Correction Fault Attack on CROSS under Unknown Bit Flips

Sönke Jendral[1], Elena Dubrova[1], Qian Guo[2], and Thomas Johansson[2]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
{jendral,dubrova}@kth.se
[2] Lund University, Lund, Sweden
{qian.guo,thomas.johansson}@eit.lth.se

Abstract. Recognising the need for PQC signature schemes with different size and performance trade-offs than the ML-DSA and SLH-DSA standards, in 2023 NIST launched a competition for additional signature algorithms. Among the current candidates in this competition is CROSS, a code-based scheme derived from the syndrome-decoding problem and suitable for memory-constrained devices. This paper presents a fault attack on CROSS that recovers the secret key by flipping one or more bits in the scheme's public parity-check matrix. Unlike previous PQC fault attacks that typically rely on precisely controlled fault injections, which is often an unrealistic assumption, our approach exploits bit flips with *unknown* position and value, resembling the Rowhammer fault model. The attack builds upon the correction-based methodology introduced for Dilithium (Euro S&P'22; CHES'24) and exploits structural properties of CROSS to substantially relax attacker requirements. We demonstrate the attack on an ARM Cortex-M4 processor using voltage fault injection. We further show that prior work on partial key exposure attacks (CRYPTO'22) can be extended to CROSS under non-trivial erasure rates, reducing the attack complexity. The attack remains effective in the presence of memory-integrity protection mechanisms such as error-correcting codes. Finally, we propose countermeasures for hardening CROSS implementations against physical attacks.

Keywords: Fault Injection · CROSS · Code-based Cryptography · Post-Quantum Digital Signature · Key Recovery Attack · Partial Key Exposure Attack

1 Introduction

In response to the emerging threat posed by large-scale quantum computers, the National Institute of Standards and Technology (NIST) hosted a competition for post-quantum cryptographic (PQC) algorithms, culminating in the publication of standards for key-encapsulation mechanism ML-KEM [41] and digital signature schemes ML-DSA [40] and SLH-DSA [43] in August 2024, as well as the selection of Falcon [22] and HQC [23] for future standardisation. To enable

deployment of PQC algorithms in a variety of use cases, including on embedded platforms, and to improve confidence in security through diversification of underlying problems, NIST subsequently launched a competition for additional signature schemes [39]. This competition entered its second round in March 2025 [42].

Among the submissions selected to advance to the second round is CROSS [2], a code-based scheme built on a variant of the well-studied syndrome decoding problem. CROSS is designed to be existentially unforgeable under chosen message attacks (EUF-CMA), meaning an adversary with access to a signing oracle and the public key cannot generate a valid signature for a new message without querying the oracle for this message. NIST selected CROSS partly due to its performance benefit over SLH-DSA [44]. The CROSS specification [2] explicitly highlights its suitability for memory-constrained devices, such as smartcards, and its use in X.509 certificates.

This paper presents a fault attack on the second-round variant of CROSS. Historically, fault attacks were considered largely theoretical, relying on fault injection techniques that were difficult to perform outside specialized laboratories. However, advances in fault attack methods, such as Rowhammer [37] and the commoditisation of fault injection equipment, have significantly lowered the bar for attackers, enabling inexpensive attacks on real-world systems without fine-grained control. At the same time, the planned deployment of PQC algorithms, such as CROSS, on resource-constrained embedded devices increases their vulnerability to physical attacks. These developments demonstrate that security analyses for PQC algorithms must extend beyond purely mathematical soundness to account for realistic fault models and physical threats. Ignoring these concerns risks overlooking practical vulnerabilities that could undermine the security of PQC implementations.

Correction fault attacks. The first fault attack on a cryptographic algorithm was presented by Boneh et al. [8] in 1997. It uses random faults to recover the secret of RSA-based schemes. Since then, substantial progress has been made focusing both on devising new methods for deliberately injecting faults, for example, through clock or voltage glitching [33,10,31], electromagnetic pulses [17,36], or laser beams [1,15], and on identifying novel attacks that exploit them. In the context of PQC algorithms, many such attacks have been demonstrated, including against standardised schemes ML-DSA (Dilithium) [19,11,5,29,45] and SLH-DSA (SPHINCS+) [12,48,9].

In 2022, Islam et al. [27] introduced a powerful approach for attacking Dilithium that enables differential-style attacks from a single signature, even in the presence of randomisation measures that would otherwise prevent such attacks. Their key insight is that the verification procedure allows an attacker to repeatedly modify a faulty signature and check its validity, thereby reconstructing the corresponding non-faulty signature. The difference between the two signatures can then reveal information about the secret key, as in a differential attack. They call their approach a *correction attack*, as it involves *correcting* the faulty signature. Subsequently, in 2024, Krahmer et al. [32] expanded on this idea and introduced

a fault attack targeting a public matrix of Dilithium. They argue that public components are unlikely to be protected by side-channel countermeasures, making their attack potentially applicable to otherwise protected implementations.

However, previous fault attacks, including both correction fault attacks [27,32] and a previous fault attack on CROSS [35], commonly assume unrealistically strong attacker capabilities. They also require precise faults, such as the ability to precisely skip certain instructions, or precisely flip certain bits or set them to known values. Furthermore, both previous correction attacks [27,32] target components specific to the Dilithium signature scheme. Since Dilithium and CROSS are fundamentally different schemes based on different underlying paradigms, attacks against one are not generally applicable to the other.

Research Questions. In this work, we address four key research questions:

Q1: Does the correction-based approach generalise beyond Dilithium to CROSS?
Q2: Is the resulting attack feasible under more realistic weak attacker assumptions, specifically, without requiring knowledge of the fault's value or position?
Q3: Can the secret key of CROSS be recovered from partial information revealed by the attack?
Q4: Can such an attack be performed in the presence of countermeasures, such as error correcting codes?

Contributions. Our main contributions are:

i) We present a correction fault attack on CROSS targeting the public matrix $\mathbf{H}$ that is applicable under both the instruction skipping and the arbitrary bit flip fault models. Unlike correction attacks on Dilithium [27,32], and a previous fault attack on CROSS [35], the presented approach does not require knowledge of the fault's value or position. In addition, it is applicable in the presence of error-correcting codes.
ii) We describe how the attack can be performed using instruction skipping during the derivation of $\mathbf{H}$ and experimentally validate the feasibility of this approach on an ARM Cortex-M4 processor.
iii) We further demonstrate that, akin to prior partial key exposure attacks on BIKE, Rainbow, NTRU [20] and RSA [26], the secret key of CROSS can be recovered from a subset of known positions under non-trivial erasure rates, thereby reducing the complexity of the presented attack.
iv) We propose countermeasures to mitigate the attack and discuss their effectiveness and impact on overall performance. We also describe variants of the attack capable of circumventing certain countermeasures.

Organisation of the paper. The rest of this paper is organised as follows. Section 2 describes related work. Section 3 provides background information on the CROSS algorithm and its underlying problem. Section 4 introduces the attack. Section 5 describes post-processing approaches to recover the secret key

from partial information. Section 6 analyses the attack success probability under random bit flips. Section 7 describes the practical evaluation of the attack using voltage fault injection. Section 8 discusses differences between the presented attack and previous correction fault attacks on Dilithium. Section 9 discusses potential countermeasures against the attacks and their effectiveness. Finally, Section 10 concludes the paper.

2 Related work

This section provides an overview of previous physical attacks on CROSS making use of fault injection or side-channel information, as well as fault injection attacks on other PQC algorithms that follow a similar strategy to the presented attack.

2.1 Side-channel and fault attacks on CROSS

Schupp and Sigl [46] present a correlation power analysis (CPA) side-channel attack on the reference implementation of the first-round variant of CROSS. The attack targets the computation of per-round syndromes $\mathbf{s}' = \mathbf{u}[i]\mathbf{H}^\mathsf{T}$ for rounds where $\text{chall}_2[i] = 1$. In these rounds, the vector $\mathbf{u}'[i]$ is revealed in the signature, and recovering the secret key is equivalent to recovering the restricted vector $\mathbf{v}[i]$ where $\mathbf{u}[i] = \mathbf{v}[i] * \mathbf{u}'[i]$. This approach is preferable over recovering $\mathbf{u}[i]$ directly, due to the limited number of possible values in the restricted set. The attack works by correlating hypotheses for individual entries of $\mathbf{v}$ with the power consumption during the vector-matrix multiplication to identify the correct entries. They experimentally validate their attack on a STM32F303RCT7 processor and demonstrate that the attack succeeds using only one or two power traces.

Czuprynko et al. [16] improve upon the attack [46] through several insights: 1) part of the secret vector $\mathbf{e}$ can be recovered from the public key, 2) information from all rounds can be combined in the attack, 3) the search space can be reduced by computing a hypothesis for $\mathbf{e}$ and then deriving the corresponding $\mathbf{u}[i]$, and 4) intermediate computations can also be exploited. They experimentally validate their approach on a STM32F030 processor and demonstrate that it reduces the number of required rounds. As a result, the secret key can be recovered from a single power trace for all parameter sets. They also describe how their attack can be applied to the first-round variant of another code-based scheme, LESS, though it is not effective against the second-round variant of LESS.

Mondal et al. [35] describe fault injection attacks on CROSS and LESS that target the computation of a reference tree used to determine which seeds of a GGM tree are revealed. The approach induces a bit flip that causes both the response for $\text{chall}_2[i] = 0$ and for $\text{chall}_2[i] = 1$ to be revealed, which directly allows recovering the secret key. They simulate their attack and demonstrate that the secret key can be recovered from a single faulty signature, assuming the fault injection always succeeds. The key differences from our work are that 1) they assume the fault only flips a single bit at an attacker-known position to a

known value and 2) they do not experimentally validate their attack. We assess the feasibility of their attack in the presence of countermeasures in Section 9.

2.2 Correction fault attacks on Dilithium

Islam et al. [27] introduce a signature correction attack for lattice-based scheme Dilithium. In their attack, they use Rowhammer to induce a single bit flip in an entry of the vector $\mathbf{s}_1$, which produces a signature that differs from a correct signature in one entry by a value depending on the faulted entry of $\mathbf{s}_1$. Enumerating all possible values for this difference until the corrected signature passes verification reveals the entry of $\mathbf{s}_1$. They experimentally validate their attack on a system with DDR3 memory and demonstrate that it can partially or fully recover 956 of the 1,024 coefficients from 6,853 single-bit faulty signatures, thereby reducing the key recovery complexity to 89 bits. They do not report the number of attempts required to obtain the faulty signatures.

Krahmer et al. [32] extend this idea in two different ways. The first attack aims to disturb the computation of the responses $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ such that it equals one of the summands, allowing the other summand to be revealed through correction of the faulty signature. The second attack, which is most similar to our approach, instead targets a public matrix $\mathbf{A}$. The idea is to change a single entry of $\hat{\mathbf{A}} = \mathbf{A} + \Delta$, which results in $\hat{\mathbf{A}}\mathbf{y} = (\mathbf{A} + \Delta)\mathbf{y} = \mathbf{A}\mathbf{z} - c\mathbf{t} + c\mathbf{s}_2 + \Delta\mathbf{y} \approx \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d + \Delta\mathbf{y}$. This allows recomputing $\mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d + \mathbf{x}$ to find $\mathbf{x} = \Delta\mathbf{y}$. Since Δ is known and has only one entry, this reveals an entry of $\mathbf{y}$ and therefore the secret key. Both attacks require one successful fault to recover one coefficient of the secret key. They practically evaluate their second attack on a STM32F405 processor and demonstrate that it can recover 961 of 1,024 coefficients using 10,240 signatures, and, by applying lattice reduction, the complete secret key.

