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 is a post-quantum cryptographic signature scheme known for its efficiency and compact key sizes, making it a second-round candidate in the NIST post-quantum cryptography standardization process. This paper presents a comprehensive fault analysis of SNOVA, focusing on both permanent and transient faults during signature generation. We introduce several fault injection strategies that exploit SNOVA's structure to recover partial or complete secret keys with limited faulty signatures. Our analysis reveals that as few as 22 to 68 faulty signatures, depending on the security level, can suffice for key recovery. We propose a novel fault-assisted reconciliation attack, demonstrating its effectiveness in extracting the secret key space via solving a quadratic polynomial system. Simulations show transient faults in key signature generation steps can significantly compromise SNOVA's security. To address these vulnerabilities, we propose a lightweight countermeasure to reduce the success of fault attacks without adding significant overhead. Our results highlight the importance of fault-resistant mechanisms in post-quantum cryptographic schemes like SNOVA to ensure robustness.

Keywords: Physical attack · Fault-attack · SNOVA · MQ-based cryptography.

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

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf.

defined as follows: given a system of m quadratic equations in n variables over a finite field $\mathbb{F}_q$, find a solution $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_q^n$ such that:

$$\begin{cases} Q_1(x_1, x_2, \ldots, x_n) = 0 \\ Q_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ Q_m(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

where each Q_i is a quadratic polynomial of the form:

$$Q_i(x_1, x_2, \ldots, x_n) = \sum_{1 \leq j \leq k \leq n} a_{ijk} x_j x_k + \sum_{1 \leq j \leq n} b_{ij} x_j + c_i,$$

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. 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, in contrast, involve deliberately introducing errors during cryptographic execution, such as memory corruption or inducing bit flips, to extract sensitive information by analyzing erroneous outputs. To mitigate these threats, countermeasures such as constant-time algorithms, masking techniques, noise generation, and redundancy checks can be integrated, significantly enhancing the robustness of cryptographic implementations. These safeguards help ensure that the theoretical security of MQ-based schemes translates into practical resilience, providing strong protection against both side-channel and fault attacks in real-world environments.

1.1 MQ signature schemes

One of the earliest efforts to convert the MQ problem into a digital signature scheme was introduced in 1988 with the C^* scheme [20]. However, Patarin successfully attacked this scheme in 1995 [23], rendering it insecure. Since then, numerous advancements have been made in digital signature schemes based on multivariate polynomials. A prominent example of these advancements is the Unbalanced Oil-Vinegar (UOV) scheme, which offers strong security features.

The UOV-like schemes are an extension of the original Oil-Vinegar (OV) scheme and they are designed to enhance security by introducing an imbalance between the number of "oil" and "vinegar" variables. 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 public key consists of o quadratic polynomials P_i in $n = v + o$ variables over a finite field $\mathbb{F}_q$, defined as:

$$P_i(v_1, \ldots, v_v, o_1, \ldots, o_o) = \sum_{1 \le j \le k \le v} a_{ijk} v_j v_k + \sum_{j=1}^{v} \sum_{k=1}^{o} b_{ijk} v_j o_k + \sum_{1 \le j \le k \le o} c_{ijk} o_j o_k + \sum_{j=1}^{v} d_{ij} v_j + \sum_{k=1}^{o} e_{ik} o_k + f_i,$$

where $a_{ijk}, b_{ijk}, c_{ijk}, d_{ij}, e_{ik}, f_i \in \mathbb{F}_q$.

The private key consists of the secret lineal transformation and the so-called central map that link the vinegar and oil variables to the public key polynomials. During the signing process, random values are selected for the vinegar variables $v_1, \ldots, v_v$. These values are then substituted into the quadratic polynomials, resulting in a system of linear equations involving the oil variables. Next, the system is solved to determine the values of the oil variables $o_1, \ldots, o_o$, and the signature is produced by combining the values of both the vinegar and oil variables.

To verify the signature, the signed values are substituted into the public polynomials to ensure they satisfy the equations $P_i - y_i = 0$ for all i.

In the world of UOV-like schemes, we can remark that there are several benefits such as *small* signatures, fast verification, and reasonable public key sizes. This is the case of MAYO [4,6] and SNOVA [19,28].

1.2 Fault attacks

Fault attacks are techniques used to exploit cryptographic implementations. This section provides an overview of fault attacks.

Fault Attacks. Fault injection attacks intentionally disrupt a device's normal operation to induce errors and extract information from cryptographic processes. Techniques such as electromagnetic pulses, lasers, voltage glitches, and DRAM row hammering are commonly used for fault injection, each varying in precision and complexity [1,7,16]. For example, laser-based injections offer high accuracy but are expensive, whereas DRAM row hammering requires extensive profiling. Inducing faults often requires multiple attempts; however, this does not necessarily mean causing multiple distinct faults. It may involve repeatedly inducing a specific fault over several runs to collect sufficient data for exploitation.

Fault Attacks against other MQ-based cryptosystems Table 1 compares previous fault injection attacks on multivariate signature schemes, highlighting key

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

Algorithm	#Signatures	#Faults	Evaluation	Assumptions
Multiple [13]	Multiple	Multiple	Theoretical	None
UOV/Rainbow [17]	Multiple	Multiple	Theoretical	None
UOV [26]	44–103	Multiple	Theoretical	None
LUOV [21]	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 [2]	1	1	Practical	Zero-initialization
MAYO [15]	1	1	Practical, Simulation	None
SNOVA permanent fault strategy	22–68	1	Theoretical, Simulation	None
SNOVA Fault-assisted reconciliation attack	1	Multiple	Theoretical, Simulation	None

features such as the number of signatures and faults required, the evaluation method, and any assumptions made.

Recently, [15] 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.

In this work, we explore a similar attack. However, instead of targeting Keccak, we focus directly on the vinegar variables by inducing faults—i.e., 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 distinct from previous works.

1.3 Our contributions

In this paper, we explore fault injection attacks and their impact on the security of SNOVA. We present an analysis demonstrating that by inducing either a single permanent fault or transient faults in specific operations of the signature generation algorithm, an attacker can create faulty signatures that reveal partial private key information. Our comprehensive analysis covers several scenarios. First, we investigate which rows of T, the matrix representation of the SNOVA private linear map, an attacker can recover after fixing certain $\mathbb{F}_{16}$ elements in some (or all) vinegar variables during signature generation. We then generalize this by examining the recovery of rows of T after fixing bits in the binary representation of the vinegar variables. In both cases, we show that collecting enough faulty signatures allows an attacker to recover T partially, compromising the

private key. We then turn our attention to another scenario where, by inducing transient faults in the generation of vinegar variables, the attacker can recover the secret space via a reconciliation attack. Simulations of our fault attack are included to substantiate our claims.

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. This paper is organized as follows. Section 2 presents a self-contained description of the SNOVA signature scheme. Section 3 presents our fault analysis on SNOVA, introducing several scenarios along with the implications of our recovery algorithms. Section 4 describes our experimental setting, our simulation of our fault attacks on SNOVA, and its results. Section 5 presents a countermeasure to protect SNOVA against the fault attack introduced previously. Finally, Section 6 concludes our work by highlighting our findings and their implications and delineating future works.

2 A simple non-commutative UOV scheme

In this section, we introduce SNOVA, 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 $nl \times nl$ block diagonal matrix with Q-blocks along the diagonal.

