] \leftarrow 0$;
8      for $\alpha$ from 0 to $l^2-1$ do
9          for $d_j$ from 0 to $v-1$ do
10             for $d_k$ from 0 to $v-1$ do
11                 $\texttt{hash}_s[i] \leftarrow \texttt{hash}_s[i] + \texttt{Left}_\alpha[d_j] \cdot P_i^{11}[d_j][d_k] \cdot \texttt{Right}_\alpha[d_k]$;
12     for $d_j$ from 0 to $v-1$ do
13         for $d_k$ from 0 to $v-1$ do
14             $\texttt{hash}_s[i] \leftarrow \texttt{hash}_s[i] + \texttt{Left}_\alpha[d_j] \cdot P_i^{12}[d_j][v+d_k] \cdot \texttt{Right}_\alpha[d_k]$;
15     for $d_j$ from 0 to $o-1$ do
16         for $d_k$ from 0 to $v-1$ do
17             $\texttt{hash}_s[i] \leftarrow \texttt{hash}_s[i] + \texttt{Left}_\alpha[v+d_j] \cdot P_i^{21}[d_j][d_k] \cdot \texttt{Right}_\alpha[d_k]$;
18     for $d_j$ from 0 to $o-1$ do
19         for $d_k$ from 0 to $o-1$ do
20             $\texttt{hash}_s[i] \leftarrow \texttt{hash}_s[i] + \texttt{Left}_\alpha[v+d_j] \cdot P_i^{22}[d_j][d_k] \cdot \texttt{Right}_\alpha[v+d_k]$;
21 $\texttt{hash}_s \leftarrow (\texttt{hash}_s[0], \ldots, \texttt{hash}_s[m-1])^t$;
22 return $\texttt{hash}_s$;
```

B Implementation details

```c
 // generate the vinegar value
        Keccak_HashInstance hashInstance;
        Keccak_HashInitialize_SHAKE256(&hashInstance);
        Keccak_HashUpdate(&hashInstance, pt_private_key_seed, 8 * seed_length_private);
        Keccak_HashUpdate(&hashInstance, digest, 8 * bytes_digest);
        Keccak_HashUpdate(&hashInstance, array_salt, 8 * bytes_salt);
        Keccak_HashUpdate(&hashInstance, &num_sign, 8);
        Keccak_HashFinal(&hashInstance, NULL);
        Keccak_HashSqueeze(&hashInstance, vinegar_in_byte, 8 * ((v_SNOVA * lsq_SNOVA + 1) >> 1));

        counter = 0;
        for (int index = 0; index < v_SNOVA; index++) {
            for (int i = 0; i < rank; ++i) {
                for (int j = 0; j < rank; ++j) {
                    set_gf16m(X_in_GF16Matrix[index], i, j,
                              ((counter & 1) ? (vinegar_in_byte[counter >> 1] >> 4) : (vinegar_in_byte[counter >> 1] & 0xF)));
                    counter++;
                }
            }
        }
```

Listing 1.2: Code snippet of generation of vinegar values.

```c
uint8_t x[v_SNOVA*lsq_SNOVA] = {0};
uint8_t I[v_SNOVA*lsq_SNOVA] = {0};
get_F16(v_SNOVA, o_SNOVA, I, x, 0.5);
for (int index = 0; index < v_SNOVA; index++) {
            for (int i = 0; i < rank; ++i) {
                for (int j = 0; j < rank; ++j) {
                    if (x[index * lsq_SNOVA + i * l_SNOVA + j] == 1) {
                        set_gf16m(X_in_GF16Matrix[index], i, j, I[index * lsq_SNOVA+ i * l_SNOVA + j]);
                    } else {
                        set_gf16m(X_in_GF16Matrix[index], i, j,
                                  ((counter & 1) ? (vinegar_in_byte[counter >> 1] >> 4) : (vinegar_in_byte[counter >> 1]
                                      & 0xF)));
                        counter++;
                    }
                }
            }
        }
```

Listing 1.3: Code snippet for setting random values as Algorithm 6.

```c
for (int index = 0; index < v_SNOVA; ++index) {
    gf16m_clone(signature_in_GF16Matrix[index], X_in_GF16Matrix[index]);
    for (int i = 0; i < o_SNOVA; ++i) {
        gf16m_mul(T12[index][i], X_in_GF16Matrix[v_SNOVA + i], gf16m_secret_temp0);
        gf16m_add(signature_in_GF16Matrix[index], gf16m_secret_temp0, signature_in_GF16Matrix[index]);
    }
}
for (int index = 0; index < o_SNOVA; ++index) {
    gf16m_clone(signature_in_GF16Matrix[v_SNOVA + index], X_in_GF16Matrix[v_SNOVA + index]);
}
```

Listing 1.4: Usage of X_IN_GF16MATRIX to generate the final signature.

C Agorithm 5's runtime complexity

Given a homogeneous multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^N \to \mathbb{F}_q^M$, we denote $\texttt{MQ}(N, M, q)$ the field multiplications required to find a nontrivial solution u satisfying $\mathcal{P}(u) = a \in \mathbb{F}_q^M$ if such solution exists. The runtime complexity of Algorithm 5 is bounded by

$$\mathcal{O}(q \sum_{\beta \in \mathcal{C}} \sum_{\gamma \in \varGamma_\beta} \binom{lv}{\gamma} \cdot \texttt{MQ}(lv - \gamma, ml^2, q)) \tag{13}$$

field multiplications.