We discuss the differences between these attacks [27,32] and the presented attack in more detail in Section 8.

3 Background

3.1 Notation

We adopt the notation in the CROSS specification [2]: Vectors over $\mathbb{F}_p$ are denoted using bold letters, e.g., $\mathbf{e} \in \mathbb{E}^n \subset \mathbb{F}_p^n$; matrices over $\mathbb{F}_p$ using bold capital letters, e.g., $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$; vectors over $\mathbb{F}_z$ using bold overlined letters, e.g., $\bar{\mathbf{e}} \in \mathbb{F}_z^n$. The $k \times k$ identity matrix is denoted as Id_k, and concatenation of x, y is denoted as $x \mid y$. Additionally, we use $\mathbf{u}[i] \in \mathbb{F}_p^n$ to denote the i-th vector in a list of vectors, and use $\mathbf{u}[i]_j \in \mathbb{F}_p$ to denote its j-th entry. When explaining our attack, we use tildes to indicate faulty matrices or vectors, e.g., $\tilde{\mathbf{s}}$, and hats to indicate matrices, vectors, or scalars guessed by an attacker, e.g., $\hat{\mathbf{e}}$.

3.2 Linear Codes and Syndrome Decoding

Definition 1 (Linear code). *A $[n,k]$-linear code $\mathcal{C}$ of length n over $\mathbb{F}_p$ is a k-dimensional subspace of $\mathbb{F}_p^n$. A matrix $\mathbf{G} \in \mathbb{F}_p^{k\times n}$ is a* generator matrix *of $\mathcal{C}$ if $\mathcal{C} = \{\mathbf{uG} : \mathbf{u} \in \mathbb{F}_p^k\}$. A matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$ is a* parity-check matrix *for $\mathcal{C}$ if $\mathbf{GH}^\mathsf{T} = \mathbf{0}$. Both $\mathbf{G}$ and $\mathbf{H}$ can be represented in* systematic *form using a matrix $\mathbf{A} \in \mathbb{F}_p^{k\times(n-k)}$ as $\mathbf{G} = \begin{pmatrix}\mathrm{Id}_k & \mathbf{A}\end{pmatrix}$ and $\mathbf{H} = \begin{pmatrix}-\mathbf{A}^\mathsf{T} & \mathrm{Id}_{n-k}\end{pmatrix}$. The matrices $\mathbf{A}$ and $-\mathbf{A}^\mathsf{T}$ are called the* non-systematic *parts of $\mathbf{G}$ and $\mathbf{H}$, respectively.*

CROSS is a code-based signature scheme that relies on the assumed hardness of the so-called Restricted Syndrome Decoding Problem (R-SDP) [2]. The R-SDP considers a restricted set $\mathbb{E} = \langle g\rangle = \{g^i \mid i \in \mathbb{F}_z\} \subseteq \mathbb{F}_p^*$, which is a cyclic subgroup of order z of the multiplicative group $\mathbb{F}_p^*$, where g is a generator. The parameter sets for the CROSS-R-SDP variants use values $g = 2, z = 7$, and $p = 127$, giving $\mathbb{E} = \{1, 2, 4, 8, 16, 32, 64\}$. The restricted set allows representing entries of $\mathbf{e}$ with their exponents, thereby reducing their size.

Definition 2 (R-SDP). *Given a full-rank parity-check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, a syndrome $\mathbf{s} \in \mathbb{F}_p^{n-k}$, and a restricted set $\mathbb{E}$, find a vector $\mathbf{e} \in \mathbb{E}^n$ such that $\mathbf{s} = \mathbf{eH}^\mathsf{T}$.*

The R-SDP has been shown to be NP-complete [3] and CROSS parameters are chosen such that the problem remains hard even when using state-of-the-art solvers. For more details, we refer to the security analysis of the CROSS submission [3].

A second variant of the R-SDP, the R-SDP(G) problem, relies on an additional multiplicative subgroup

$$G = \langle \mathbf{a}_1, \ldots, \mathbf{a}_m\rangle = \left\{ \bigstar_{i=1}^m \mathbf{a}_i^{\bar{u}_i} \mid \bar{u}_i \in \mathbb{F}_z \right\} \subset \mathbb{E}^n,$$

where $\mathbf{a}_i$ are generators and $\star$ is componentwise multiplication. This subgroup can also be represented using a generator matrix

$$\bar{\mathbf{M}} \coloneqq \begin{pmatrix} (\bar{\mathbf{a}}_1)_1 & \cdots & (\bar{\mathbf{a}}_1)_n \\ \vdots & & \vdots \\ (\bar{\mathbf{a}}_m)_1 & \cdots & (\bar{\mathbf{a}}_m)_n \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{a}}_1 \\ \vdots \\ \bar{\mathbf{a}}_m \end{pmatrix}$$

such that $\mathbf{e} = g^{\bar{\mathbf{e}}} \in G \iff \bar{\mathbf{e}} = \bar{\mathbf{e}}_G\bar{\mathbf{M}}$, where $\bar{\mathbf{e}}_G$ is the information vector belonging to $\mathbf{e}$. This allows further reducing the size of the vector [2]. It is also possible to derive the corresponding parity-check matrix $\bar{\mathbf{H}} \in \mathbb{F}_z^{(n-m)\times n}$ from $\bar{\mathbf{M}}$ and use it to represent G. The corresponding problem is almost identical to the R-SDP.

Definition 3 (R-SDP(G)). *Given a full-rank parity-check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, a syndrome $\mathbf{s} \in \mathbb{F}_p^{n-k}$, and a subset G of $\mathbb{E}^n$, find a vector $\mathbf{e} \in G$ such that $\mathbf{s} = \mathbf{eH}^\mathsf{T}$.*

In the context of our attack, the distinction between R-SDP and R-SDP(G) is not relevant. For simplicity, we thus focus on the R-SDP variant, though the attack translates directly to the R-SDP(G) variant.

Table 1. Overview of the CROSS parameter sets [2]. Variants 'fast', 'balanced', and 'small' have different signature sizes and performance characteristics not shown here.

Algorithm	Variant	p	z	n	k	m	t	w
CROSS-R-SDP 1	fast	127	7	127	76	-	157	82
	balanced						256	215
	small						520	488
CROSS-R-SDP 3	fast	127	7	187	111	-	239	125
	balanced						384	321
	small						580	527
CROSS-R-SDP 5	fast	127	7	251	150	-	321	167
	balanced						512	427
	small						832	762
CROSS-R-SDP(G) 1	fast	509	127	55	36	25	147	76
	balanced						256	220
	small						512	484
CROSS-R-SDP(G) 3	fast	509	127	79	48	40	224	119
	balanced						268	196
	small						512	463
CROSS-R-SDP(G) 5	fast	509	127	106	69	48	300	153
	balanced						356	258
	small						642	575

Algorithm 1 CROSS.KeyGen() [2]

Output: Public key pk, secret key sk

1: $\text{Seed}_{\text{sk}} \leftarrow \{0,1\}^{2\lambda}$
2: $(\text{Seed}_{\mathbf{e}}, \text{Seed}_{\text{pk}}) \leftarrow \mathsf{CSPRNG}(\text{Seed}_{\text{sk}} \mid 3t+1)$
3: $\mathbf{V} \leftarrow \mathsf{CSPRNG}(\text{Seed}_{\text{pk}} \mid 3t+2)$
4: $\mathbf{H} \leftarrow [\mathbf{V} \mid \text{Id}_{n-k}]$
5: $\bar{\mathbf{e}} \leftarrow \mathsf{CSPRNG}(\text{Seed}_{\mathbf{e}} \mid 3t+3)$
6: $\mathbf{e} \leftarrow g^{\bar{\mathbf{e}}}$
7: $\mathbf{s} \leftarrow \mathbf{e}\mathbf{H}^{\mathsf{T}}$
8: **return** $(\text{sk} := \text{Seed}_{\text{sk}}, \text{pk} := (\text{Seed}_{\text{pk}}, \mathbf{s}))$

3.3 CROSS algorithm

This section describes the key generation, signing, and verification algorithms of the R-SDP variant of CROSS. Table 1 provides an overview of the main parameters for all specified parameter sets. For further details, including the R-SDP(G) variant, we refer to the specification [2].

Key Generation (Algorithm 1). The key generation algorithm computes a random seed Seed_{sk} and derives a secret and public seed from it. The public seed Seed_{pk} is used to sample a random parity-check matrix $\mathbf{H}$, which consists of a random block $\mathbf{V}$ and an identity block Id_{n-k}. The secret seed $\text{Seed}_{\mathbf{e}}$ is used to compute the secret restricted vector $\mathbf{e}$ by sampling its exponents $\bar{\mathbf{e}}$. The algorithm then computes the syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^{\mathsf{T}}$. The secret key consists of the seed Seed_{sk} (to allow rederiving the public seed), while the public key consists of the public seed Seed_{pk} and the syndrome $\mathbf{s}$.

Algorithm 2 CROSS.Sign(sk, Msg) [2]