The subring $\mathbb{F}_q[S]$ *of* $\mathcal{R}$ is defined to be

$$\mathbb{F}_q[S] = \{a_0 + a_1 S + \ldots + a_{l-1} S^{l-1} | a_0, a_1, \ldots, a_{l-1} \in \mathbb{F}_q\}$$

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 given by $\mathcal{F} = [\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m] : \mathcal{R}^n \to \mathcal{R}^m$, and $\mathcal{F}_i$ is defined as

$$\mathcal{F}_k(X_1, X_2, \ldots, X_n) = \sum_{\alpha=1}^{l^2} A_\alpha \cdot \Big(\sum_{(i,j)\in\Omega} X_i^t \cdot (Q_{\alpha 1} F_{k,ij} Q_{\alpha 2}) X_j \Big) \cdot B_\alpha$$

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 $k \in [m]$.

Invertible linear map for this scheme is the map $\mathcal{T} : \mathcal{R}^n \to \mathcal{R}^n$ 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 $\mathbb{F}_q[S]$. 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 as shown by Algorithm 1. It outputs a SNOVA key pair $(\mathtt{sk}, \mathtt{pk})$.

The public key `pk` is a representation of $\mathcal{P}$. A full public key consists of the list of matrices $\big\{P_k = \begin{bmatrix} P_k^{11} & P_k^{12} \\ P_k^{21} & P_k^{22} \end{bmatrix} : k \in [m]\big\}$ and the list of matrices $\big\{A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2} : \alpha \in [l^2]\big\}$. However, it is enough to store $(\mathtt{Spublic}, \{P_k^{22}\}_{k\in[m]})$. Indeed, the public seed `Spublic` 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 $\big\{F_k : k \in [m]\big\}$. In practice, a private seed

$\texttt{Sprivate}$ 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}, as shown by the lines from 8 to 11 of Algorithm 1.

Table 2 summarizes current SNOVA parameters, and public and private keys sizes, as well as, signature sizes for each security level.

Algorithm 1 Key generation algorithm

1: **function** $\texttt{KeyGen}(v, o, l, \lambda)$
2: $m \leftarrow o; n \leftarrow o + v$;
3: $\texttt{Sprivate} \xleftarrow{\$} \{0,1\}^{2\lambda}$
4: $\texttt{Spublic} \xleftarrow{\$} \{0,1\}^{2\lambda}$
5: $T^{12} \leftarrow \texttt{PRG}(\texttt{Sprivate})$, where $T^{12}_{ij} \in \mathbb{F}_{16}[S]$.
6: $\{P^{11}_k\}_{k\in[m]}, \{P^{12}_k\}_{k\in[m]}, \{P^{21}_k\}_{k\in[m]}, \{A_\alpha\}_{\alpha\in[l^2]}, \{B_\alpha\}_{\alpha\in[l^2]}, \{Q_{\alpha 1}\}_{\alpha\in[l^2]}, \{Q_{\alpha 2}\}_{\alpha\in[l^2]} \leftarrow \texttt{PRG}(\texttt{Spublic})$
7: **for** $(k \leftarrow 1, k \leq m, k \leftarrow k+1)$ **do**
8: $F^{11}_k \leftarrow P^{11}_k$
9: $F^{12}_k \leftarrow P^{11}_k T^{12} + P^{12}_k$
10: $F^{21}_k \leftarrow (T^{12})^t P^{11}_k + P^{21}_k$
11: $P^{22}_k \leftarrow (T^{12})^t \left(P^{11}_k T^{12} + P^{12}_k\right) + P^{21}_k T^{12}$
12: **end for**
13: $\texttt{sk} \leftarrow (\{F^{11}_k\}_{k\in[m]}, \{F^{12}_k\}_{k\in[m]}, \{F^{21}_k\}_{k\in[m]}, T^{12})$
14: $\texttt{pk} \leftarrow (\texttt{Spublic}, \{P^{22}_k\}_{k\in[m]})$
15: **return** $(\texttt{sk}, \texttt{pk})$;
16: **end function**

Table 2. Parameters for SNOVA [19].

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)
	$(49, 11, 16, 3, 192)$	5989.5(+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)
	$(66, 15, 16, 3, 256)$	15187.5(+16)	364.5(+16)	432297(+48)
	$(60, 10, 16, 4, 256)$	8000(+16)	560(+16)	389312(+48)

The $\texttt{Sign}$ function runs a UOV-like signing procedure as shown by Algorithm 2. It digitally signs a message $\texttt{M}$ under the private key $\texttt{sk}$. It first samples a $\texttt{salt}$ 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 $(\mathtt{S}, \mathtt{salt})$.

Algorithm 2 signs message $\mathtt{M}$

function $\mathtt{sign}(v, o, l, \lambda, \mathtt{sk}, spublic, \mathtt{M})$
$m \leftarrow o$
$n \leftarrow o + v$
$(\{F_k^{11}\}_{k\in[m]}, \{F_k^{12}\}_{k\in[m]}, \{F_k^{21}\}_{k\in[m]}, T^{12}) \leftarrow \mathtt{sk}$
$\{A_\alpha\}_{\alpha\in[l^2]}, \{B_\alpha\}_{\alpha\in[l^2]}, \{Q_{\alpha 1}\}_{\alpha\in[l^2]}, \{Q_{\alpha 2}\}_{\alpha\in[l^2]} \leftarrow PRG(\mathtt{spublic})$
$\mathtt{digest} \leftarrow \mathcal{H}_0(\mathtt{M})$
$\mathtt{salt} \xleftarrow{R} \{0,1\}^\lambda$
$\mathtt{is\_done} \leftarrow \mathtt{False}$
$cont \leftarrow 0$;
$[Y_1, Y_2, \ldots, Y_m] \leftarrow \mathcal{H}_1(\mathtt{Spublic}||\mathtt{digest}||\mathtt{salt})$
$F_k \leftarrow \sum_{\alpha=1}^{l^2} A_\alpha \big(\sum_{i=1}^{v}\sum_{j=1}^{v} X_i^t (Q_{\alpha 1} F_{k,ij}^{11} Q_{\alpha 2}) X_j + \sum_{i=1}^{v}\sum_{j=1}^{m} X_i^t (Q_{\alpha 1} F_{k,ij}^{12} Q_{\alpha 2}) X_j + \sum_{j=1}^{m}\sum_{i=1}^{v} X_j^t (Q_{\alpha 1} F_{k,ji}^{21} Q_{\alpha 2}) X_i \big) \cdot B_\alpha$
for all $i \in [m]$.
while $\mathtt{not\ is\_done}$ **do**
  $[\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v] \leftarrow PRG(Sprivate||\mathtt{digest}||\mathtt{salt}||cont)$
  Compute $F_{k,VV} \leftarrow \sum_{\alpha=1}^{l^2} A_\alpha \big(\sum_{i=1}^{v}\sum_{j=1}^{v} \mathtt{V}_i^t (Q_{\alpha 1} F_{k,ij}^{11} Q_{\alpha 2}) \mathtt{V}_j \big) \cdot B_\alpha$ for all $k \in [m]$.
  Express $Y_k - F_{k,VV} = \sum_{\alpha=1}^{l^2} A_\alpha \big(\sum_{i=1}^{v}\sum_{j=1}^{m} \mathtt{V}_i^t (Q_{\alpha 1} F_{k,ij}^{12} Q_{\alpha 2}) X_j + \sum_{j=1}^{m}\sum_{i=1}^{v} X_j^t (Q_{\alpha 1} F_{k,ji}^{21} Q_{\alpha 2}) \mathtt{V}_i \big) \cdot B_\alpha$ for all $k \in [m]$ as an equation system on the oil variables $\overrightarrow{X_1}, \overrightarrow{X_2}, \ldots, \overrightarrow{X_m}$.

