

Oil Vinegar and Sparks

Key Recovery from UOV via Single EM Fault Injection

Fabio Campos, **Daniel Hahn**, Daniel Könnecke, Marc Stöttinger

FDTC Conference 2026

UOV Basics

NIST L1: UOV-lp:
 $n = 112, m = 44$
 $q = 256$

hsrm*

- Multivariate signature scheme
 - third round of the NIST PQC standardization process (“Additional Digital Signatures”)

$$\mathcal{P} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$$

Public Key

System of m multivariate quadratic polynomials, with n variables

$$\mathbf{t} \in \mathbb{F}_q^m$$

Target Hash

Output vector from the message to be signed

$$\mathbf{s} \in \mathbb{F}_q^n$$

Signature

Vector of n variables that solves equation system for $\mathbf{t}$

Finding solution: $\mathcal{P}(\mathbf{s})=\mathbf{t}$ is hard MQ Problem

UOV Basics

NIST L1: UOV-lp:
 $n = 112, m = 44$
 $q = 256$

hsrm*

- Hidden trapdoor: $\mathcal{P}(\mathbf{o}) = 0$
 - Variables that collapse polynomial system to 0

$$\mathcal{O} \subset \mathbb{F}_q^n$$

Oil Space
(Secret Key)

$$\mathbf{o} \in \mathcal{O}$$

Oil Vector

$$\mathbf{v} \in \mathbb{F}_q^v$$

Vinegar Vector
(Randomly generated)

- Mixing Oil and Vinegar: $\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathcal{P}(\mathbf{v}) + \mathcal{P}(\mathbf{o}) + \mathcal{P}'(\mathbf{v}, \mathbf{o}) = \mathbf{t}$
 - Collapse to linear equation system: $\mathcal{P}'(\mathbf{v}, \mathbf{o}) = \mathbf{t} - \mathcal{P}(\mathbf{v})$
 - Solve against $\mathbf{t}$ to calculate signature $\mathbf{s}$
- ➔ Secret Oil Space allows efficient computation of signature $\mathbf{s}$ that satisfies the public polynomial system for any chosen message hash $\mathbf{t}$

Recovering O from leakage

- Entire oil space can be recovered from one oil vector
 - [1] “Kipnis-Shamir attack”: reveal 2nd oil vector in polynomial time
 - [2] “Reconciliation attack”: compute remaining oil space
- Oil Vector can be computed from vinegar and signature
($s = v + o$)
- Protection of individual oil and vinegar vectors is essential

[1] A. Kipnis and A. Shamir, “Cryptanalysis of the Oil and Vinegar Signature Scheme”, CRYPTO 1998

[2] J. Ding, B.-Y. Yang, C.-H. O. Chen, M.-S. Chen, and C.-M. Cheng, “New Differential-Algebraic Attacks and Reparametrization of Rainbow”, ACNS 2008

Relevant Related Work

- [3] Extract vinegar variables with **SCA** → Compute oil vector from vinegar & signature → Recover oil space
- [4] Fault attacks against **deterministic** UOV-based signature schemes (fixed vinegar variables)
- [5] SOK paper covering multiple SCA and FI attacks **theoretically on the algorithm level.**

[3] T. Aulbach, F. Campos, J. Krämer, S. Samardjiska, and M. Stöttinger, "Separating Oil and Vinegar with a Single Trace.", TCHES 2023

[4] S. Bauer, F. De Santis, and K. Koleci, "Fault attacks against UOV-based signatures," FDTC 2025

[5] T. Aulbach, F. Campos, J. Krämer. "SoK: On the Physical Security of UOV-Based Signature Schemes", PQCrypto 2025

Skip-Addition Attack

- Prevent addition of oil and vinegar during signature generation
→ Oil vector leaks into public signature

$$s = v + o \xrightarrow{\text{⚡}} s = o$$

Implementation Analysis

- Vinegar is copied first to the signature buffer and Oil is added onto it
- Oil is added onto pre-existing memory state of signature when skipping memcpy

⚡ (1) $s = v$

(2) $s = s + o$

```
1  int ov_sign( uint8_t *signature, const sk_t *sk, const uint8_t
    *message, size_t mlen ) {
2
3      uint8_t vinegar[_V_BYTE]; // vinegar
4      uint8_t y[_PUB_N_BYTE];   // oil
5
6      // generate random vinegar
7      hash_final_digest( vinegar, _V_BYTE, &h_vinegar);
8
9      uint8_t *w = signature; // w = T^-1 * x
10
11     // Copy vinegar to signature
12     memcpy( w, vinegar, _V_BYTE );
13
14     // identity part of T.
15     memcpy( w + _V_BYTE, x_o1, _O_BYTE );
16     // 0 part of T.
17     gformat_prod(y, sk->0, _V_BYTE, _O, x_o1 );
18
19     // Add oil onto signature
20     gf256v_add(w, y, _V_BYTE );
21
22     // add salt
23     memcpy( signature + _PUB_N_BYTE, salt, _SALT_BYTE );
24
25     return 0;
26 }
```