Input: Secret key sk, message Msg
Output: Signature Sgn

$\bar{\mathbf{e}}, \mathbf{H} \leftarrow \mathsf{ExpandSK}(\text{Seed}_{\text{sk}})$
Seed $\leftarrow \{0,1\}^{\lambda}$, Salt $\leftarrow \{0,1\}^{2\lambda}$
(Seeds[1], ..., Seeds[t]) $\leftarrow \mathsf{SeedLeaves}(\text{Seed}, \text{Salt})$
for i **from** 1 **to** t **do**
  $\bar{\mathbf{e}}'[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}(\text{Seed}[i] \mid \text{Salt} \mid i + 2t - 1)$
  $\bar{\mathbf{v}}[i] \leftarrow \bar{\mathbf{e}} - \bar{\mathbf{e}}'[i]$
  $\mathbf{v}[i] \leftarrow g^{\bar{\mathbf{v}}[i]}$
  $\mathbf{u}[i] \leftarrow \mathbf{v}[i] * \mathbf{u}'[i]$
  $\mathbf{s}'[i] \leftarrow \mathbf{u}[i]\mathbf{H}^{\mathsf{T}}$
  $\text{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \bar{\mathbf{v}}[i] \mid \text{Salt} \mid i + 2t - 1)$
  $\text{cmt}_1[i] \leftarrow \mathsf{Hash}(\text{Seed}[i] \mid \text{Salt} \mid i + 2t - 1)$
$\text{digest}_{\text{cmt}_0} \leftarrow \mathsf{TreeRoot}(\text{cmt}_0)$
$\text{digest}_{\text{cmt}_1} \leftarrow \mathsf{Hash}(\text{cmt}_1)$
$\text{digest}_{\text{cmt}} \leftarrow \mathsf{Hash}(\text{digest}_{\text{cmt}_0} \mid \text{digest}_{\text{cmt}_1})$
$\text{digest}_{\text{chall}_1} \leftarrow \mathsf{Hash}(\mathsf{Hash}(\text{Msg}) \mid \text{digest}_{\text{cmt}} \mid \text{Salt})$
$\text{chall}_1 \leftarrow \mathsf{CSPRNG}(\text{digest}_{\text{chall}_1} \mid 3t - 1)$
for i **from** 1 **to** t **do**
  $\mathbf{e}'[i] \leftarrow g^{\bar{\mathbf{e}}'[i]}$
  $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \text{chall}_1[i]\mathbf{e}'[i]$
$\text{digest}_{\text{chall}_2} \leftarrow \mathsf{Hash}(\mathbf{y} \mid \text{digest}_{\text{chall}_2})$
$\text{chall}_2 \leftarrow \mathsf{CSPRNG}(\text{digest}_{\text{chall}_2} \mid 3t)$
Proof $\leftarrow \mathsf{TreeProof}(\text{cmt}_0, \text{chall}_2)$
Path $\leftarrow \mathsf{SeedPath}(\text{Seed}, \text{Salt}, \text{chall}_2)$
for i **from** 1 **to** t **do**
  if $\text{chall}_2[i] = 0$ **then**
    $\text{resp}[i]_0 \leftarrow (\mathbf{y}[i], \bar{\mathbf{v}}[i])$
    $\text{resp}[i]_1 \leftarrow \text{cmt}_1[i]$
return Sgn $\coloneqq$ (Salt, $\text{digest}_{\text{cmt}}$, $\text{digest}_{\text{chall}_2}$, Path, Proof, resp)

Signing (Algorithm 2). The signing algorithm begins by constructing a set of seeds using a GGM tree. Such a tree allows efficiently communicating only a subset of seeds with logarithmic cost, by revealing intermediate nodes instead of leaves. The algorithm then computes per-round syndromes $\mathbf{s}'[i] = \mathbf{u}[i]\mathbf{H}^{\mathsf{T}}$ by sampling random vectors $\bar{\mathbf{e}}'$ and $\mathbf{u}'$, as well as commitments to the syndrome and the seed. This step requires multiplication with the parity-check matrix $\mathbf{H}$. The commitments are combined across all rounds to form digests, which in turn are used to sample the first challenge. The algorithm then computes the responses $\mathbf{y}$ for each round, which are again combined and used to sample the second challenge. This challenge dictates for each round whether to reveal the per-round seed (allowing independent recomputation of the response $\mathbf{y}$ and corresponding commitment to check their correctness) or to reveal $\mathbf{y}$ and $\mathbf{v}$ (allowing verification that $\mathbf{v}$ is in the restricted set and recomputation of the corresponding commitment to check its correctness). This is realised by computing paths of the

Algorithm 3 CROSS.Verify(pk, Msg, Sgn) [2]

Input: Public key pk, message Msg, signature Sgn
Output: Boolean

$(\text{Seed}_{\text{pk}}, \mathbf{s}) \leftarrow \text{pk}$
$(\text{Salt}, \text{digest}_{\text{cmt}}, \text{digest}_{\text{chall}_2}, \text{Path}, \text{Proof}, \text{resp}) \leftarrow \text{Sgn}$
$\mathbf{V} \leftarrow \mathsf{CSPRNG}(\text{Seed}_{\text{pk}} \mid 3t+2)$
$\mathbf{H} \leftarrow [\mathbf{V} \mid \text{Id}_{n-k}]$
$\text{digest}_{\text{chall}_1} \leftarrow \mathsf{Hash}(\mathsf{Hash}(\text{Msg}) \mid \text{digest}_{\text{cmt}} \mid \text{Salt})$
$\text{chall}_1 \leftarrow \mathsf{CSPRNG}(\text{digest}_{\text{chall}_1} \mid 3t-1)$
$\text{chall}_2 \leftarrow \mathsf{CSPRNG}(\text{digest}_{\text{chall}_2} \mid 3t)$
$(\text{Seed}[i])_{i:\text{chall}_2[i]=1} \leftarrow \mathsf{RebuildLeaves}(\text{Path}, \text{chall}_2, \text{Salt})$
for i **from** 1 **to** t **do**
  if $\text{chall}_2[i] = 1$ **then**
    $\text{cmt}_1[i] \leftarrow \mathsf{Hash}(\text{Seed}[i] \mid \text{Salt} \mid i + 2t - 1)$
    $\bar{\mathbf{e}}'[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}(\text{Seed}[i] \mid \text{Salt} \mid i + 2t - 1)$
    $\mathbf{e}'[i] \leftarrow g^{\bar{\mathbf{e}}[i]}$
    $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \text{chall}_1[i]\mathbf{e}'[i]$
  if $\text{chall}_2[i] = 0$ **then**
    $\text{cmt}_1[i] \leftarrow \text{resp}[i]_1$
    $(\mathbf{y}[i], \bar{\mathbf{v}}[i]) \leftarrow \text{resp}[i]_0$
    Verify $\bar{\mathbf{v}} \in \mathbb{F}_z^n$, otherwise abort
    $\mathbf{v}[i] \leftarrow g^{\bar{\mathbf{v}}[i]}$
    $\mathbf{y}'[i] \leftarrow \mathbf{v}[i] * \mathbf{y}[i]$
    $\mathbf{s}'[i] \leftarrow \mathbf{y}'[i]\mathbf{H}^{\mathsf{T}} - \text{chall}_1[i]\mathbf{s}$
    $\text{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \bar{\mathbf{v}}[i] \mid \text{Salt} \mid i + 2t - 1)$
$\text{digest}_{\text{cmt}_0} \leftarrow \mathsf{RecomputeRoot}(\text{cmt}_0, \text{Proof}, \text{chall}_2)$
$\text{digest}_{\text{cmt}_1} \leftarrow \mathsf{Hash}(\text{cmt}_1)$
$\text{digest}'_{\text{cmt}} \leftarrow \mathsf{Hash}(\text{digest}_{\text{cmt}_0} \mid \text{digest}_{\text{cmt}_1})$
$\text{digest}'_{\text{chall}_2} \leftarrow \mathsf{Hash}(\mathbf{y} \mid \text{digest}_{\text{chall}_1})$
return true if $\text{digest}_{\text{cmt}} = \text{digest}'_{\text{cmt}}$ **and** $\text{digest}_{\text{chall}_2} = \text{digest}'_{\text{chall}_2}$ **else false**

GGM tree that reveal exactly the required nodes with minimal communication cost, along with a proof that allows verifying the correctness of the tree from a recomputed set of hidden leaves. Finally, the algorithm outputs the responses for rounds where both $\mathbf{y}$ and $\mathbf{v}$ are revealed, and combines the salt, two digests, the path, the proof, and the responses to form the signature.

Verification (Algorithm 3). The verification algorithm extracts the components of the signature, reconstructs the parity-check matrix, and recomputes the first and second challenge. It then reconstructs the revealed leaves in the GGM tree. For each round, it either recomputes the expected response $\mathbf{y}$ and corresponding commitment, or it verifies that $\mathbf{v}$ is in the restricted set and recomputes the corresponding commitment from the per-round syndrome $\mathbf{s}'$. The algorithm then combines the commitments to compute the digests, and accepts the signature as valid if the digests match those in the signature.

4 Faulting the Parity-Check Matrix

Intuitive idea. To illustrate the idea behind the attack, we begin with a simple example in terms of the underlying R-SDP, rather than the full CROSS signature scheme. Let $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$ be a random parity-check matrix and let $\mathbf{s} = \mathbf{e}\mathbf{H}^\mathsf{T}$ be the syndrome of a secret vector $\mathbf{e}$.

We assume that an attacker faults the parity-check matrix to

$$\tilde{\mathbf{H}} = \mathbf{H} + \Delta,$$

where $\Delta \in \mathbb{F}_p^{(n-k)\times n}$ has exactly one non-zero entry. For now, we further assume that this value and its position are known to the attacker. If the attacker then learns a faulty syndrome

$$\tilde{\mathbf{s}} = \mathbf{e}\tilde{\mathbf{H}}^\mathsf{T} = \mathbf{e}\left(\mathbf{H} + \Delta\right)^\mathsf{T},$$

the difference between this faulty syndrome and the non-faulty syndrome,

$$\tilde{\mathbf{s}} - \mathbf{s} = \mathbf{e}\left(\tilde{\mathbf{H}} - \mathbf{H}\right)^\mathsf{T} = \mathbf{e}\Delta^\mathsf{T},$$

directly reveals an entry of $\mathbf{e}$.

Extension to CROSS. The idea described above is not directly applicable to CROSS. In particular, the syndrome computation involving the secret vector $\mathbf{e}$ only occurs once during key generation, and is therefore not exploitable, as injecting a fault to cause computation of a faulty syndrome would then prevent the attacker from learning the non-faulty syndrome.

Instead, we turn our attention to the per-round syndromes $\mathbf{s}'[i] = \mathbf{u}[i]\mathbf{H}^\mathsf{T}$, computed for vectors $\mathbf{u}[i] = \mathbf{u}'[i] * \mathbf{v}[i]$. If an attacker learns a faulty per-round syndrome $\tilde{\mathbf{s}}'[i]$, the same approach as above can be used to recover an entry of $\mathbf{u}[i]$ instead of $\mathbf{e}$. Knowledge of an entry of $\mathbf{u}[i]$ together with the public response

$$(\mathbf{y}[i] = \mathbf{u}'[i] + \mathrm{chall}_1[i]\mathbf{e}'[i], \mathbf{v}[i] = \mathbf{e}/\mathbf{e}'[i])$$

directly reveals an entry of $\mathbf{e}$.

However, this approach is also not directly applicable to CROSS: While it is possible to compute the non-faulty per-round syndromes $\mathbf{s}'[i]$ using the approach from the verification algorithm, namely by computing

$$\mathbf{s}'[i] = \left(\mathbf{y}[i] * \mathbf{v}[i]\right)\mathbf{H}^\mathsf{T} - \mathrm{chall}_1[i]\mathbf{s},$$

the faulty per-round syndromes $\tilde{\mathbf{s}}'[i]$ do not appear explicitly in the signature. Instead, the signature contains only a hash of all per-round syndromes. It is still possible to determine the values of the faulty per-round syndromes, by modifying the non-faulty per-round syndromes until the computed hash over these modified syndromes matches that in the signature. In other words, the approach is to *correct* the signature until it passes verification, hence this attack is a *correction attack*.

Due to the large number of rounds and the fact that the differences between the non-faulty and faulty syndromes depend on the per-round random vectors $\mathbf{u}'[i]$, such an approach is not straightforward and needs to be aware of the signature structure across all rounds to be computationally feasible. We furthermore want to relax the assumptions regarding the attacker's capabilities. Specifically, the attack would be significantly more practical if it did not require knowledge of the fault's value or position in the parity-check matrix and if it could be performed under imprecise faults.

A structure-aware approach. We thus make the following observation about the applicability of a signature correction approach that takes into consideration the structure of the signature.