$$\begin{aligned} M_{1,1}\overrightarrow{X_1} + M_{1,2}\overrightarrow{X_2} + \ldots + M_{1,m}\overrightarrow{X_m} &= \overrightarrow{Y_1} - \overrightarrow{F_{1,VV}} \\ M_{2,1}\overrightarrow{X_1} + M_{2,2}\overrightarrow{X_2} + \ldots + M_{2,m}\overrightarrow{X_m} &= \overrightarrow{Y_2} - \overrightarrow{F_{2,VV}} \\ &\vdots \\ M_{m,1}\overrightarrow{X_1} + M_{m,2}\overrightarrow{X_2} + \ldots + M_{m,m}\overrightarrow{X_m} &= \overrightarrow{Y_m} - \overrightarrow{F_{m,VV}} \end{aligned}$$

  Represent this equation system as an $(ml^2) \times (ml^2 + 1)$ matrix $\mathbf{A}$ over $\mathbb{F}_{16}$
  $L_O, \mathtt{output} \leftarrow \mathtt{Gauss}(\mathbf{A})$
  if $\mathtt{output}$ **then**
    $[\mathtt{V}_{v+1}, \mathtt{V}_{v+2}, \ldots, \mathtt{V}_n] \leftarrow L_O$
    $\mathtt{V} \leftarrow [\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v, \mathtt{V}_{v+1}, \ldots, \mathtt{V}_n]$
    $T \leftarrow \begin{bmatrix} I^{11} & T^{12} \\ O & I^{22} \end{bmatrix}$
    $\mathtt{S} \leftarrow T \cdot \mathtt{V}^t$
    $\mathtt{is\_done} \leftarrow \mathtt{True}$
  else
    $cont \leftarrow cont + 1$
  end if
end while
return $(\mathtt{S}, \mathtt{salt})$;
end function

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

Reconciliation attack. Ikematsu, and Akiyama [14], Li and Ding [18], and Nakamura, Tani, and Furue [22] 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, [14] and [18] 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 $k \in [m], i, j \in \{0, 1, \ldots, l-1\}$.

Once $\mathbf{u}_0$ is found, any $\mathbf{u}$ in the linearly independent set $\{\Lambda_{S^j}\mathbf{u}_0 : 0 \leq j \leq l-1\}$ 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

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

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 3.7. This attack leverages induced transient faults to recover the secret key space by solving the system as mentioned earlier.

3 Fault analysis on SNOVA

In this section, we present our comprehensive fault analysis on SNOVA. We first state our attack model, then explore a first unsuccessful attempt at mounting a fault attack, and then turn our attention to our successful attempts at mounting a fault attack against SNOVA.

3.1 Adversarial Model

We consider an adversarial model similar to those assumed in previous works, particularly in [17], within the context of UOV and RAINBOW.

In this model, the attacker targets the signature generation process and can induce transient or permanent faults affecting specific operations within Algorithm 2. These faults enable the attacker to manipulate certain values during the execution of the signature generation algorithm. Importantly, the attacker may be unaware of the exact number of manipulated values or their specific content.

Subsequently, the attacker can invoke the faulty Algorithm 2 multiple times –where the number of invocations depends on the attack strategy– to gather message-signature pairs. Each signature is generated with the tampered values, and the attacker's ultimate goal is to analyze these pairs to extract partial information about the private key.

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 2, 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 2 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

$$\mathcal{P}_k(\mathtt{S}^{(\omega)}) = \sum_{\alpha=1}^{l^2} A_\alpha (T\mathtt{S}^{(\omega)})^t \Lambda_{Q_{\alpha 1}} F_k \Lambda_{Q_{\alpha 2}} (T\mathtt{S}^{(\omega)}) \cdot B_\alpha$$

for any $k \in [m]$. Therefore, it holds

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

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 22 of Algorithm 2.

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.

We remark, nonetheless, 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 following scenario focuses on inducing a permanent fault affecting the line 13 of Algorithm 2 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 2, 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.
2. For each $\omega \in [N_{msg}]$, the attacker calls Algorithm 2 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 3 with parameters $v, o, l, [\mathtt{S}^{(1)}, \ldots, \mathtt{S}^{(N_{msg})}]$ to obtain $\mathtt{dic}$, a dictionary-like data structure indexed by $[v] \times [l]^2$. When $N_{msg} > lo + 1$, Algorithm 3 will output $\mathtt{dic}$ such that $\mathtt{dic}[(i, r_0, r_1)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I$ and $(r_0, r_1) \in J_i$, and $\mathtt{dic}[(i, r_0, r_1)] = \mathtt{None}$ otherwise.

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 22 of Algorithm 2, it holds $T\mathtt{S}^{(\omega)} = \mathtt{V}^{(\omega)}$, that is,

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

Since $T^{12}_{ij} \in \mathbb{F}_q[S]$, it holds $T^{1,2}_{ij} = \sum_{j_1=1}^{l} t^{12}_{ij,j_1} S^{j_1-1}$, where $t^{12}_{ij,j_1} \in \mathbb{F}_q$, and

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

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

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

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

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

Let us fix $2 \leq \omega \leq N_{msg}$ 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,

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

Note that 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 $2 \leq \omega \leq N_{msg}$, a linear system of $(N_{msg}-1)|J_i|$ equations and lo unknowns over $\mathbb{F}_q$ can be obtained and is given by

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

However the attacker does not know either I or J_i. If J_i were known by the attacker, 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).

In order for the attacker to gain knowledge of I and J_i for $i \in I$, and recover $T_{i,j}^{12}$ for $i \in I, j \in [o]$, the attacker may try to solve $v \cdot l^2$ linear systems separately, i.e. one for $i \in [v]$ and $(r_0, r_1) \in [l]^2$,

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

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 3 will output `dic` such that $\mathtt{dic}[(i, r_0, r_1)] = [T_{i1}^{12}, \ldots, T_{io}^{12}]$ for $i \in I, (r_0, r_1) \in J_i$ and $\mathtt{dic}[(i, r_0, r_1)] = \mathtt{None}$ 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 3 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 $\mathtt{V}_i$ 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.

Algorithm 3 partially recovers T^{12} from fixed field elements.

function recover_partial_T_F16$(v, o, l, [\mathtt{S}^{(1)}, \ldots, \mathtt{S}^{(N_{msg})}])$
$S \leftarrow \mathtt{getS}(l)$
$\mathtt{dic} \leftarrow \{\}$
for $(i \leftarrow 1, i \leq v, i \leftarrow i+1)$ **do**
  for $(r_0 \leftarrow 1, r_0 \leq l, r_0 \leftarrow r_0+1)$ **do**
    for $(r_1 \leftarrow 1, r_1 \leq l, r_1 \leftarrow r_1+1)$ **do**
      $\mathbf{A} \leftarrow \mathbb{F}_{16}^{(N_{msg}-1)\times ol}$
      $\mathbf{Y} \leftarrow \mathbb{F}_{16}^{(N_{msg}-1)\times 1}$
      for $(\omega \leftarrow 2, \omega \leq N_{msg}, \omega \leftarrow \omega+1)$ **do**
        $\mathtt{S}_i^{(\omega,1)} \leftarrow \mathtt{S}_i^{(1)} - \mathtt{S}_i^{(\omega)}$
        for $(j \leftarrow 1, j \leq o, j \leftarrow j+1)$ **do**
          for $(j_1 \leftarrow 1, j_1 \leq l, j_1 \leftarrow j_1+1)$ **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 for
        end for
        $\mathbf{Y}_{(\omega-1),0} \leftarrow \mathtt{S}_{i,(r_0,r_1)}^{(\omega,1)}$
      end for
      $\mathtt{X}, \mathtt{output} \leftarrow \mathtt{Gauss}(\mathbf{A}, \mathbf{Y})$
      if output **then**
        $[T_{i1}^{12}, \ldots, T_{io}^{12}] \leftarrow \mathtt{get\_elements\_in\_FqS}(\mathtt{X}, l)$
        $\mathtt{dic}[(i, r_0, r_1)] \leftarrow [T_{i1}^{12}, \ldots, T_{io}^{12}]$
      else
        $\mathtt{dic}[(i, r_0, r_1)] \leftarrow \mathtt{None}$
      end if
    end for
  end for
end for
return dic
end function

