

Revisiting Correction Attack on CROSS: Improved Attack Method through Analyzing Power Traces

Xiao Huang¹ Chao Sun², Mehdi Tibouchi³

¹Southeast University, China

²East China Normal University, China

³NTT Social Informatics Laboratories, Japan

October 11, 2026



Background Information about CROSS

Scheme Basics

- CROSS signing uses a secret error vector $\mathbf{e}$ and a public parity-check matrix $H = [-V^T \mid I_{n-k}]$.
- Private Key: $\mathbf{e} \in \{1, g, g^2, \dots, g^{z-1}\}^n$.
- Public Key: $\mathbf{s} = \mathbf{e}H^T$.
- V^T is expanded from the public seed Seed_{pk} , one entry at a time.

Attacks on CROSS

Side-Channel Attacks

- Horizontal attack.
- Correlation power analysis.

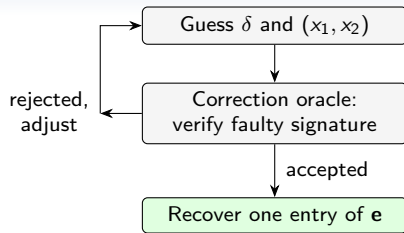
Fault Attacks

- Fault attack on the GGM-tree structure
- Correction fault attack on the public matrix H

Our focus: improving the existing correction fault attack by Jendral et al.

Baseline: The Correction Attack by Jendral *et al.*

- Inject a random fault into one entry of V^T to get a faulty signature; the verifier then uses the faulty matrix $\bar{H} = H + \Delta$.
- Verification serves as a correction oracle: the attacker guesses the faulty index, fault change δ and fault location (x_1, x_2) offline, then checks via verification.
- An accepted verification reveals one entry of the secret vector e_{x_1} .



The attack works in principle, but random, blind fault injection is the practical bottleneck

Bottleneck of the Baseline Method

Two challenges of random fault injection:

1. Random fault timing

Unpredictable timing requires many attack attempts to recover the full key.

2. Unknown faulty index

The unknown fault location leads to a large offline enumeration space.

Cost of full key recovery for R-SDP 1:

$< 2\%$

Success rate

$> 10^6$

Attack attempts

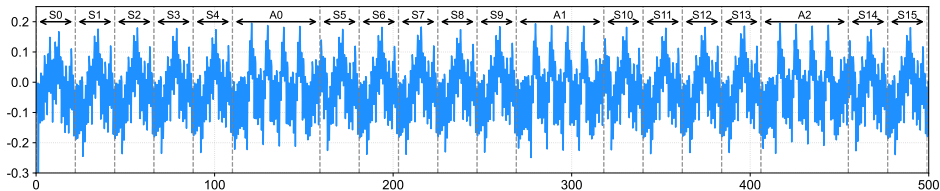
2^{41}

Offline correction-oracle
executions

Key question: Can we reduce attack costs?

Regularity in Key Expansion (1)

- The expansion of V^T follows a highly **regular execution pattern**.
- Profile a trace $\Rightarrow$ locate a chosen entry's window and build a **clock-cycle** $\leftrightarrow$ **matrix-index** map.
- A chosen entry can then be hit **directly**.



S: (S_0 – S_{15}): one timing window per matrix entry A : buffer refill

We turn random fault injection into targeted fault injection

Regularity in Key Expansion (2)

Before sampling:

Public-key seed $\longrightarrow$ 8-bit integer array $\longrightarrow$ 64-bit sub-buffer $\longrightarrow$ Sampling loop

Three stages in each sampling loop:

S1. Buffer Refill

Refill with 32 bits into
buffer.

Conditional

S2. Extract & Write

Extract 7 or 9 bits to
assign entries in V .

Every iteration

S3. Range Check

Accept the value and
move to next index.

Conditional

Key Insight: The structured sampling loop makes matrix-entry timing predictable.

Fault Model and Attacker Capabilities

Target Device

- Fixed secret key across executions
- Intermediate random values refreshed for each execution

Attacker

- Access to the target device; collect faulty signatures
- Access to a profiling device without the secret key

Successful Attack: Exactly one entry of V^T is corrupted
(zero corrupted entries or two or more corrupted entries considered as failure)

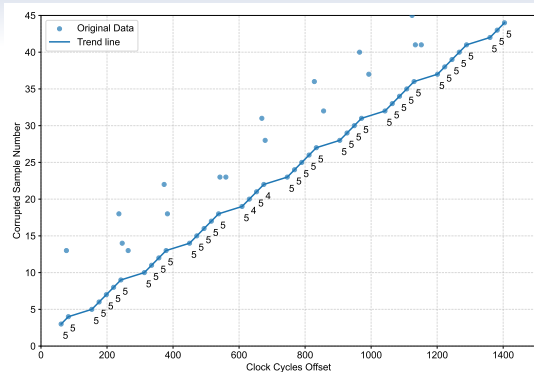
The profiling device is used to test fault injection and learn the timing pattern before attacking the real target.