Theorem 1 (Structure-Aware Correction). *Let $\tilde{\mathbf{H}} = \mathbf{H} + \Delta$ be a faulty parity-check matrix where Δ has exactly one non-zero entry $\delta_{x_1,x_2} \neq 0$, and let $\tilde{\mathbf{s}}'[i] = \mathbf{u}[i]\tilde{\mathbf{H}}^\mathsf{T}$ be the per-round syndromes computed from this parity-check matrix for rounds $i \in [1..t]$. Let $\hat{\Delta} \in \mathbb{F}_p^{(n-k)\times n}$ be a matrix of the same structure as Δ, i.e., with exactly one non-zero entry $\hat{\delta}_{\hat{x}_1,\hat{x}_2}$, and let $\hat{\mathbf{e}} \in (\mathbb{E} \cup \{0\})^n$ be a vector with exactly one non-zero entry, $\hat{\mathbf{e}}_{\hat{x}_2} \neq 0$. If*

$$\tilde{\mathbf{s}}'[i] = (\mathbf{y}[i] * \mathbf{v}[i])(\mathbf{H} + \hat{\Delta})^\mathsf{T} - \mathrm{chall}_1[i](\mathbf{s} + \hat{\mathbf{e}}\hat{\Delta}^\mathsf{T}) \tag{1}$$

holds for at least $t - w$ rounds, then $\mathbf{e}_{\hat{x}_2} = \hat{\mathbf{e}}_{\hat{x}_2}$, except with negligible probability.

Proof. We simplify Equation (1):

$$\begin{aligned} \tilde{\mathbf{s}}'[i] = (\mathbf{y}[i] * \mathbf{v}[i])(\mathbf{H} + \hat{\Delta})^\mathsf{T} - \mathrm{chall}_1[i](\mathbf{s} + \hat{\mathbf{e}}\hat{\Delta}^\mathsf{T}) &\iff \\ \mathbf{u}[i](\mathbf{H} + \Delta)^\mathsf{T} = \mathbf{u}[i](\mathbf{H} + \hat{\Delta})^\mathsf{T} + \mathrm{chall}_1[i](\mathbf{e} - \hat{\mathbf{e}})\hat{\Delta}^\mathsf{T} &\iff \\ \mathbf{u}[i](\Delta - \hat{\Delta})^\mathsf{T} = \mathrm{chall}_1[i](\mathbf{e} - \hat{\mathbf{e}})\hat{\Delta}^\mathsf{T}. & \end{aligned} \tag{2}$$

Clearly, this equation holds when $\Delta = \hat{\Delta}$ and $\mathbf{e}_{\hat{x}_2} = \hat{\mathbf{e}}_{\hat{x}_2}$. We can ignore the case where the equation holds for $\Delta \neq \hat{\Delta}$ and $\mathbf{e}_{\hat{x}_2} = \hat{\mathbf{e}}_{\hat{x}_2}$, as it still allows us to identify the correct $\mathbf{e}_{\hat{x}_2}$. We thus only consider the case where the equation holds for $t - w$ rounds despite $\mathbf{e}_{\hat{x}_2} \neq \hat{\mathbf{e}}_{\hat{x}_2}$.

If the non-zero entries in Δ^T and $\hat{\Delta}^\mathsf{T}$ occur in different rows, i.e., if $x_2 \neq \hat{x}_2$, then $\mathbf{u}[i]_{x_1} = 0$ for all $t - w$ rounds, which occurs only with probability $p^{-(t-w)}$. Since this probability is already negligible, we do not further consider this case and now simply assume $x_2 = \hat{x}_2$.

We further simplify Equation (2) to

$$\mathbf{u}[i]_{x_1}\delta_{x_1,x_2} - \mathbf{u}[i]_{\hat{x}_1}\hat{\delta}_{\hat{x}_1,x_2} = \mathrm{chall}_1[i](\mathbf{e}_{\hat{x}_1} - \hat{\mathbf{e}}_{\hat{x}_1})\hat{\delta}_{\hat{x}_1,x_2}. \tag{3}$$

If $x_1 \neq \hat{x}_1$, then Equation (3) imposes a linear constraint on $\mathbf{u}[i]_{x_1}, \mathbf{u}[i]_{\hat{x}_1}$ and $\mathrm{chall}_1[i]$, which holds only with negligible probability $p^{-(t-w)}$ across all $t - w$ rounds for uniformly random $\mathbf{u}$ and chall_1.

If $x_1 = \hat{x}_1$, then Equation (3) has two cases. The first is $\delta_{x_1,x_2} \neq \hat{\delta}_{x_1,x_2}$, which is a linear constraint on $\mathbf{u}[i]_{x_1}$ and $\mathrm{chall}_1[i]$, and holds with per-round

probability p^{-1}. The second is $\delta_{x_1,x_2} = \hat{\delta}_{x_1,x_2}$, which requires $\text{chall}_1[i] = 0$, and again holds with per-round probability p^{-1}. In both cases, the overall probability is negligible across all $t - w$ rounds. □

Remark 1. The approach is stated for the case where the parity-check matrix $\tilde{\mathbf{H}}$ contains exactly one faulty entry. However, it can be extended to cases where multiple entries of $\tilde{\mathbf{H}}$ are faulty. The faulty entries should occur in different rows of $\tilde{\mathbf{H}}^{\mathsf{T}}$, to reveal multiple entries of $\mathbf{e}$ from a single signature. The probability of Equation (1) to hold for an incorrect $\hat{\mathbf{e}}$ under multiple faulty entries is lower than that under a single faulty entry, and therefore remains negligible.

Remark 2. The requirement for Equation (1) to hold for at least $t - w$ rounds stems from the signature structure. To compute the right-hand side of Equation (1) for round i, the vectors $\mathbf{y}[i]$ and $\mathbf{v}[i]$ must be revealed to the attacker, which occurs when $\text{chall}_2[i] = 0$. Since chall_2 is a t-dimensional bit vector with weight w, exactly $t - w$ such rounds exist and can be used in the approach. While the considerations above might suggest that the approach could work with fewer rounds while maintaining a negligible failure probability, in practice, it is necessary to compute (or correct) the $\tilde{\mathbf{s}}'$ values for all $t - w$ rounds to verify the guess by computing the hash. Therefore, there is no advantage for an attacker in using fewer rounds.

Algorithm. Based on the observation in Theorem 1, it is possible to modify the CROSS verification algorithm, so that it can be used to recover an entry of $\mathbf{e}$ from a successfully faulted signature. Algorithm 4 shows this approach. The algorithm simultaneously enumerates all possible matrices $\hat{\Delta}$ with exactly one changed entry, and all possible weight-1 vectors $\hat{\mathbf{e}}$, checking whether Equation (1) holds for a specific pair across all rounds. The check is realised by computing $\text{digest}_{\text{cmt}_0}$, a hash of each per-round syndrome $\mathbf{s}'$. The algorithm also recomputes the $\text{digest}_{\text{cmt}_1}$ value, which is not affected by the fault but is required to recompute the $\text{digest}'_{\text{cmt}}$ value, which can be compared to the $\text{digest}_{\text{cmt}}$ from the signature. If both digests match, the recomputed per-round syndromes s' match those in the signature (i.e., Equation (1) holds for all $t - w$ rounds for which the syndromes can be recomputed), and, by Theorem 1, the correct guess for $\hat{\mathbf{e}}$ is returned.

Complexity. The complexity of Algorithm 4 is dominated by the enumeration of the non-zero entry in $\hat{\Delta}$ and the non-zero entry in $\hat{\mathbf{e}}$, which differs slightly between parameter sets and variants. Generally, the maximum number of attempts is given by

$$(n - k) \cdot n \cdot (p - 1) \cdot z,$$

where each attempt requires at most $t - w + 6$ (fast) or $2t - w$ (small, balanced) hash computations for verification, ignoring savings from the tree compression. Table 2 shows the complexities for the different parameter sets in terms of the number of verification attempts.

Table 2. Approximate algorithmic complexity in terms of the number of verification attempts for Algorithm 4 across different parameter sets.

Parameter set	p	z	n	k	Complexity
CROSS-R-SDP 1	127	7	127	76	$\approx 2^{22.45}$
CROSS-R-SDP 3	127	7	187	111	$\approx 2^{23.58}$
CROSS-R-SDP 5	127	7	251	150	$\approx 2^{24.41}$
CROSS-R-SDP(G) 1	509	127	55	36	$\approx 2^{26.00}$
CROSS-R-SDP(G) 3	509	127	79	48	$\approx 2^{26.67}$
CROSS-R-SDP(G) 5	509	127	106	69	$\approx 2^{27.91}$

5 Partial Key Exposure Attack on CROSS

The method described in Section 4 is applicable under several fault models and weak assumptions regarding attacker capabilities. However, the approach is fundamentally limited to recovering only one or very few entries of the secret vector $\mathbf{e}$ from each faulty signature. In this section, we discuss methods to recover the full vector $\mathbf{e}$ with partial information about its entries, thereby reducing the number of required faults.

Formally, such a model was described by Esser, May, Verbel, and Wen [20] as a *Partial Key Exposure Attack* in the context of BIKE, Rainbow, and NTRU, as well as by Heninger and Shacham in the context of RSA [26]. As our approach allows to recover individual entries of $\mathbf{e}$ with only negligible failure probability, we are focusing on what they call the *erasure model*, in which entries of $\mathbf{e}$ are erased (unknown) with a certain *erasure rate* r, but known positions are assumed to be correct.

Theorem 2 (Reduction of R-SDP(G) under Partial Key Exposure). *Let an instance of the computational R-SDP(G) problem be defined by parameters (p, z, n, k, m), as described in Section 3.2. Assume an attacker gains knowledge of a sub-vector $\mathbf{e}_I$ of $\mathbf{e}$ at a set of c distinct indices $I \subset \{1, \ldots, n\}$. Let $J = \{1, \ldots, n\} \setminus I$ be the set of $n' = n - c$ unknown indices. Then, the problem of finding the remaining sub-vector $\mathbf{e}_J$ is computationally equivalent to solving a new, independent instance of R-SDP(G) defined by the parameters (p, z, n', k', m') and a new public key $(\mathbf{H}', \mathbf{s}')$, where:*

- *New Length:* $n' = n - c$
- *New Dimension:* $k' = k - c$
- *New Subgroup Dimension:* $m' = (n - c) - \mathrm{rank}(\bar{\mathbf{H}}_J)$, *where* $\bar{\mathbf{H}}_J$ *is the submatrix of* $\bar{\mathbf{H}}$ *containing columns indexed by* J.
- *New Parity-Check Matrix:* $\mathbf{H}'$ *is the submatrix of* $\mathbf{H}$ *with columns indexed by* J.
- *New Syndrome:* $\mathbf{s}' = \mathbf{s} - \mathbf{e}_I \mathbf{H}_I^\top$, *where* $\mathbf{H}_I$ *is the submatrix of* $\mathbf{H}$ *with columns indexed by* I.

The security of the original instance is thereby reduced from Work(n, k, m) to Work(n', k', m'). The case for R-SDP is recovered by setting $m = n$ and $m' = n'$.