1. The attacker causes a single permanent fault, which affects the line 13 of Algorithm 2, such that some bits of $\mathtt{V}_i \in \mathcal{R}$, with $i \in I \subseteq [v]$ and $|I| \geq 1$, are fixed and unknown. In particular, for the variable $\mathtt{V}_i$, there is a fixed non-empty subset $B_i \subseteq [l] \times [l] \times [N_{bits}]$, such that $\bar{\mathtt{V}}_{i,(r_0,r_1,b)} \in \mathbb{F}_2$, $(r_0, r_1, b) \in B_i$ is fixed and unknown.
2. For each $\omega \in [N_{msg}]$, the attacker calls Algorithm 2 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 calls Algorithm 4 with parameters $v, o, l, [\mathtt{S}^{(1)}, \ldots, \mathtt{S}^{(N_{msg})}]$ to get $\mathtt{dic}$, a dictionary-like data structure indexed by $[v] \times [l]^2 \times [N_{bits}]$. When $N_{msg} > N_{bits} \cdot lo + 1$, Algorithm 4 will output $\mathtt{dic}$ such that $\mathtt{dic}[(i, r_0, r_1, b)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I, (r_0, r_1, b) \in B_i$ and $\mathtt{dic}[(i, r_0, r_1, b)] = \mathtt{None}$ 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

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

where $\kappa^{i,j,j_1,\omega,1}_{(r_0,r_1)} = t^{12}_{ij,j_1} \mathtt{S}^{(j_1,\omega,1)}_{v+j,(r_0,r_1)}$. 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 $(r_0, r_1, b) \in [l] \times [l] \times [N_{bits}]$,

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

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 4 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 4 will output $\mathtt{dic}$ such that $\mathtt{dic}[(i, r_0, r_1, b)] = [T^{12}_{i1}, \ldots, T^{12}_{io}]$ for $i \in I, (r_0, r_1, b) \in B_i$ and $\mathtt{dic}[(i, r_0, r_1, b)] = \mathtt{None}$ otherwise.

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

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

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

Note that the previous equations can always be arranged as

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

$$\text{with } \mathbf{S}_i = \begin{bmatrix} \mathtt{S}_i^{(1)} \\ \vdots \\ \mathtt{S}_i^{(N_{msg})} \end{bmatrix}^t, \mathbf{S}_{j_1,j} = \begin{bmatrix} S^{j_1-1}\mathtt{S}_{v+j}^{(1)} \\ \vdots \\ S^{j_1-1}\mathtt{S}_{v+j}^{(N_{msg})} \end{bmatrix}^t, \mathbf{V}_i = \begin{bmatrix} \mathtt{V}_i^{(1)} \\ \vdots \\ \mathtt{V}_i^{(N_{msg})} \end{bmatrix}^t \in \mathcal{R}^{1\times N_{msg}}.$$

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

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

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

3.6 Discussion of previous scenarios

We remark that the scenarios described in Sections 3.3 and 3.4 are particular cases of related randomness attacks [24] because what the attacker accomplishes by injecting a permanent fault is force the signature algorithm to reuse the same sub-bitstrings within the $N_{bits}vl^2$ bitstring that represent the vinegar values $\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v$, even though these fixed sub-bitstrings are unknown to the attacker each time a valid signature is generated for a random message. Therefore, partially recovering the private linear map represented by T is always possible by using the techniques presented in Sections 3.3 and 3.4, as long as the attacker finds any other means of fixing sub-bitstrings within the $N_{bits}vl^2$ bitstring and collects enough valid signatures generated using these fixed values.

Moreover, if the attacker might distinguish a bit at a fixed position of each $\mathtt{V}_i$ during signature generation, then the attacker could leverage Algorithm 4 to recover T after collecting a sufficient number of signatures. Indeed, define $\mathcal{J} := [v] \times [l]^2 \times [N_{bits}]$, $\mathcal{G}$ and $\mathcal{O}_{\mathcal{I}}$ as shown by Algorithm 5. Suppose that the attacker is given access to the set $\mathcal{I} \leftarrow \mathcal{G}()$ and the oracle $\mathcal{O}_{\mathcal{I}}$.

This adversary can leverage his knowledge of $\mathcal{I}$, his access to $\mathcal{O}_{\mathcal{I}}$ and Algorithm 4 to fully recover the private linear transformation $\mathcal{T}$ as follows.

Algorithm 4 Partially Recovers T^{12} from Fixed Bits

function `recover_partial_T_F2`$(v, o, l, [\mathtt{S}^{(1)}, \ldots, \mathtt{S}^{(N_{msg})}])$
$S \leftarrow \mathtt{getS}(l)$
$\mathtt{dic} \leftarrow \{\}$
for $i \leftarrow 1$ **to** v **do**
  for $r_0 \leftarrow 1$ **to** l **do**
    for $r_1 \leftarrow 1$ **to** l **do**
      for $b \leftarrow 1$ **to** 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 $j \leftarrow 1$ **to** o **do**
            for $j_1 \leftarrow 1$ **to** 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
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+1} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+2} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,3)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+3} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,2)}^{(j_1,k,1)}$
                $\mathbf{A}_{(\omega-1),j\cdot l+4\cdot j_1+4} \leftarrow \mathtt{S}_{v+j,(r_0,r_1,4)}^{(j_1,k,1)} + \mathtt{S}_{v+j,(r_0,r_1,1)}^{(j_1,k,1)}$
              end if
            end for
          end for
          $\mathbf{Y}_{(\omega-1),0} \leftarrow \mathtt{S}_{i,(r_0,r_1,b)}^{(\omega,1)}$
        end for
        $\mathtt{X}, \mathtt{output} \leftarrow \mathtt{Gauss}(\mathbf{A}, \mathbf{Y})$
        if `output` **then**
          $[T_{i1}^{12}, \ldots, T_{io}^{12}] \leftarrow \mathtt{get\_elements\_in\_FqS\_from\_bits}(\mathtt{X}, l)$
          $\mathtt{dic}[(i, r_0, r_1, b)] \leftarrow [T_{i1}^{12}, \ldots, T_{io}^{12}]$
        else
          $\mathtt{dic}[(i, r_0, r_1, b)] \leftarrow \mathtt{None}$
        end if
      end for
    end for
  end for
end for
return `dic`
end function

Algorithm 5 defines functions $\mathcal{G}$ and $\mathcal{O}_{\mathcal{I}}$.