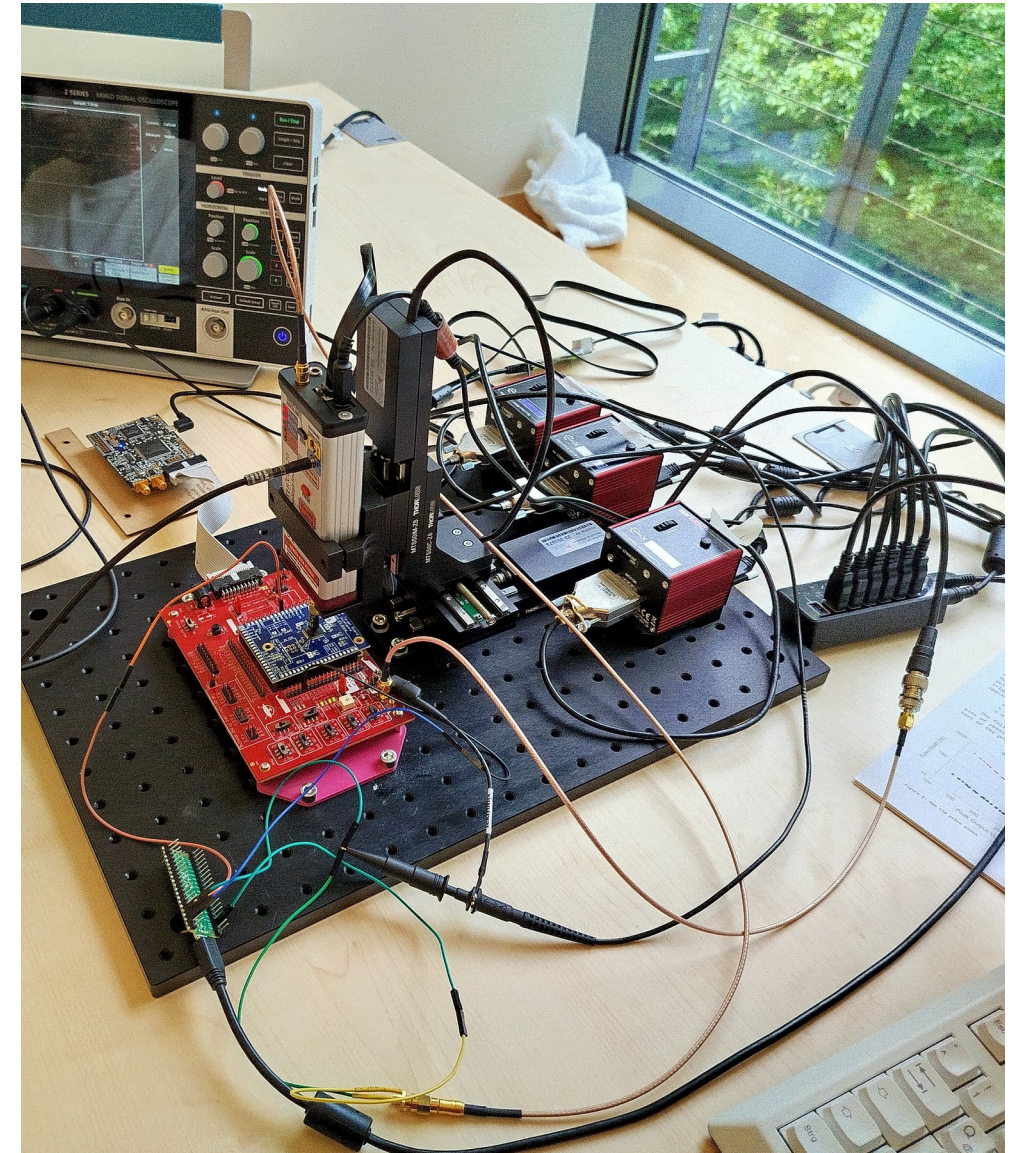
Attack Exploitability

- Attacker requires knowledge of pre-existing memory state for signature buffer
 - Deterministic Initialization value
 - Stack Zeroization
 - Multiple Signature Generations with same signature pointer (pqm4 benchmarks)

$$s = v + o \xrightarrow{\text{⚡}} s = s' + o$$

Fault Injection Setup

- STM32F415 Microcontroller
- Chipwhisperer Lite (flash & reset)
- ChipSHOUTER: EM Fault Injector
- Motorized XYZ Stage
- Raspberry Pi Pico: Pulse Generator
- Oscilloscope: Monitoring