Algorithm 4 $\mathsf{Recover}(\mathrm{pk}, \mathrm{Msg}, \mathrm{Sgn})$

Input: Public key pk, message Msg, signature Sgn
Output: Entry of $\mathbf{e}$ or $\perp$

$(\mathrm{Seed}_{\mathrm{pk}}, \mathbf{s}) \leftarrow \mathrm{pk}$
$(\mathrm{Salt}, \mathrm{digest}_{\mathrm{cmt}}, \mathrm{digest}_{\mathrm{chall}_2}, \mathrm{Path}, \mathrm{Proof}, \mathrm{resp}) \leftarrow \mathrm{Sgn}$
$\mathbf{V} \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_p^{(n-k)\times k}}(\mathrm{Seed}_{\mathrm{pk}}, 3t+2)$
$\mathbf{H} \leftarrow [\mathbf{V} \mid \mathrm{Id}_{n-k}]$
$\mathrm{digest}_{\mathrm{Msg}} \leftarrow \mathsf{Hash}(\mathrm{Msg})$
$\mathrm{digest}_{\mathrm{chall}_1} \leftarrow \mathsf{Hash}(\mathrm{digest}_{\mathrm{Msg}} \mid \mathrm{digest}_{\mathrm{cmt}} \mid \mathrm{Salt})$
$\mathrm{chall}_1 \leftarrow \mathsf{CSPRNG}_{-(\mathbb{F}_p^*)^t}(\mathrm{digest}_{\mathrm{chall}_1} \mid 3t-1)$
$\mathrm{chall}_2 \leftarrow \mathsf{CSPRNG}_{-\mathcal{B}_{(t,w)}}(\mathrm{digest}_{\mathrm{chall}_2} \mid 3t)$
$(\mathrm{Seed}[i])_{i:\mathrm{chall}_2[i]=1} \leftarrow \mathsf{RebuildLeaves}(\mathrm{Path}, \mathrm{chall}_2, \mathrm{Salt})$
for i **from** 1 **to** t **do**
  if $\mathrm{chall}_2[i] = 1$ **then** $\mathrm{cmt}_1[i] \leftarrow \mathsf{Hash}(\mathrm{Seed}[i] \mid \mathrm{Salt} \mid i + 2t - 1)$
  else $\mathrm{cmt}_1[i] \leftarrow \mathrm{resp}[i]_1$
$\mathrm{digest}_{\mathrm{cmt}_1} \leftarrow \mathsf{Hash}(\mathrm{cmt}_1[1] \mid \cdots \mid \mathrm{cmt}_1[t])$
for x_1 **from** 1 **to** $n-k$ **do**
  for x_2 **from** 1 **to** n **do**
    for $\hat{\delta} \in \mathbb{F}_p \setminus \{0\}$ **do**
      for $\hat{e} \in \mathbb{E}$ **do**
        $\hat{\Delta}[x_1, x_2] = \hat{\delta}$
        $\hat{\mathbf{e}}[x_2] = \hat{e}$
        for i **from** 1 **to** t **do**
          if $\mathrm{chall}_2[i] = 0$ **then**
            $(\mathbf{y}[i], \bar{\mathbf{v}}[i]) \leftarrow \mathrm{resp}[i]_0$
            $\mathbf{v}[i] \leftarrow g^{\bar{\mathbf{v}}[i]}$
            $\mathbf{y}'[i] \leftarrow \mathbf{v}[i] * \mathbf{y}[i]$
            $\mathbf{s}'[i] \leftarrow \mathbf{y}'[i](\mathbf{H} + \hat{\Delta})^{\mathsf{T}} - \mathrm{chall}_1[i](\mathbf{s} + \hat{\mathbf{e}}\hat{\Delta}^{\mathsf{T}})$
            $\mathrm{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \bar{\mathbf{v}}[i] \mid \mathrm{Salt} \mid i + 2t - 1)$
        $\mathrm{digest}_{\mathrm{cmt}_0} \leftarrow \mathsf{RecomputeRoot}(\mathrm{cmt}_0, \mathrm{Proof}, \mathrm{chall}_2)$
        $\mathrm{digest}'_{\mathrm{cmt}} \leftarrow \mathsf{Hash}(\mathrm{digest}_{\mathrm{cmt}_0}, \mathrm{digest}_{\mathrm{cmt}_1})$
        if $\mathrm{digest}_{\mathrm{cmt}} = \mathrm{digest}'_{\mathrm{cmt}}$ **then return** $\hat{\mathbf{e}}$
return $\perp$

Proof. The proof follows from partitioning the problem's two defining linear systems based on the known indices I and unknown indices J. Over $\mathbb{F}_p$, the equation $\mathbf{e}\mathbf{H}^{\mathsf{T}} = \mathbf{s}$ is partitioned into $\mathbf{e}_I\mathbf{H}_I^{\mathsf{T}} + \mathbf{e}_J\mathbf{H}_J^{\mathsf{T}} = \mathbf{s}$. Since $\mathbf{e}_I$ is known, this reduces to a new syndrome decoding problem for the unknown part $\mathbf{e}_J$, namely $\mathbf{e}_J\mathbf{H}_J^{\mathsf{T}} = \mathbf{s} - \mathbf{e}_I\mathbf{H}_I^{\mathsf{T}} =: \mathbf{s}'$, with parameters $n' = n - c$ and $k' = k - c$.

Simultaneously, over $\mathbb{F}_z$, the subgroup constraint $\bar{\mathbf{e}}\bar{\mathbf{H}}^{\mathsf{T}} = 0$ is partitioned into $\bar{\mathbf{e}}_I\bar{\mathbf{H}}_I^{\mathsf{T}} + \bar{\mathbf{e}}_J\bar{\mathbf{H}}_J^{\mathsf{T}} = 0$. This implies $\bar{\mathbf{e}}_J$ must lie in a specific coset of the code defined by the parity-check matrix $\bar{\mathbf{H}}_J$. The dimension of this new code, m', is given as $m' = (n - c) - \mathrm{rank}(\bar{\mathbf{H}}_J)$.

For the R-SDP case where $m = n$, we have $\mathrm{rank}(\bar{\mathbf{H}}_J) = 0$, which correctly simplifies to $m' = n - c = n'$. □

Remark 3. Since CROSS uses a full-rank matrix $\bar{\mathbf{H}} \in \mathbb{F}_z^{(n-m)\times n}$ whose non-systematic part is random and where $z = 127$ is comparatively large, we have $\text{rank}(\bar{\mathbf{H}}_J) = n - m$ with high probability. We thus assume $m' = m - c$ for small values of c in the analysis below.

The best known solvers for the R-SDP and R-SDP(G) problems are described in the security analysis of the CROSS submission [3]. In the interest of brevity, we do not elaborate on the individual approaches here and instead refer to their analysis. The security analysis also includes a finite regime analysis for the cost of these approaches. Following this analysis, we determine the minimum number of entries c that must be recovered by the fault attack to be able to apply the solvers to the reduced instances given by Theorem 2 in order to recover the secret key.

The results for the CROSS-R-SDP 1 parameter set are shown in Figure 1 and other R-SDP parameter sets behave similarly. In particular, the shifted BJMM solver slightly outperforms all other approaches when aiming for an approximate computational complexity of 2^{60} bit operations[3]. For the CROSS-R-SDP(G) parameter sets, the analysis in [3] only considers a Stern solver and we therefore omit further comparison.

Concretely, the shifted BJMM solver and the Stern solver require knowledge of $c = 49$ (R-SDP 1), $c = 84$ (R-SDP 3), $c = 123$ (R-SDP 5), respectively $c = 17$ (R-SDP(G) 1), $c = 31$ (R-SDP(G) 3), and $c = 40$ (R-SDP(G) 5) entries to be able to fully recover $\mathbf{e}$ within a maximum complexity of $\approx 2^{60}$ bit operations. The corresponding erasure rates $r = 1 - c/n$ are non-trivial (> 0.5) for all parameter sets.

We also considered other approaches, including the BKW-style solver by Johansson, Guo, and Nguyen [30] and the algebraic approach by Beullens, Briaud, and Øygarden [6]. However, we estimate that neither approach is likely to significantly outperform the previously mentioned approaches.

6 Attack Probability Analysis under Unknown Bit Flips

In the following, we analyse the attack success probability under unknown bit flips. In particular, we assume uniformly i.i.d. bit flips that affect only the parity-check matrix $\mathbf{H}$, where the attacker has no influence over the location of the bit flips. This is a weak assumption that can be practically realised using many fault injection techniques, such as clock or voltage glitching [33,10,31], electromagnetic fault injection (EMFI) [17,36], laser fault injection (LFI) [1,15], and Rowhammer [37].

Remark 4. Strictly speaking, induced bit flips are unlikely to be uniformly distributed. In fact, Islam et al. [27] found in their attack that certain bits could not

[3] Attacks with computational complexities greater than 2^{60} function evaluations have already been practically demonstrated in the context of SHA-1 [47,34], thus this bound is reasonable for a dedicated attacker.

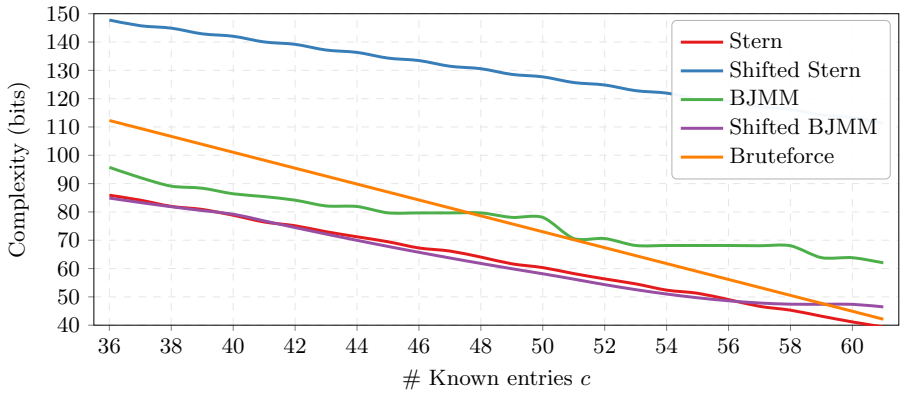


Fig. 1. Approximate bit complexity for state-of-the-art solvers on the CROSS-R-SDP 1 parameter set under c known entries, based on finite regime analysis in [3].

be flipped using their Rowhammer-based setup. In our attack, this is unlikely to be a problem: If none of the bits of a certain entry can be flipped, any of the other $n-k-1$ entries can be targeted instead. Furthermore, if no entry in a row of $\mathbf{H}^\mathsf{T}$ can be targeted, a different row can be targeted instead, as long as c entries are recovered in total.

Theorem 3 (Expected Attempts under Uniform Bit Flips). *Let ρ be the probability of flipping one or more bits of a single entry in $\mathbb{F}_p$ using a fault. The expected number of fault injection attempts required to recover the secret key is given by*

$$E(N) = \frac{n \cdot (H_n - H_{n-c})}{\rho \cdot (1-\rho)^{n(n-k)-1}} \tag{4}$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th harmonic number and c is the number of known entries required to solve the reduced instance.