1: **function** $\mathcal{G}()$
2: $\mathcal{I} \leftarrow \emptyset$
3: **for** $(i \leftarrow 1, i \leq v, i \leftarrow i+1)$ **do**
4: $\quad (r_0, r_1, b) \xleftarrow{\$} [l] \times [l] \times [N_{bits}]$
5: $\quad \mathcal{I} \leftarrow \mathcal{I} \cup \{(i, r_0, r_1, b)\}$
6: **end for**
7: **return** $\mathcal{I}$
8: **end function**

1: **function** $\mathcal{O}_{\mathcal{I}}(\iota \in \mathcal{J}, b \in \mathbb{F}_2)$
2: **if** $\iota \in \mathcal{J} \setminus \mathcal{I}$ **then**
3: $\quad c \xleftarrow{\$} \mathbb{F}_2$
4: $\quad$ **return** c
5: **end if**
6: Let $\mathtt{V}_\iota$ be the random bit chosen by the line 13 of Algorithm 2 in the most recent call.
7: **if** $b = \mathtt{V}_\iota$ **then**
8: $\quad$ **return** 1
9: **end if**
10: **return** 0
11: **end function**

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 2 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 4 $|\mathcal{I}|$ 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 4 with parameters v, o, l and $\mathcal{S}_\iota$. From Section 3.4, it follows each call of Algorithm 4 with parameters v, o, l and $\mathcal{S}_\iota$ will return $\mathtt{dic}[\iota] = [T_{i1}^{12}, \ldots, T_{io}^{12}]$.

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.

3.7 Fault-assisted reconciliation attack

As seen in Section 2, 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

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

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

$$\mathbf{u}_0 = \begin{bmatrix} \mathtt{S}_{1,:\beta} - \mathtt{V}_{1,:\beta} \\ \vdots \\ \mathtt{S}_{v,:\beta} - \mathtt{V}_{v,:\beta} \\ \vdots \\ \mathtt{S}_{n,:\beta} \end{bmatrix},$$

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 $\beta \in [l]$.

Attack strategy

1. The attacker introduces transient faults affecting line 13 of Algorithm 2, specifically targeting $\mathtt{V}_{i,j\beta}$ for all $i \in [v], j \in [l]$ and $\beta \in \mathcal{C} \subseteq [l]$ independently. In particular, for the variable $\mathtt{V}_i$, 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 some unknown and fixed value.
2. The attacker then calls Algorithm 2 with a randomly chosen message $\mathtt{M} \in \mathcal{R}^m$ and receives the signature $(\mathtt{S}, \mathtt{salt})$.
3. If Step 2 succeeds, then call Algorithm 6 with parameters $v, o, l, S, \mathcal{C}, \Gamma_\beta$ for $\beta \in \mathcal{C}$[1], where $\Gamma_\beta \subseteq [lv]$. Furthermore, F_γ denotes the set of all subsets of γ integers that can be selected from $[lv]$ and $A^c = [lv] \setminus A$ with $A \in F_\gamma$.

We remark that by choosing proper Γ_β's, the attacker can ensure that the quadratic systems to be solved at the line 9 of Algorithm 6 have ol^2 equations and $lv - \gamma < ol^2$ unknowns. Thus, they are expected to have either no solution or very few solutions.

Let $\mathtt{V} = (\mathtt{V}_1^t, \ldots, \mathtt{V}_v^t)^t \in \mathbb{F}_q^{vl^2}$. If after carrying out Steps 1 and 2, there exists $\beta \in \mathcal{C}$ such that $\mathtt{V}_{i\beta} = \omega$, for $i \in A$, with $A \in F_\gamma$ and $\gamma \in \Gamma_\beta$, then Algorithm 6 will find an $\mathtt{U}$ satisfying Eq. (12) and an secret space $\mathcal{O}$. Otherwise, the attacker may start the attack again. In Appendix B, we analyse Algorithm 6's runtime complexity.

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

Algorithm 6 attempts to find $\mathcal{O}$ after having run the attack strategy.

function `fault_assisted_reconcilation_attack`$(v, o, l, \mathtt{S}, \mathcal{C}, \Gamma_\beta$ for $\beta \in \mathcal{C})$
for $\beta \in \mathcal{C}$ **do**
  for $\gamma \in \Gamma_\beta$ **do**
    for $A \in F_\gamma$ **do**
      for $\omega \in \mathbb{F}_q$ **do**
        Set $\Omega \leftarrow (x_1, \ldots, x_{lv}, 0, \ldots, 0)^t \in \mathbb{F}_q^{ln}$
        Set $\Omega_i \leftarrow w$ for $i \in A$
        $\mathtt{X} \leftarrow \mathtt{S}_{:\beta} - \Omega$
        Attempt to solve the quadratic system

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

        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 $i \in A^c$
        if x_i for $i \in A^c$ are found **then**
          Set $\mathtt{U}$ as the solution.
          Recover $\mathcal{O}$ from the linearly independent set $\{\Lambda_{S^j}\mathtt{U} : 0 \leq j \leq l-1\}$
          return $\mathcal{O}$
        end if
      end for
    end for
  end for
end for
return $\perp$

Success Probability of our Attack Strategy. For $i \in [vl], j \in \mathcal{C}$, let $X_{ij} \in \{0, 1\}$ be a Bernoulli random variable that indicates whether $\mathtt{V}_{ij}$ is fixed to ω due to the corresponding transient fault. Let $Pr(X_{ij}) = p_{ij}$ be the probability that $X_{ij} = 1$, i.e., that $\mathtt{V}_{ij}$ is fixed to ω. Define $Y_\beta = \sum_{i=1}^{lv} X_{i\beta}$. The probability of obtaining $0 \leq \gamma \leq lv$ successful fixes out of a total of lv in the β-th column of $\mathtt{V}$ can be expressed as

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

Let ρ_β be the probability there exists $\gamma \in \Gamma_\beta$ such that $Y_\beta = \gamma$. Therefore,

$$\rho_\beta = \sum_{\gamma \in \Gamma_\beta} \Pr(Y_\beta = \gamma).$$

Let ρ denote *the success probability of Algorithm 6*, meaning there exists $\beta \in \mathcal{C}$ such that $\mathtt{V}_{i\beta} = \omega$, for $i \in A$, with $A \in F_\gamma$ and $\gamma \in \Gamma_\beta$. Therefore,

$$\rho = \Pr(Y_\beta = \gamma \text{ for some } \beta \in \mathcal{C} \text{ with } A \in F_\gamma \text{ and } \gamma \in \Gamma_\beta) = \max\{p_\beta : \beta \in \mathcal{C}\}.$$

Overall, a run of the attack strategy is successful with probability $\rho(1 - \delta)$, where δ is the failure probability of Step 2 of the attack strategy (i.e., when Algorithm 2 fails to return a signature after one iteration). Furthermore, if the

p_{ij}'s remain constant for each run of the attack strategy, the attacker is expected to execute the attack strategy $1/\rho(1-\delta)$ times.