Determine Index from Timing

- A successful fault at entry i^* serves as a timing reference.
- To target a later entry, account for the time spent on:
 - Buffer refills (t_1)
 - Sampling-loop iterations (t_2)
 - Pointer updates (t_3)
- The timing of entry $i^* + d$ is:

$$T(i^* + d) = T(i^*) + c_{\text{refill}} t_1 + c_{\text{sample}} t_2 + d t_3.$$

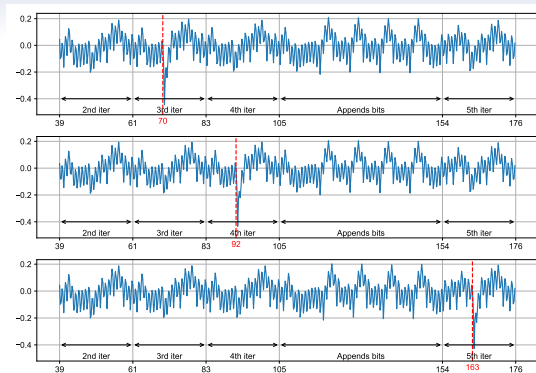
- Select the corresponding injection offset to target the desired entry directly.



Compute the timing offset directly to target the desired entry without trial injections.

Extending One Calibrated Glitch

- Platform: **STM32F303 + ChipWhisperer-Lite**.
- Buffer refill takes **49 cycles**; sampling takes **22 cycles**.
- Calibrate the glitch to corrupt entry 3 at an offset of **70 cycles**.
- Reuse the same glitch, changing only the timing:
 - Entry 4: shift by +22 cycles (offset: 92 cycles).
 - Entry 5: shift by another +22 + 49 cycles (offset: 163 cycles).
- One calibrated glitch can target multiple entries.



Results across All Parameter Sets

- For all CROSS parameter sets: R-SDP $> 98\%$; R-SDPG $> 40\%$ success rate.
- Full key recovery: less than **300 fault injections** per set after profiling.
- Including profiling, about **50k trials** are required (precision: 0.01 clock cycle)

Table: Attack Results across All Six CROSS Parameter Sets.

Parameter Set	Tested Keys	Fault Trials			Success Rate (%)
		Min	Max	Avg	
R-SDP 1 ($k = 76$)	135	76	80	76.77	98.997
R-SDP 3 ($k = 111$)	135	111	118	112.48	98.691
R-SDP 5 ($k = 150$)	135	150	155	151.50	99.008
R-SDPG 1 ($k = 36$)	135	36	87	51.55	69.923
R-SDPG 3 ($k = 48$)	135	50	80	61.85	77.747
R-SDPG 5 ($k = 69$)	135	108	203	147.80	46.728

Offline Complexity: Comparison

- Original: unknown index $\Rightarrow$ enumerate matrix positions.
- Ours: timing identifies the index $\Rightarrow$ no enumeration.
- Single-entry complexity: $n(n - k)z(p - 1) \rightarrow z(p - 1)$.

Table: Comparison of Signature Verification Invocations for Private Key Recovery.

Security	Parameters				Ours		JDGJ25	
	p	z	n	k	One	All(40%)	One	All
R-SDP 1	127	7	127	76	$2^{9.78}$	$2^{16.03}$	$2^{22.45}$	$2^{41.75}$
R-SDP 3	127	7	187	111	$2^{9.78}$	$2^{16.58}$	$2^{23.58}$	$2^{44.86}$
R-SDP 5	127	7	251	150	$2^{9.78}$	$2^{17.01}$	$2^{24.41}$	$2^{47.14}$
R-SDPG 1	509	127	55	36	$2^{15.98}$	$2^{21.15}$	$2^{26.00}$	$2^{41.20}$
R-SDPG 3	509	127	79	48	$2^{15.98}$	$2^{21.56}$	$2^{26.67}$	$2^{44.50}$
R-SDPG 5	509	127	106	69	$2^{15.98}$	$2^{22.09}$	$2^{27.91}$	$2^{46.32}$

Practical Deployment and Countermeasures

Practical Deployment

- Initial calibration is required as timing is platform-dependent .
- Deterministic sampling enables transfer to other platforms after recalibration.

Countermeasures

- Simple methods: redundant expansion or hash checking.
- Correction codes can repair faulty entries, but increase computational cost.

Future Work: Evaluate and optimize countermeasures for practical deployment.

Thank You!

Questions?

Contact us via 220245548@seu.edu.cn