Proof. This problem is a variant of the *coupon collector's problem* [7, pp. 85-86], which asks for the expected number of samples to collect at least one of each item from a set, with each item occurring with a certain probability. The key insight is that with j-many items already collected, each additional item has probability $(n-j)/n$ of being a new, uncollected item [21]. This yields the expected number of samples:

$$E(N) = n\left(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{1}\right) = nH_n.$$

In our case, it suffices to recover only c entries using faults, since the full vector $\mathbf{e}$ can then be recovered from the reduced instance, as described in Section 5.

Thus, the expected number of successfully faulted matrices is:

$$E(N) = n\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-c+1}\right) = n\left(H_n - H_{n-c}\right).$$

Furthermore, a successful fault injection resulting in a faulty matrix with exactly one changed entry (and $n(n-k)-1$ unchanged entries) is expected to require

$$\frac{1}{\rho \cdot (1-\rho)^{n(n-k)-1}}$$

attempts. Combining these two results yields Equation (4). □

Remark 5. We deliberately state the above result in terms of the probability ρ of one or more bits of an entry in $\mathbb{F}_p$ being flipped, rather than in terms of the probability of individual bit flips in the entire matrix. While the R-SDP and R-SDP(G) parameter sets have $p = 127$ and $p = 509$ respectively, and thus require 7 or 9 bits to represent an entry, the reference implementation [4] actually stores elements as `uint8_t` (R-SDP) or `uint16_t` (R-SDP(G)) types. Flipping any of the 8 or 16 bits of the entry can be used to perform the attack, as the unused bits still affect modular reductions. The reference implementation further stores only the non-systematic part of $\mathbf{H}$, which increases the probability of a faulty matrix having the correct structure, but decreases the probability of each successfully faulted matrix having a fault in a new row. In practice, the minimum expected number of attempts required to generate sufficiently many faulty signatures to recover the secret key decreases slightly (by between 10–32% depending on the parameter set), but we provide only the more representative results for the full matrix here.

Table 3. Expected number of fault injection attempts required to successfully recover the key under uniformly random bit flips, as computed from Equation (4).

Parameter set	p	z	n	k	c	$E(N)$	ρ
CROSS-R-SDP 1	127	7	127	76	49	1,084,423	1.536e-4
CROSS-R-SDP 3	127	7	187	111	84	4,292,825	6.952e-5
CROSS-R-SDP 5	127	7	251	150	123	11,615,749	3.997e-5
CROSS-R-SDP(G) 1	509	127	55	36	17	57,109	9.562e-4
CROSS-R-SDP(G) 3	509	127	79	48	31	259,842	4.077e-4
CROSS-R-SDP(G) 5	509	127	106	69	40	532,129	2.544e-4

Using Equation (4), we can determine the minimum expected number of attempts under unknown uniform bit flips, which is shown in Table 3. The probabilities ρ for one or more bits of an entry in $\mathbb{F}_p$ being flipped must be high enough for at least one entry to be affected, without being too high to affect multiple entries. In practice, achieving these values may be difficult, but is generally also not necessary, as the distribution of bit flips is likely only locally uniform, which simplifies obtaining a matrix with the correct structure.

7 Practical Evaluation

This section describes the practical evaluation of the presented attack under the instruction skipping fault model, induced through voltage fault injection.

7.1 Experimental Setup

In the experiments, we use a CW308-STM32F4 target board with an ARM Cortex-M4 STM32F415RGT6 processor running at 24 MHz. The target board is mounted on a CW308 UFO board, and voltage fault injection is performed using a ChipWhisperer-Husky.

We target the second-round reference implementation of CROSS [4], compiled with `arm-none-eabi-gcc` at optimisation level `-O3`. The target device has only 128KB of SRAM available, which is insufficient to fit the implementation. We thus make a minor modification to explicitly store three large matrices and one vector in the core-coupled memory (CCM), an additional 64KB block of memory that is accessible only to the processor and is mapped to a different memory region but functions like SRAM. Note that this modification has no effect on the section of code under attack.

We target the CROSS-R-SDP 1 fast parameter set, to ease experimental evaluation by reducing runtime and memory usage. The attack applies directly to all other parameter sets and variants, including the CROSS-R-SDP(G) parameter sets.

7.2 Faulting the Derivation

Our attack targets the `csprng_fp_mat` procedure invoked within the `expand_pk` function at the beginning of the signing algorithm. This procedure is responsible for pseudorandomly deriving the parity-check matrix $\mathbf{H}$ (or, more accurately, its non-systematic part) from the public key using rejection sampling. Note that, for efficiency, the reference implementation samples the non-systematic part in transposed form and does not store the systematic part explicitly.

The procedure is shown in Listing 1. It works as follows: First, a large buffer, `prng_buf`, is initialised using the pseudorandom generator (PRG). The size of this buffer is chosen such that the probability of the rejection sampling to fail is negligible. The procedure then maintains a 64-bit sub-buffer, from which blocks of 7 (R-SDP) or 9 (R-SDP(G)) bits are extracted at a time. The sub-buffer is refreshed using a refresh buffer initialised from the PRG output whenever fewer than 32 bits remain in the sub-buffer. For each block extracted from the sub-buffer, the procedure checks if the corresponding value is smaller than p and, if so, places it at the output. Otherwise, the element is rejected and another attempt is made. This process is repeated until all $k \times (n-k)$ entries of the transposed non-systematic part of the parity-check matrix $\mathbf{H}$ have been generated.

Faults can be injected into this procedure to cause incorrect output in several ways. We consider, at a high level, several scenarios in which skipping a step in the `csprng_fp_mat` procedure leads to a faulty parity-check matrix. Note that, in

contrast to other instruction skipping attacks, we are not constrained to skipping only one specific instruction.

The first category of possible faults that lead to faulty output either prematurely force the sub-buffer to be refreshed or (partially) prevent refreshing or updating of it. Such faults can occur through instruction skips in the conditionals in lines 13 and 14 of Listing 1, by aborting the loop in line 16, or by interfering with the tracking of the number of elements in the buffer, sub-buffer, or refresh buffer in lines 19–22 and line 29. They can also occur by skipping the shifting in line 28.

The second category of possible faults concerns the extraction, rejection, and placing of entries. By skipping (part of) line 24 or by skipping the condition in line 25, an incorrect entry may be placed. Recall that elements are stored as `uint8_t` or `uint16_t` types despite being only 7 or 9 bits long though the unused bits still affect modular multiplication operations and can thus be used for the attack.

Whether any of the above-mentioned steps can be practically skipped depends on the generated assembly code and precision of the fault injection equipment. Whether successfully skipping a step leads to a faulty parity-check matrix of the correct structure (i.e., with exactly one changed entry) further depends on the internal state of the procedure. In the next section, we demonstrate that the `csprng_fp_mat` procedure is indeed exploitable in practice.

7.3 Results

We evaluate the attack using the experimental setup described in Section 7.1. Since the rejection sampling takes different lengths of time depending on the generated sequence of elements (and thus the public key), we fix the key in our experiments. Note that this is also the case for a real attack. Also note that an attacker can precisely determine the number of rejection sampling steps from the public key. The procedure takes approximately 87,907 clock cycles, and we perform one fault injection attempt for each clock cycle.

We identify 996 clock cycles where inserting a fault causes a faulty parity-check matrix $\tilde{\mathbf{H}}$ with exactly one changed entry. The actual number of vulnerable clock cycles may be higher, as we perform only a single attempt and therefore do not account for unreliable faults. This estimation provides more representative results. We further identify 305 clock cycles where inserting a fault causes two changed entries. In only 9 of these cases are both faults located in different rows of $\tilde{\mathbf{H}}^\mathsf{T}$. For the sake of simplicity, we thus focus exclusively on the first case.

The 87,907 clock cycles correspond to approximately 3.663 ms of execution time on the (comparatively slow) 24 MHz experimental platform. An attacker capable of inserting a fault at any point during this window therefore has an approximately 1.13% chance of obtaining a faulty parity-check matrix.

In Section 6 we analysed the expected number of fault injection attempts required to recover the secret key under uniformly i.i.d. bit flips. We use the same approach here, but now substitute the above-derived probability instead and account for the fact that only the non-systematic part of $\mathbf{H}$ is stored. Then

```
void csprng_fp_mat(FP_ELEM res[K][N-K], void* prng_state) {
    uint8_t prng_buf[CSPRNG_MAT_BUF_LEN];
    csprng_randombytes(prng_buf,prng_state);
    int placed = 0;
    uint64_t sub_buffer = 0;
    for (int i=0; i<8; i++) {
        sub_buffer |= ((uint64_t)prng_buf[i]) << 8*i;
    }
    int in_sub_buf = 64;
    int in_buf = 8;
    int remaining = sizeof(prng_buf) - in_buf;
    while (placed < K*(N-K)) {
        if (in_sub_buf <= 32 && remaining > 0) {
            int refresh_amount = (remaining >= 4) ? 4 : remaining;
            uint32_t refresh_buf = 0;
            for (int i=0; i < refresh_amount; i++) {
                refresh_buf |= ((uint32_t)prng_buf[in_buf+i]) << 8*i;
            }
            in_buf += refresh_amount;
            sub_buffer |=  ((uint64_t)refresh_buf) << in_sub_buf;
            in_sub_buf += 8*refresh_amount;
            remaining -= refresh_amount;
        }
        *((FP_ELEM*)res+placed) = sub_buffer & MASK_FOR_P;
        if (*((FP_ELEM*)res+placed) < P) {
            placed++;
        }
        sub_buffer = sub_buffer >> BITS_FOR_P;
        in_sub_buf -= BITS_FOR_P;
    }
}
```

Listing 1: C code of the `csprng_fp_mat` procedure.

the expected number of fault injection attempts (for the CROSS-R-SDP 1 fast parameter set) is:

$$E(N) = \frac{76 \cdot (H_{76} - H_{76-49})}{0.0113} \approx 6862.$$

In practice, this number may be lower since it is likely that an attacker would have at least some control over the timing of their fault and that faults would thus not be uniformly distributed across the runtime of the procedure. Since the entries of $\mathbf{H}$ are sampled sequentially, even imprecise control could be used to ensure that attempts target different rows, thereby reducing the overall number of attempts. Assuming only single-entry faults in $\tilde{\mathbf{H}}$, the minimum number of required attempts is $c = 49$, though this requires much greater timing control

and perfectly reliable fault injection, which are unlikely to be achievable in a realistic setting.

8 Comparison to Correction Fault Attacks on Dilithium

The presented attack makes use of CROSS's structure to allow it to work even with weak attacker capabilities. In this section, we elaborate on the differences between our approach and previous correction-based attacks on Dilithium [27,32].

The two main differences between the attack by Islam et al. [27] and the presented attack are that the attack [27] 1) targets a secret component of the scheme and 2) assumes that only single bit flips occur. As pointed out in [32], secret components are more obvious targets for attacks and thus more likely to be protected by countermeasures. Furthermore, while the authors of [27] experimentally demonstrate that single bit flips can be induced using Rowhammer, this may not be the case for less precise fault injection techniques, such as voltage glitching or EMFI, where several bits may be flipped at a time. Since they do not further comment on the effect of a multi-bit flip on their attack, it is unclear if the attack could be conducted using such techniques at all (either through a different key recovery algorithm or by detecting and filtering multi-bit flips). As a consequence, as discussed in Section 9.1, the attack [27] is therefore not applicable in the presence of single-error correcting codes.