We note that Step 1 of this attack strategy can potentially be implemented by inducing a single transient fault, similar to the one found in the C implementation of Keccak used to generate the vinegar variables for MAYO [15]. For SNOVA, such a transient fault would target $\mathtt{V}_{i\beta}$ for $i \in [lv], \beta \in [l]$. Therefore, the attacker would need to run the attack strategy $1/(1-\delta)$ times if $p_{i\beta} = 1$ for all i and β. However, if the attacker only can ensure $\epsilon \leq p_{i\beta} \leq 1$ with $\epsilon \geq 0$, they can choose some Γ_ϵ for all $\beta \in [l]$, and hence

$$(1-\delta)\sum_{\gamma\in\Gamma_\epsilon}\binom{lv}{\gamma}\epsilon^\gamma(1-\epsilon)^{lv-\gamma} \leq (1-\delta)\max\{\rho_\beta = \sum_{\gamma\in\Gamma_\epsilon}\Pr(Y_\beta=\gamma) : \beta\in\mathcal{C}\} \leq (1-\delta).$$

Moreover, the attacker might improve the attack strategy's success probability by inducing transient faults that specifically target $\mathtt{V}_{i\beta}$ for all $i \in [lv]$, and $\beta \in \mathcal{C}$ with $|\mathcal{C}| = 1$. In such a case, δ is expected to be very low. Therefore, if the attacker only can ensure $\epsilon \leq p_{i\beta} \leq 1$ for a $\epsilon \geq 0$, they might set a proper Γ_ϵ such that $\sum_{\gamma\in\Gamma_\beta}\binom{lv}{\gamma}\epsilon^\gamma(1-\epsilon)^{lv-\gamma} \approx 1$, and expect to run the attack strategy[2] only once. We analyze various scenarios in our simulations in Section 4.2.

3.8 Alternative Versions of SNOVA

The SNOVA team recently released a preprint [27] that proposes two new versions of SNOVA to counteract Buellen's attack [5].

The first alternative version of SNOVA is choose random matrices $A_{k,\alpha}, B_{k,\alpha} \in \mathcal{R}$ and $Q_{k,\alpha1}, Q_{k,\alpha2} \in \mathbb{F}_q[S]$, for $k \in [o]$ and $\alpha \in [l^2]$, and define the k-th coordinate of the public map $\mathcal{P}(U)$ as

$$\mathcal{P}_k(U_1,\ldots,U_n) = \sum_{\alpha=1}^{l^2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{k,\alpha}\cdot U_i^t(Q_{k,\alpha1}P_{k,i,j}Q_{k,\alpha2})U_j\cdot B_{k,\alpha}.$$

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

$$\mathcal{P}_k(U) = \sum_{\alpha=1}^{l^4}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{\alpha}\cdot U_i^t(Q_{\alpha1}P_{k,i,j}Q_{\alpha2})U_j\cdot B_{\alpha},$$

where the matrices $A_\alpha, B_\alpha \in \mathcal{R}$, and $Q_{\alpha1}, Q_{\alpha2} \in \mathbb{F}_q[S]$, 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 is *still vulnerable to our fault attacks* described in Sections 3.3 and 3.4 since we 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.

[2] The runtime of Algorithm 6 depends on Γ_ϵ.

4 Experiments of our fault attacks

To validate our claims, we conducted simulations of our fault attack and presented a step-by-step procedure for our tests along with their results. We begin by explaining the conceptual framework of the attack. We implement the SNOVA signature scheme in SAGE, adhering to its specification document [19]. This implementation serves as the foundation for our analysis. Additionally, we utilized the latest version of the SNOVA code provided by the SNOVA team[3] to generate the signatures. In these simulations, we introduced faults by fixing specific values in the vinegar variables, thereby mimicking the fault injection process described in our attack model in Section 3.1.

4.1 Evaluating the fault attack from Sections 3.3 and 3.4

We evaluate our attack by considering two scenarios.

In *Scenario I*, we replace line 13 of Algorithm 2 with the function described in Algorithm 7. This function takes a set of SNOVA parameters, a binary string $\mathtt{x}$ of size l^2v, and a list $\mathtt{L}$ of the same size containing random field elements from $\mathbb{F}_{16}$. The binary string $\mathtt{x}$ directs Algorithm 7 regarding which elements are to be drawn from $\mathtt{L}$ and which are to be generated randomly. This approach ensures that each time the signature algorithm is executed, the same field elements in each $\mathtt{V}_i$ remain fixed.

Algorithm 7 simulates a fault by fixing $\mathbb{F}_{16}$ elements in the vinegar variables.

1: **function** `assign_values_to_vinegar_variables_fault_F16`$(v, o, l, \mathtt{x}, \mathtt{L})$
2: $\mathtt{V} \leftarrow []$
3: **for** $(i \leftarrow 1, i \leq v, i \leftarrow i+1)$ **do**
4: $\quad \mathtt{V}_i \leftarrow [0]^{l \times l}$
5: $\quad$ **for** $(r_0 \leftarrow 1, r_0 \leq l, r_0 \leftarrow r_0 + 1)$ **do**
6: $\quad\quad$ **for** $(r_1 \leftarrow 1, r_1 \leq l, r_1 \leftarrow r_1 + 1)$ **do**
7: $\quad\quad\quad$ **if** $(\mathtt{x}_{i \cdot l^2 + r_0 \cdot l + r_1} = 1)$ **then**
8: $\quad\quad\quad\quad \mathtt{V}_i[r_0, r_1] \leftarrow \mathtt{L}_{i \cdot l^2 + r_0 \cdot l + r_1}$
9: $\quad\quad\quad$ **else**
10: $\quad\quad\quad\quad \mathtt{V}_i[r_0, r_1] \xleftarrow{\$} \mathbb{F}_{16}$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad \mathtt{V}.\mathtt{append}(\mathtt{V}_i)$
15: **end for**
16: **return** $\mathtt{V}$

In *Scenario II*, we replace line 13 of Algorithm 2 with the function represented by Algorithm 8. This function takes a SNOVA parameter set, a binary string $\mathtt{x}$

[3] https://github.com/PQCLAB-SNOVA/SNOVA using commit 3d7e8c7cebdd57293d74dc6c2608656697b99597.

of size $4l^2v$, and a list $\mathtt{L}$ of size $4l^2v$ containing random field elements from $\mathbb{F}_2$. Similar to the previous scenario, Algorithm 8 guarantees that the same bits in the binary representation of each $\mathtt{V}_i$ are fixed each time Algorithm 2 is executed.

Algorithm 8 simulates a fault by fixing $\mathbb{F}_2$ elements in the vinegar variables.

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

In summary, our *test experiments* run the following step-by-step procedure:

1. Select a SNOVA parameter set and then create a key pair $(\mathtt{sk}, \mathtt{pk})$ by calling Algorithm 1.
2. In either scenario, create the bitstring $\mathtt{x}$ by performing either l^2v or $4 \cdot l^2v$ Bernoulli trials given a probability $0 < \rho < 1$. Furthermore, create $\mathtt{L}$ by generating a list of size $|x|$ of random field elements (from either $\mathbb{F}_{16}$ or $\mathbb{F}_2$).
3. Collect N_{msg} signatures by calling the tweaked version of Algorithm 2. The value of N_{msg} depends on the scenario we are running. For the scenario I, N_{msg} is set to $o \cdot l + 2$, while, for the scenario II, N_{msg} is set to $4 \cdot o \cdot l + 2$
4. Once N_{msg} signatures are collected, then call the corresponding recovery algorithm, i.e. either Algorithm 3 for Scenario I or Algorithm 4 for Scenario II.
5. Compare the recovered part of T with the corresponding part of the real T to verify the effectiveness of our recovery algorithms.

Our *experimental results* confirm that our recovery algorithms perform as expected. Specifically, when provided with the required number of faulty signatures, they successfully recover the correct components of T, as outlined in

Sections 3.3 and 3.4. Table 3 summarizes the minimum number of faulty signatures required for each SNOVA parameter set to guarantee the success of our recovery algorithms.