EMFI Parameter Discovery

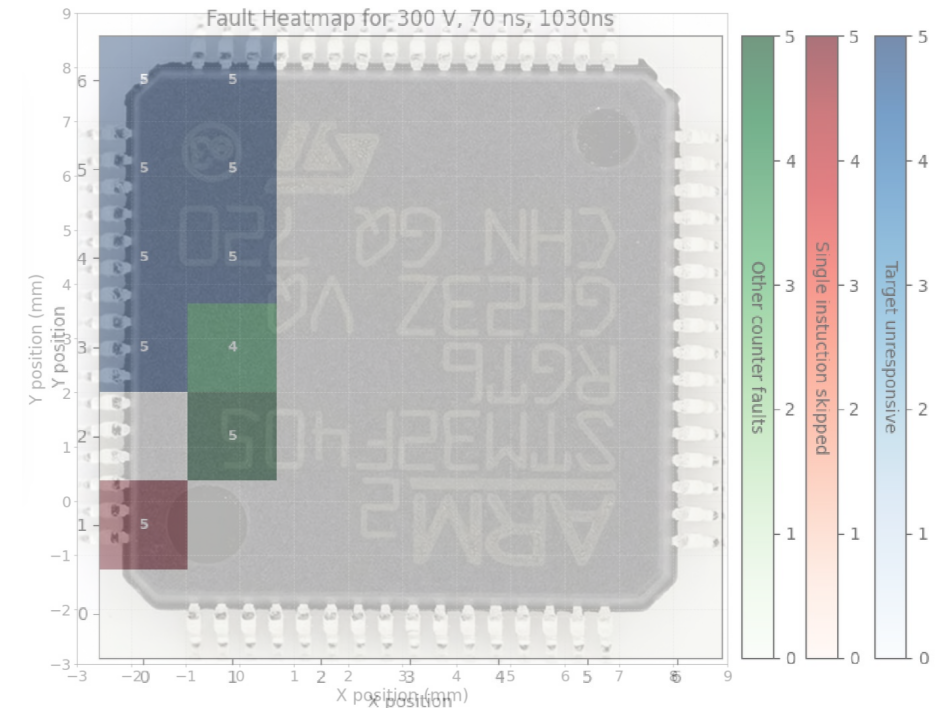
- Target Clock Speed → 7.37 MHz
 - Injection Coil Type → 4mm CW
 - Coil Position
 - Discharge Voltage
 - Pulse Length
 - Pulse Delay → Profiling Step 3
- Profiling Step 1 & 2

Profiling Step 1

- Unrolled loop incrementing register counter
 - 100 consecutive *ADD* operations
- GPIO trigger signal from target
- Inject faults for each combination of:
 - Voltage: (400, 350, 300 V)
 - Pulse length: (45, 50, 55, 60, 70 ns)
 - Coil X, Y position: (7x7 grid, 1 mm spacing)
- Glitch offset: (1 clock cycle = 140 ns, 10 ns steps)

Profiling result:

→ Glitch Parameters to skip a single addition instruction



Distinguished Outcomes:

- C=100 → “No Fault”
- C=99 → “Single Instruction Skip”
- Other C → “Other Fault”
- No response → “Crash”

Profiling Step 2

- Test identified glitch parameters on „b/“ instruction of memcpy
 - 1) Initialize buffers ***s** and ***d** with known data **s'** and **d'**
 - 2) Emit GPIO trigger signal
 - 3) Memcpy 68 bytes from source ***s** to destination ***d**
 - 4) Inject fault during memcpy operation (GPIO trigger signal)
 - 5) Communicate content of ***d** to host PC

Profiling result:

→ Glitch Parameters to skip memcpy function

Distinguished Outcomes:

- $s = d$ → “No Fault”
- $d = d'$ → “Memcpy fully skipped”
- $d \neq s$ → “Corrupted”
- No response → “Crash”

Profiling Step 3

- Same firmware as final attack
- Glitch parameters from profiling step 1
- Tune trigger offset to align with memcpy
- Validate success with known keypair