The first attack of Krahmer et al. [32] is only applicable under precise instruction skipping, therefore we do not consider it further here. The main difference between the second attack [27] and our attack is that they assume knowledge of the position and value of the fault, and therefore modify their target implementation to zero-initialise the matrix $\mathbf{A}$ in their experiments. This severely limits the practicality of their attack under imprecise fault injection techniques. To address this limitation, they suggest repeating the attack and checking for consistency between attempts. However, beyond increasing the number of required attempts for a successful attack, such repetition may be difficult to realise with imprecise faults. Fundamentally, due to Dilithium's structure, their approach cannot determine whether a secret value has been correctly recovered using a single signature. Our approach instead leverages CROSS's multi-round structure such that an incorrectly recovered secret value can only occur with negligible probability, even when a single signature is considered.

9 Countermeasures and their Effectiveness

In this section we propose countermeasures for the presented fault attack and discuss their effectiveness. We also describe variants of the attack capable of bypassing certain countermeasures.

9.1 Hardware-level Countermeasures

Error-correcting codes. Modern DDR5 devices contain on-die error-correcting codes (ECC) capable of correcting single bit flips [28]. The error correction works

by computing the syndrome of a codeword using a parity-check matrix[4]. If the syndrome is non-zero and matches a column of this matrix, the error can be corrected by flipping the codeword's corresponding bit. If the syndrome is non-zero but does not match any column, then a multi-bit error is detected but not corrected. The benefit of the on-die ECC is that it is automatically used for every read of $\mathbf{H}$, which is necessary to prevent an attacker from faulting $\mathbf{H}$ only after it has been verified.

It is therefore reasonable to consider whether such an ECC can be used to protect against the presented attack. However, this is not the case: With a single-error ECC like the one used for DDR5, it is possible to inject multi-bit errors that are neither detected nor corrected. Since the attack only requires an entry of $\mathbf{H}$ to be faulty, but has no requirement on how many bits of this entry are flipped, the attack is still applicable in this case. Using a more powerful ECC capable of detecting 7- or 9-bit errors would effectively prevent multi-bit error injection into a single entry from being used, but may be infeasible to implement in current hardware.

Remark 6. The attack by Mondal et al. [35] requires a precise bit flip in the reference tree used to determine which seeds to reveal. Since the reference implementation [4] explicitly checks for a value of 1 for releasing a seed (as opposed to checking for any non-zero value), this requires a single bit flip $0 \rightarrow 1$. Under a single-error ECC, the attack is therefore not possible, as all single-bit errors are corrected, and all multi-bit errors result in more than one flipped bit after correction. This also applies to the attack of Islam et al. [27], which similarly requires single-bit flips and thus cannot be conducted in the presence of single-error ECCs.

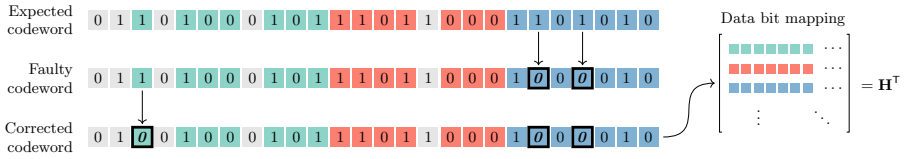


Fig. 2. Simplified illustration of a $(26, 21)$-ECC protecting three 7-bit entries of $\mathbf{H}$. Bits of the same entry are colour-highlighted and parity bits are gray. A two-bit fault injected into the last entry causes an erroneous single-bit correction in the first entry. Since the two faulty entries map into different rows in $\mathbf{H}^{\mathsf{T}}$, two entries of $\mathbf{e}$ can be recovered by the attack.

Remark 7. Depending on how the parity-check matrix is stored and how the ECC is implemented, the presented attack may require fewer successful faults

[4] Note that the parity-check matrix here is *not* related to the parity-check matrix $\mathbf{H}$ considered in the attack.

than without ECC. Consider the case where $\mathbf{H}$ is stored in non-transposed row-major order (i.e., adjacent elements belong to the same row, but different columns) and that a similar ECC as described earlier is used, namely that the ECC is applied across multiple adjacent entries. Some multi-bit faults can lead to correction of bits that were not faulty and thereby cause two faulty entries in the same row of $\tilde{\mathbf{H}}$. When $\tilde{\mathbf{H}}$ is subsequently transposed, this allows recovering multiple different entries of $\mathbf{e}$ from a single faulty signature. The same logic applies if the parity-check matrix is stored in transposed column-major order. A simplified illustration of such a case is shown in Figure 2, where the correction causes an additional bit in the first entry to be flipped, thus leading to two faulty entries in different rows of $\tilde{\mathbf{H}}^{\mathsf{T}}$. This problem can be prevented by using an ECC whose codewords cover only a single entry or by using codes only for detection, not correction.

Fault detection sensors. A different approach is to use sensors to detect the fault injection attempts and abort the computation upon detection, thereby preventing an attacker from learning a faulty signature. Many such sensors have been proposed in the past, including for protection against clock/voltage glitching, EMFI, and LFI [52,18,50,25,38]. While they require physical implementation and thus cannot be retrofitted to existing devices, these sensors could effectively prevent the presented attack. A potential issue arises when the parity-check matrix is stored in expanded form in flash memory, in which case it might be possible to circumvent fault detectors by performing an attack while the device is switched off [24].

9.2 Algorithm-level Countermeasures

Generic SCA countermeasures. Shuffling and masking [13] are commonly used countermeasures for passive side-channel attacks. While they are not primarily intended to protect against fault injection attacks, some attacks become infeasible in their presence.

Shuffling refers to a random reordering of operations in time, which makes it difficult to identify when a device is computing a sensitive value. This approach is not applicable to the computation of $\mathbf{H}$, as the rejection sampling used to compute the individual entries has to be performed sequentially. Furthermore, the presented attack does not rely on precise timing of the fault, so approaches that simply randomise the order or timing of computations would increase the number of required attempts, but not prevent the attack.

Masking refers to the splitting of sensitive information into two or more randomised shares that can be combined to recover the original value. Computations are then performed on the individual shares and leakage of one share does not reveal the sensitive value without knowledge of the other shares. Simple masking schemes, such as Boolean masking, would not prevent the attack as flipping a bit in one of the shares would cause a bit flip in the combined entry. More sophisticated code-based masking schemes [49] might be able to detect bit flips. As described previously, our attack is still applicable under certain ECCs,

thus whether such an approach would be effective depends on the chosen masking scheme. Additionally, the computational overhead of masking, especially for complex masking schemes, makes it impractical to apply to $\mathbf{H}$, which contains only public information.

Sign-then-verify. A straightforward countermeasure to any correction fault attack is to perform verification of the signature after it has been generated and to withhold any signatures that do not pass verification. Since correction-based attacks depend on correcting the faulty signature to extract secret information, they are effectively prevented by this approach. While it may be possible to skip (part of) the verification using a precise fault and thereby circumvent this countermeasure, such a higher-order attack would be significantly more difficult in practice. However, performing verification during each signature generation may cause an unacceptable increase in computational overhead. Based on the performance analysis provided in the CROSS specification [2], such an approach would increase runtime by 57–73%, which may be especially problematic on slower embedded devices that are more likely to be targeted by fault attacks.

Duplication-and-comparison. Similar to the sign-then-verify countermeasure, it is also possible to perform signature generation twice and to only release the signature if it is the same in both attempts. Such a countermeasure would be effective under the assumption that an attacker is not able to precisely inject the same fault into two attempts, and that the parity-check matrix is not reused between attempts. However, it increases the computational costs for the signature generation (though it may be possible to run both attempts in parallel, so the increase in runtime may be small). As before, such an increase may be problematic especially on embedded platforms. Depending on its implementation, the duplication may also make it easier to perform side-channel analysis, as it provides two distinct executions with the same data within a single signature generation [51]. Such a countermeasure therefore requires careful consideration before implementation.

Remark 8. It is possible to adapt the presented attack to work in the presence of the sign-then-verify and duplication-and-comparison countermeasures, albeit at a lower probability and with stronger attacker assumptions, by using ineffective faults [14]. The idea is to fault the parity-check matrix during the computation of a single per-round syndrome $\mathbf{s}'[i]$ in the signing procedure, such that the faulty matrix $\hat{\mathbf{H}}$ differs in exactly one entry whose position is known. If the resulting signature passes verification, then the corresponding entry of $\mathbf{u}'[i]$ must be zero, which reveals an entry of the secret key from the response $\mathbf{y}[i]$. In practice, such an attack may be difficult to realise, as $\mathbf{u}'$ is uniformly and randomly distributed over $\mathbb{F}_p^n$ and any one entry is thus only zero with probability p^{-1}. Furthermore, the required fault is very precise and can only affect a single iteration and must affect a known position. The approach also does not offer a straightforward way of detecting failed fault injection attempts and distinguishing them from ineffective faults, though it would be possible to repeat the attack and check for consistency between attempts.

10 Conclusion

This paper presented a highly effective correction-based fault attack on CROSS capable of recovering the secret key. The attack targets the public parity-check matrix $\mathbf{H}$, which has not been considered in previous physical attacks. $\mathbf{H}$ is unlikely to be protected by existing countermeasures, since it does not contain sensitive information. The attack is applicable under both the arbitrary bit flip and the instruction skipping fault models. It does not require precise control over or knowledge of the fault's value or position, and can be adapted to work in the presence of countermeasures. Our results highlight the need to protect CROSS's public parity-check matrix from physical attacks.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Swedish Civil Contingencies Agency (Grant No. 2020-11632), Sweden's Innovation Agency Vinnova (Grant No. 2025-00475), and by the Swedish Research Council (Grant No. 2021-04602 and 2023-03654).