Table 3. Minimum number of signatures per SNOVA parameter set

Security Level	(v, o, q, l, λ)	Recovery from fixed field elements by Algorithm 3	Recovery from fixed bits by Algorithm 4
I	$(37, 17, 16, 2, 128)$	34	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 utilize the C version of the SNOVA code to evaluate our attack by generating faulty signatures. Specifically, we incorporate Algorithm 7 into the C code. The vinegar values are generated within the function `sign_digest_core_ref` found in the file `snova_kernel.h`. To generate these values, SNOVA uses Keccak, and in the reference implementation, the authors rely on the eXtended Keccak Code Package (XKCP)[4]. Listing 1.1 details the method for generating these values and demonstrates how they are assigned to the appropriate structure, a matrix variable named X_IN_GF16MATRIX.

```
// 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++;
```

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

```
                }
            }
        }
```

Listing 1.1. Code snippet of generation of vinegar values.

To generate faulty signatures, we employ the `get_F16` function, which generates random field elements in $\mathbb{F}_{16}$ and creates a binomial random variable array, denoted as x. This array plays a crucial role in determining which entries in the matrix X_IN_GF16MATRIX will be assigned random values versus those that will be derived from the vinegar variables. As illustrated in Listing 1.2, we modify the variable `X_in_GF16Matrix` to incorporate these random values, replacing the original values obtained from the hash function. For more details about the implementation of the function `get_F16`, see Appendix A.

```
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 1.2. Code snippet for setting random values as Algorithm 6.

Now, we use the faulty 'X_IN_GF16MATRIX' to generate the signature, where the matrix 'T12' is part of the private key. The process is shown in Listing 1.3. Specifically, the matrix 'T12' is multiplied with parts of the 'X_IN_GF16MATRIX', and the results are accumulated to form the final signature.

```
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 1.3. Usage of X_IN_GF16MATRIX to generate the final signature.

As one expects, the results obtained from SAGE—when generating signatures and employing our recovery algorithms—are quite comparable to those produced

by the C code for generating faulty signatures, followed by the execution of recovery algorithms in SAGE. The C code and SAGE scripts are publicly accessible in this repository: https://github.com/gbanegas/fault_sim_snova.

4.2 Evaluating the fault-assisted reconciliation attack

Using our SAGE implementation and the C version, we also evaluated the fault attack described in Section 3.7.

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 3.7. We replace line 13 of Algorithm 2 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 2 is run. A "failure" in Step 2 occurs if the modified algorithm cannot compute a signature after one iteration. A success in Algorithm 6 occurs if the secret space can be computed after Step 2 has completed successfully. Therefore, the success rate of Algorithm 6 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 6 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 6's runtime was computed by using Eq. (13) and the Cryptographic Estimators library [10].

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 6 Success Rate		Attack Strategy Success Rate		Algorithm 6 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

5 Countermeasure

In this section, we present a countermeasure for the fault attacks described in Section 3.

This 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 [17].

Specifically, Algorithm 9 implements this countermeasure for SNOVA and should be invoked by Algorithm 2 immediately after executing line 13.

Algorithm 9 accepts three positive integers, Γ and Λ, with the condition that $\Gamma < \Lambda$, and Υ, as well as a tuple of finite field elements $(\alpha_1, \ldots, \alpha_{l^2v})$ of size l^2v. This tuple represents the SNOVA vinegar values $[\mathtt{V}_1, \mathtt{V}_2, \ldots, \mathtt{V}_v]$ generated at line 13 of Algorithm 2. Furthermore, the function `compare`, called by Algorithm 9 at line 14, takes two tuples of size l^2v: $(\alpha_1, \ldots, \alpha_{l^2v})$ and $(\beta_1, \ldots, \beta_{l^2v})$. It returns a tuple of size l^2v where the j-th entry is 1 if $\alpha_j \neq \beta_j$ and 0 otherwise. Finally, the function `checkColumn` takes $(\alpha_1, \ldots, \alpha_{l^2v}), x, \beta, \Upsilon$ and checks if there are at least Υ occurrences of x in the sequence $\mathtt{V}_{1,1\beta}, \ldots, \mathtt{V}_{1,l\beta}, \ldots, \mathtt{V}_{v,l\beta}$.

Algorithm 9 Countermeasure by checking and storing vinegar values

1: **function** `countermesure`$(\Gamma, \Lambda, \Upsilon, (\alpha_1, \ldots, \alpha_{l^2v}))$
2: **for** $x \in \mathbb{F}_q$ **do**
3: **for** $\beta \in \{1, 2, \ldots, l\}$ **do**
4: **if** `checkColumn`$((\alpha_1, \ldots, \alpha_{l^2v}), x, \beta, \Upsilon)$ **then**
5: **return** `fail`
6: **end if**
7: **end for**
8: **end for**
9: **if** `L` has not been created **then**
10: `L` $\leftarrow []$
11: **end if**
12: `count` $\leftarrow [0]^{vl^2}$
13: **for** $(i \leftarrow 0, i < |\mathtt{L}|, i \leftarrow i + 1)$ **do**
14: `count` $\leftarrow$ `count` $+$ `compare`$(\mathtt{L}[i], (\alpha_1, \ldots, \alpha_{l^2v}))$
15: **end for**
16: **if** `count`$[j] > \Gamma$ for some $j \in [l^2v]$ **then**
17: **return** `fail`
18: **end if**
19: **if** $|\mathtt{L}| = \Lambda$ **then**
20: `L.removeEntryAtIndex`(0)
21: **end if**
22: `L.append`$((\alpha_1, \ldots, \alpha_{l^2v}))$
23: **return** `success`

Why does this countermeasure work? Let us assume that the countermeasure is already in place in the signing algorithm, with $\Lambda = l \cdot o$ and $\Gamma < \Lambda$.

We begin by analyzing the check performed between lines 1 and 8, which targets the attack strategy outlined in Section 3.7. Let $\mathtt{V} = (\mathtt{V}_1^t, \ldots, \mathtt{V}_v^t)^t$ and let Z_β be the random variable that counts the number of occurrences of $x \in \mathbb{F}_q$ in the β-th column of $\mathtt{V}$. Z_β follows a binomial distribution with probability $p = 1/q$ in the absence of faults. Therefore, we have to take Υ such that $\Pr(Z_\beta \geq \Upsilon)$, the probability of `checkColumn` returning `True` at line 3, is negligible. For any current SNOVA parameters, $\Upsilon = \lfloor l \cdot v \cdot p + r\sqrt{l \cdot v \cdot p(1-p)} \rfloor$, where $r \geq 6$ is natural number.

On the one hand, if the step described in Section 3.7 successfully fixes at least Υ entries in the β-th column of $\mathtt{V}$ to x, the function `checkColumn` will return `True`. On the other hand, if the faults fix at most $\Upsilon - 1$ entries in any column of $\mathtt{V}$ to x, the corresponding checks will return `False`. In this case, the attacker can proceed with the attack strategy. However, running Algorithm 6 under these conditions would lead to a prohibitive runtime, as discussed in Appendix B.

We now examine the countermeasure against the attack strategy described in Section 3.3.

Assume that $|\mathtt{L}| = \Lambda$, meaning the list maintained by Algorithm 9 has reached its maximum allowed size, and that the `countermeasure` function is called with parameters Γ, Λ, and $(\alpha_1, \ldots, \alpha_{l^2 v})$.