Algorithm 1 Target Firmware

```
1: uint8_t pk[PK_LEN], sk[SK_LEN];
2: uint8_t m[MLEN], sm[SMLLEN];

3: while true do
4:     wait_for_start();           ▷ 1. Wait for start signal
5:     set_trigger();              ▷ 2. Raise trigger signal
6:     crypto_sign(sm, m, len(m), sk);  ▷ 3. Sign message
7:     clear_trigger();
8:     uart_send(sm, len(sm));      ▷ 5. Return signature
9: end while
```

Profiling result:

→ Glitch insertion delay from trigger signal to skip memcpy in final attack firmware

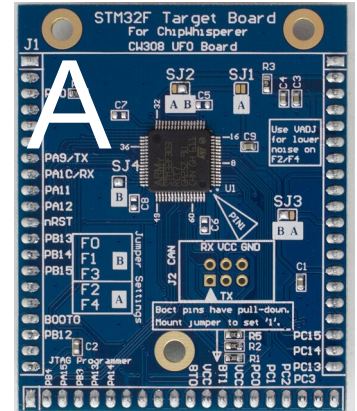
Attack Execution

- 1) Generate faultless signature
 - Initializes signature buffer with known state
 - Not required when memory content is otherwise known
 - 2) Generate second signature and inject fault
 - Prevents memcpy of vinegar into signature
 - 3) Extract oil vector by subtracting the two signatures
 - 4) Recover the secret oil space with Kipnis-Shamir and Reconciliation attack
- Success Rate > 80 %**

Real-World Feasibility

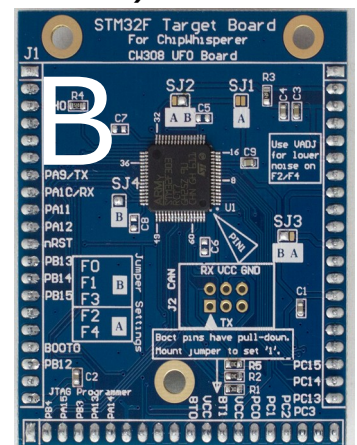
- Attack depends on application-level characteristics (memory predictability)
- Time consuming validation of glitch success (~ 30 minutes for full key recovery)
 - Profiling on replica device is strongly encouraged
- Portability of profiling results between devices (A → B):
 - Evaluated transferability of parameters from STM32F405 to STM32F415 (same firmware)

1) profile



STM32F405

2) attack



STM32F415

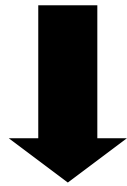
Countermeasures

Avoid intermediate storage of unmixed oil or vinegar in signature

1 Change *gf256v_add* function signature

```
1 uint8_t vinegar[_V_BYTE]; // vinegar
2 uint8_t y[_PUB_N_BYTE]; // oil
3 // ...
4 uint8_t *w = signature;
5 // Transform solved oil variables
6 gformat_prod(y, sk->0, _V_BYTE, _0, x_o1 );
7 // Mix oil and vinegar, output to signature
8 gf256v_add(w, vinegar, y, _V_BYTE );
```

gf256v_add(dest, src, len);



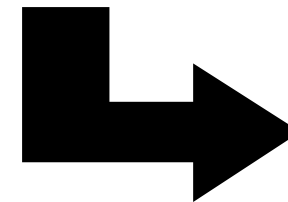
gf256v_add(out, in_1, in_2, len);

2 Mix **o** and **v** in local “**vinegar**” variable

```
1 uint8_t vinegar[_V_BYTE]; // vinegar
2 uint8_t y[_PUB_N_BYTE]; // oil
3 // ...
4 uint8_t *w = signature;
5
6 gformat_prod(y, sk->0, _V_BYTE, _0, x_o1 );
7 gf256v_add(vinegar, y, _V_BYTE );
8 memcpy( w, vinegar, _V_BYTE );
```

$$(1) \quad s = v$$

$$(2) \quad s = s + o$$



$$(1) \quad v = v + o$$

$$(2) \quad s = v$$

Conclusion

- First practical fault attack on non-deterministic UOV
- Fast parameter discovery methodology
- Portability of EMFI parameters between devices
- Simple Countermeasures

Questions?

Contact:

RheinMain University of Applied Sciences and Arts
[hs-rm.de](https://www.hs-rm.de)

Unter den Eichen 5,
65195 Wiesbaden, Germany

Daniel Hahn, Marc Stöttinger, Daniel Könnecke

- Daniel.Hahn@hs-rm.de
- Marc.Stoettinger@hs-rm.de
- Daniel@koe-mail.com

Darmstadt University of Applied Sciences
[h-da.de](https://www.h-da.de)

Schöfferstraße 3
64295 Darmstadt

Fabio Campos

- campos@sopmac.de