References

1. Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: 2010 IEEE 16th International On-Line Testing Symposium. pp. 235–239 (2010). https://doi.org/10.1109/IOLTS.2010.5560194
2. Baldi, M., Barenghi, A., Battagliola, M., Bitzer, S., Gianvecchio, M., Karl, P., Manganiello, F., Pavoni, A., Pelosi, G., Pintore, F., Santini, P., Schupp, J., Signorini, E., Slaughter, F., Wachter-Zeh, A., Weger, V.: CROSS. https://www.cross-crypto.com/CROSS_Specification_v2.2.pdf (July 2025)
3. Baldi, M., Barenghi, A., Battagliola, M., Bitzer, S., Gianvecchio, M., Karl, P., Manganiello, F., Pavoni, A., Pelosi, G., Pintore, F., Santini, P., Schupp, J., Signorini, E., Slaughter, F., Wachter-Zeh, A., Weger, V.: CROSS. https://www.cross-crypto.com/CROSS_SecurityDetails_v2.2.pdf (July 2025)
4. Baldi, M., Barenghi, A., Battagliola, M., Bitzer, S., Gianvecchio, M., Karl, P., Manganiello, F., Pavoni, A., Pelosi, G., Pintore, F., Santini, P., Schupp, J., Signorini, E., Slaughter, F., Wachter-Zeh, A., Weger, V.: CROSS_Implementation. https://github.com/CROSS-signature/CROSS-implementation (July 2025)
5. Bauer, S., De Santis, F.: Forging Dilithium and Falcon signatures by single fault injection. In: 2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 81–88 (2023). https://doi.org/10.1109/FDTC60478.2023.00017
6. Beullens, W., Briaud, P., Øygarden, M.: A security analysis of restricted syndrome decoding problems. IACR Commun. Cryptol. **1**(3), 33 (2024). https://doi.org/10.62056/A06CY7QIU, https://doi.org/10.62056/a06cy7qiu
7. Blom, G., Holst, L., Sandell, D.: Problems and Snapshots from the World of Probability. Springer New York, New York, NY, 1st ed. 1994. edn. (1994). https://doi.org/10.1007/978-1-4612-4304-5

8. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of cryptology **14**(2), 101–119 (2001)
9. Boy, J., Purnal, A., Pätschke, A., Wilke, L., Eisenbarth, T.: SLasH-DSA: Breaking SLH-DSA using an extensible end-to-end Rowhammer framework (2025), https://arxiv.org/abs/2509.13048
10. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 199–224 (Feb 2019). https://doi.org/10.13154/tches.v2019.i2.199-224
11. Calle Viera, A., Berzati, A., Heydemann, K.: Fault attacks sensitivity of public parameters in the Dilithium verification. In: Bhasin, S., Roche, T. (eds.) Smart Card Research and Advanced Applications. pp. 62–83. Springer Nature Switzerland, Cham (2024)
12. Castelnovi, L., Martinelli, A., Prest, T.: Grafting trees: A fault attack against the SPHINCS framework. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 165–184. Springer International Publishing, Cham (2018)
13. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - Proceedings of CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26
14. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007. pp. 181–194. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
15. Colombier, B., Grandamme, P., Vernay, J., Chanavat, É., Bossuet, L., de Laulanié, L., Chassagne, B.: Multi-spot laser fault injection setup: New possibilities for fault injection attacks. In: Grosso, V., Pöppelmann, T. (eds.) Smart Card Research and Advanced Applications. pp. 151–166. Springer International Publishing, Cham (2022)
16. Czuprynko, M., Mukherjee, A., Roy, S.S.: Correlation Power Analysis of LESS and CROSS. In: Nitaj, A., Petkova-Nikova, S., Rijmen, V. (eds.) Progress in Cryptology - AFRICACRYPT 2025. pp. 270–295. Springer Nature Switzerland, Cham (2026)
17. Dumont, M., Lisart, M., Maurine, P.: Electromagnetic fault injection : How faults occur. In: 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 9–16 (2019). https://doi.org/10.1109/FDTC.2019.00010
18. El-Baze, D., Rigaud, J.B., Maurine, P.: An embedded digital sensor against em and bb fault injection. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 78–86 (2016). https://doi.org/10.1109/FDTC.2016.14
19. ElGhamrawy, M., Azouaoui, M., Bronchain, O., Renes, J., Schneider, T., Schönauer, M., Seker, O., van Vredendaal, C.: From MLWE to RLWE: A differential fault attack on randomized & deterministic Dilithium. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 262–286 (Aug 2023). https://doi.org/10.46586/tches.v2023.i4.262-286
20. Esser, A., May, A., Verbel, J., Wen, W.: Partial key exposure attacks on BIKE, Rainbow and NTRU. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 346–375. Springer Nature Switzerland, Cham (2022)
21. Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discrete Applied Mathematics **39**(3), 207–229 (1992). https://doi.org/10.1016/0166-218X(92)90177-C

22. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON: Fast-fourier lattice-based compact signatures over NTRU. https://falcon-sign.info/falcon.pdf (October 2020)
23. Gaborit, P., Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Véron, P., Barreto, P.L., Ghosh, S., Gueron, S., Güneysu, T., Misoczki, R., Richter-Brokmann, J., Sendrier, N., Tillich, J.P., Vasseur, V.: Hamming Quasi-Cyclic (HQC). https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf (August 2025)
24. Grandamme, P., Tissot, P.A., Bossuet, L., Dutertre, J.M., Colombier, B., Grosso, V.: Switching off your device does not protect against fault attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(4), 425–450 (Sep 2024). https://doi.org/10.46586/tches.v2024.i4.425-450
25. He, W., Breier, J., Bhasin, S.: Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 27–46. Springer International Publishing, Cham (2016)
26. Heninger, N., Shacham, H.: Reconstructing RSA private keys from random key bits. In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009. pp. 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
27. Islam, S., Mus, K., Singh, R., Schaumont, P., Sunar, B.: Signature correction attack on Dilithium signature scheme. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). pp. 647–663 (2022). https://doi.org/10.1109/EuroSP53844.2022.00046
28. JEDEC: DDR5 SDRAM standard, JESD79-5. Tech. Rep. JESD79-5C, JEDEC Solid State Technology Association, Arlington, VA (July 2024), https://www.jedec.org/standards-documents/docs/jesd79-5c01
29. Jendral, S., Mattsson, J.P., Dubrova, E.: A single-trace fault injection attack on hedged module lattice digital signature algorithm (ML-DSA). In: 2024 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 34–43 (2024). https://doi.org/10.1109/FDTC64268.2024.00013
30. Johansson, T., Guo, Q., Nguyen, V.: A BKW-Style solver for the restricted syndrome decoding problem. In: International Workshop on Code-Based Cryptography (CBCC) 2025. Madrid, Spain (May 2025), proceedings of the International Workshop on Code-Based Cryptography 2025
31. Kazemi, Z., Papadimitriou, A., Souvatzoglou, I., Aerabi, E., Ahmed, M.M., Hely, D., Beroulle, V.: On a low cost fault injection framework for security assessment of cyber-physical systems: Clock glitch attacks. In: 2019 IEEE 4th International Verification and Security Workshop (IVSW). pp. 7–12 (2019). https://doi.org/10.1109/IVSW.2019.8854391
32. Krahmer, E., Pessl, P., Land, G., Güneysu, T.: Correction fault attacks on randomized CRYSTALS-Dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(3), 174–199 (2024). https://doi.org/10.46586/TCHES.V2024.I3.174-199
33. Kumar, R., Jovanovic, P., Polian, I.: Precise fault-injections using voltage and temperature manipulation for differential cryptanalysis. In: 2014 IEEE 20th International On-Line Testing Symposium (IOLTS). pp. 43–48 (2014). https://doi.org/10.1109/IOLTS.2014.6873670
34. Leurent, G., Peyrin, T.: SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust. In: Capkun, S., Roesner, F. (eds.) 29th

USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020. pp. 1839–1856. USENIX Association (2020), https://www.usenix.org/conference/usenixsecurity20/presentation/leurent
35. Mondal, P., Adhikary, S., Kundu, S., Karmakar, A.: ZKFault: Fault attack analysis on zero-knowledge based post-quantum digital signature schemes. In: Chung, K., Sasaki, Y. (eds.) Advances in Cryptology - Proceedings of ASIACRYPT'2024. Lecture Notes in Computer Science, vol. 15491, pp. 132–167. Springer (2024). https://doi.org/10.1007/978-981-96-0944-4_5
36. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In: 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 77–88 (2013). https://doi.org/10.1109/FDTC.2013.9
37. Mutlu, O., Kim, J.S.: Rowhammer: A retrospective. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(8), 1555–1571 (2020). https://doi.org/10.1109/TCAD.2019.2915318
38. Muttaki, M.R., Zhang, T., Tehranipoor, M., Farahmandi, F.: FTC: A universal sensor for fault injection attack detection. In: 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 117–120 (2022). https://doi.org/10.1109/HOST54066.2022.9840177
39. National Institute of Standards and Technology: NIST announces additional digital signature candidates for the PQC standardization process. https://csrc.nist.gov/news/2023/additional-pqc-digital-signature-candidates (June 2023)
40. National Institute of Standards and Technology: Module-Lattice-Based Digital Signature Standard. Tech. Rep. NIST FIPS 204, National Institute of Standards and Technology, Gaithersburg, MD (August 2024). https://doi.org/10.6028/NIST.FIPS.204
41. National Institute of Standards and Technology: Module-Lattice-Based Key Encapsulation Mechanism Standard. Tech. Rep. NIST FIPS 203, National Institute of Standards and Technology, Gaithersburg, MD (August 2024). https://doi.org/10.6028/NIST.FIPS.203
42. National Institute of Standards and Technology: NIST announces 14 candidates to advance to the second round of the additional digital signatures for the post-quantum cryptography standardization process. https://csrc.nist.gov/news/2024/pqc-digital-signature-second-round-announcement (October 2024)
43. National Institute of Standards and Technology: Stateless Hash-Based Digital Signature Standard. Tech. Rep. NIST FIPS 205, National Institute of Standards and Technology, Gaithersburg, MD (August 2024). https://doi.org/10.6028/NIST.FIPS.205
44. National Institute of Standards and Technology: Status report on the first round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. Tech. Rep. NIST IR 8528, National Institute of Standards and Technology, Gaithersburg, MD (October 2024). https://doi.org/10.6028/NIST.IR.8528
45. Ravi, P., Roy, D.B., Bhasin, S., Chattopadhyay, A., Mukhopadhyay, D.: Number "not used" once - practical fault attack on pqm4 implementations of NIST candidates. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 232–250. Springer International Publishing, Cham (2019)
46. Schupp, J., Sigl, G.: A horizontal attack on the codes and restricted objects signature scheme (CROSS). Cryptology ePrint Archive, Paper 2025/116 (2025), https://eprint.iacr.org/2025/116

47. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 570–596. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_19
48. Wagner, A., Wesselkamp, V., Oberhansl, F., Schink, M., Strieder, E.: Faulting Winternitz One-Time Signatures to forge LMS, XMSS, or SPHINCS+ signatures. In: Johansson, T., Smith-Tone, D. (eds.) Post-Quantum Cryptography. pp. 658–687. Springer Nature Switzerland, Cham (2023)
49. Wang, W., Méaux, P., Cassiers, G., Standaert, F.X.: Efficient and private computations with code-based masking. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(2), 128–171 (Mar 2020). https://doi.org/10.13154/tches.v2020.i2.128-171
50. Yanci, A.G., Pickles, S., Arslan, T.: Detecting voltage glitch attacks on secure devices. In: 2008 Bio-inspired, Learning and Intelligent Systems for Security. pp. 75–80 (2008). https://doi.org/10.1109/BLISS.2008.26
51. Yu, Y., Marranghello, F., Teijeira, V.D., Dubrova, E.: One-sided countermeasures for side-channel attacks can backfire. In: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks. p. 299–301. WiSec '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3212480.3226104
52. Zussa, L., Dehbaoui, A., Tobich, K., Dutertre, J.M., Maurine, P., Guillaume-Sage, L., Clediere, J., Tria, A.: Efficiency of a glitch detector against electromagnetic fault injection. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6 (2014). https://doi.org/10.7873/DATE.2014.216