Let E denote the event where line 17 of Algorithm 9 returns `fail` in the absence of fault injection. For this event to occur, there must exist some $j \in [l^2 v]$ such that $\mathtt{count}[j] > \Gamma$, meaning the tuple $(\mathtt{L}[i][j])_{i=0}^{\Lambda-1}$ contains $\mathtt{count}[j]$ instances of α_j. Let X_j be the random variable that counts the number of occurrences of α_j in the tuple $(\mathtt{L}[i][j])_{i=0}^{\Lambda-1}$. Each X_j follows a binomial distribution with probability $p_j = \frac{1}{|\mathbb{F}_q|}$, and the variables are mutually independent. Therefore, the probability that Algorithm 9 reaches line 17 is

$$\begin{aligned} p_{\text{fail}} &= \Pr(X_j > \Gamma \text{ for some } j \in [l^2 v]) \\ &= 1 - \Pr(X_j \leq \Gamma \text{ for all } j \in [l^2 v]). \end{aligned}$$

On the other hand, if an attacker introduces a permanent fault injection, as described in Section 3.3, the signing algorithm will abort after producing Γ faulty signatures. In this case, the attacker could construct $v^2 l$ linear systems (as in Eq. (9)), each consisting of $\Gamma - 1$ equations and lo unknowns. Since $\Gamma - 1 < \Lambda = lo$, these linear systems are under-determined, meaning they have multiple solutions. Moreover, the attacker is unaware of which field elements, if any, from each vinegar variable $\mathtt{V}_i$ are fixed by the permanent fault. In other words, out of the total vl^2 field elements representing a vinegar variable set, the attacker does not know the indices of the fixed elements or their values.

Let us consider the j-th linear system. To solve this system, the attacker could specialize $lo - \Gamma + 1$ variables (i.e., assign random values to $lo - \Gamma + 1$ variables) and then solve the resulting linear system with $\Gamma - 1$ equations and $\Gamma - 1$ unknowns. Consequently, the probability of finding the correct solution

for this particular system is approximately $p_j^{\Gamma-lo-1}$. However, the attacker will not be able to verify whether a given solution is correct.

Therefore, Γ must be chosen such that both p_{fail} and the probability of successfully executing a key recovery attack via fault injection is negligible. Table 5 provides possible values for Γ and Λ corresponding to 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 several fault attack strategies against the SNOVA cryptographic scheme. Initially, we proposed two methods of executing a fault attack, showing 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. Our experiments, implemented in both SAGE and C, demonstrated the efficiency of our algorithm under varying fault conditions.

In addition to these earlier methods, we introduced a new fault-assisted reconciliation attack in Section 3.7. This attack leverages induced transient faults to recover the secret key space by solving a quadratic system. The attack was evaluated using the lowest security parameter set for SNOVA, and the results indicated a high success rate under specific probability conditions for fault occurrence. Our experiments validated the feasibility of this attack, emphasizing its potential to weaken SNOVA's security when fault injections are possible.

To mitigate these vulnerabilities, we proposed a lightweight countermeasure that can effectively reduce the probability of successful key recovery without significantly impacting SNOVA's performance. This countermeasure is flexible and scalable, making it applicable across various SNOVA parameter sets.

Our findings underline the importance of robust fault-resistant implementations in post-quantum cryptographic schemes such as SNOVA. Future work could focus on further optimizing the countermeasures and exploring the impact of these attacks on other cryptographic systems.

References

1. Manuel Agoyan, Jean-Michel Dutertre, Philippe Hoogvorst, Emmanuel Jaulmes, Alfredo Tria, and Florent Valette. Efficient fault attacks on AES. In *IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 16–23. IEEE, 2010.
2. Tobias Aulbach and Florian König. Exploring the implementation security of the post-quantum signature scheme MAYO. In *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2023*, pages 10–30, 2023.
3. Tobias Aulbach, Florian König, and Jürgen Kraämer. Recovering Rainbow's secret key with a first-order fault attack. In *Progress in Cryptology - AFRICACRYPT 2022*, pages 348–368, 2022.
4. 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.
5. Ward Beullens. Improved cryptanalysis of SNOVA. Cryptology ePrint Archive, Paper 2024/1297, 2024.
6. Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO, June 2023. Available at https://pqmayo.org/assets/specs/mayo.pdf.
7. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. *Advances in Cryptology—CRYPTO'97*, pages 513–525, 1997.
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 A. Verbel, Floyd Zweydinger, and Emanuele Bellini. {CryptographicEstimators}: a software library for cryptographic hardness estimation. {*IACR*} *Cryptol. ePrint Arch.*, page 589, 2023.
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. Yoshinori Hashimoto, Tsuyoshi Takagi, and Kouichi Sakurai. General fault attacks on multivariate public key cryptosystems. *Post-Quantum Cryptography*, pages 1–18, 2011.
14. Yasuhiko Ikematsu and Rika Akiyama. Revisiting the security analysis of SNOVA. Cryptology ePrint Archive, Paper 2024/096, 2024. https://eprint.iacr.org/2024/096.

15. Sönke Jendral and Elena Dubrova. MAYO key recovery by fixing vinegar seeds. Cryptology ePrint Archive, Paper 2024/1550, 2024.
16. Yoongu Kim, Peter Daly, Jeremie Kim, Christopher Fallin, Ji Hye Lee, Donghyuk Lee, Chris Lusky, Justin Meza, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.
17. Joachim Krämer and Maurus Loiero. Fault attacks on uov and rainbow. *Constructive Side-Channel Analysis and Secure Design*, pages 193–214, 2019.
18. Peigen Li and Jintai Ding. Cryptanalysis of the SNOVA signature scheme. Cryptology ePrint Archive, Paper 2024/110, 2024. https://eprint.iacr.org/2024/110.
19. 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 nist-pqc: Digital signature schemes project. Proposal for NISTPQC: Digital Signature Schemes project, 2023. https://snova.pqclab.org/.
20. Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and Christoph G. Günther, editors, *Advances in Cryptology — EUROCRYPT '88*, pages 419–453, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.
21. Kaan Mus, Seungwon Shin, and Berk Sunar. Quantumhammer: A practical hybrid attack on the luov signature scheme. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1071–1084, 2020.
22. Shuhei Nakamura, Yusuke Tani, and Hiroki Furue. Lifting approach against the SNOVA scheme. Cryptology ePrint Archive, Paper 2024/1374, 2024.
23. Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1995.
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. Anis Sayari, Denis Butin, and Thomas Fuhr. Practical fault injection attacks on multivariate signature schemes. In *Constructive Side-Channel Analysis and Secure Design - COSADE 2021*, pages 101–121, 2021.
26. Ki Shim and Bum Kyu Koo. Full key recovery on uov with fault injection. *ICISC 2009*, pages 123–140, 2009.
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. https://eprint.iacr.org/2022/1742.

A Implementation details

```
// Function to generate random field element
uint8_t random_field_element() {
```

```
    return rand() % 16; // Adjust based on field size (for GF(2^k), mod
     appropriately)
}

// Function to generate binomial random variable (Bernoulli trial)
uint8_t binomial_trial(double prob) {
    double r = (double)rand() / RAND_MAX;
    return r < prob ? 1 : 0;
}

// Function equivalent to get_F16
void get_F16(int v, int o, int l, uint8_t *I, uint8_t *x, double prob) {
    int size = v * l * l;
    // Generate I array with random field elements
    for (int i = 0; i < size; i++) {
        I[i] = random_field_element();
    }
    // Generate x array with binomial random variables
    for (int i = 0; i < size; i++) {
        x[i] = binomial_trial(prob);
    }
}
```

Listing 1.4. Generate random elements of $\mathbb{F}_{16}$.

B Agorithm 6'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 non-trivial solution u satisfying $\mathcal{P}(u) = a \in \mathbb{F}_q^M$ if such solution exists. The runtime complexity of Algorithm 6 is bounded by

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

field multiplications.