

# Achilles: A Formal Framework of Leaking Secrets from Signature Schemes via Rowhammer

USENIX Security 2025

**Junkai Liang\***, Zhi Zhang\*, Xin Zhang\*, Qingni Shen, Yansong Gao, Xingliang Yuan, Haiyang Xue, Pengfei Wu, Zhonghai Wu



# A Threat to Signature Schemes

Signature Scheme serves as backbones for network security



Network protocol



Blockchain Wallet



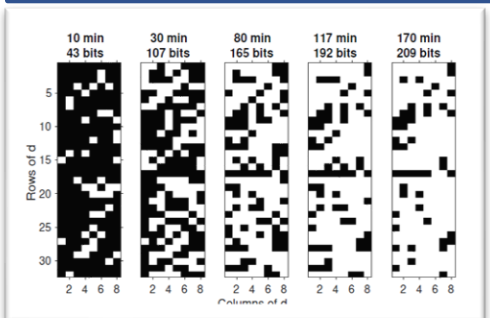
Software Updates



Secure Message

Regarding the fault injection attack (FIA), most of the schemes are vulnerable

Full Key Recovery



FIA leads to full key recovery of ECDSA, RSA, etc [22].

Post Quantum Attacks

IACR ePrint Updates @Lhree

New! Beware of Keccak: Practical Fault Attacks on SHA-3 to Compromise Kyber and Dilithium on ARM Cortex-M Devices (Yuxuan Mang and Jintong Yu and Shipai Qu and Xiaolin Zhang and Xiaowei Li and Chi Zhang and Dawu Gu)

eprint.iacr.org

Mind the Faulty Keccak: A Practical Fault Injection Attack ... ML-KEM and ML-DSA are NIST-standardized lattice-based post-quantum cryptographic algorithms. In both ...

Real Impact CVEs

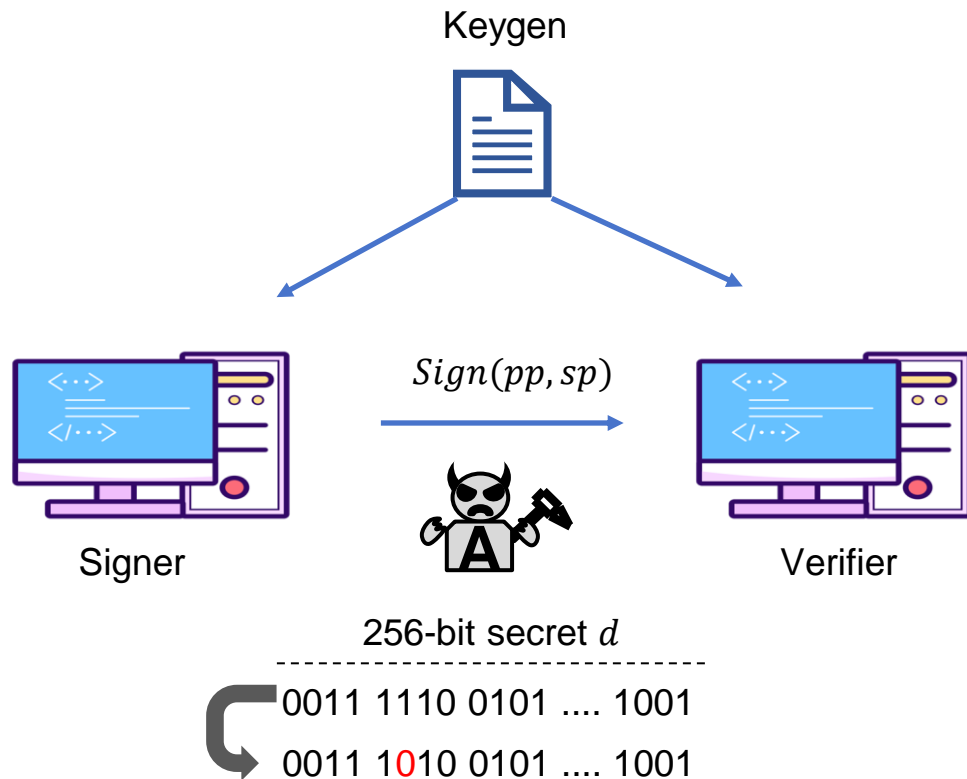
Known wolfSSL Vulnerabilities

The SSL/TLS protocol is documented and under constant scrutiny by the top experts in security and cryptography. SSL was quickly adopted as a standard world wide. SSL and TLS together secure communications between billions of computers, servers, Internet of Things (IoT) devices, and embedded systems. The security provided by an SSL/TLS Library depends on the underlying strength of its cryptography which is used to encrypt communications.

INFO	CVE ID	SEVERITY	DESCRIPTION	TIME TO FIX	FIXED IN VERSION
LINK	CVE-2022-42961	Low	Fault injection attack on RAM via Rowhammer leads to ECDSA key disclosure. Users doing operations with private ECC keys such as server side TLS connections and creating ECC signatures, who also have hardware that could be targeted with a sophisticated Rowhammer attack should update the version of wolfSSL and compile using the macro WOLFSSL_CHECK_SIG_FAULTS. Thanks to Yarkin Doroz, Berk Sunar, Koksai Must, Caner Tol, and Kristi Rahman all affiliated with the Vernam Applied Cryptography and	5 days	5.5.C

# Inject Faults into the parameters of Signature Schemes

- ❑ Parameters of the scheme are loaded into memory cells when signing.
  - ❑ The attacker exploits Rowhammer bugs to tamper with memory data.
  - ❑ Parameters of the scheme can be tampered.



Year	Paper	Scheme	Parameter
2024	Mus's[22]	ECDSA	$d$ (sp)
2023	Saad's[24]	Dilithium	$s$ (sp)
2020	Zane's[11]	RSA	$d$ (sp)
2018	Mus's[23]	LUOV	$d$ (sp)
2018	Podd's[23]	EdDSA	$h$ (pp)
2016	Razavi's[10]	RSA	$N$ (pp)
.....			

# Problems of Existing works

- ❑ Many signature schemes are **not** analysed, e.g., BLS, BB-Short, Elgamal, MLDSA, etc.
- ❑ It is still **unclear** how we utilise existing analysis for new signature schemes.

# Problems of Existing works

- ❑ Many signature schemes are **not** analysed, e.g., BLS, BB-Short, Elgamal, MLDSA, etc.
- ❑ It is still **unclear** how we utilise existing analysis for new signature schemes.



Is there a formal framework ?

# Problems of Existing works

- ❑ Many signature schemes are **not** analysed, e.g., BLS, BB-Short, Elgamal, MLDSA, etc.
- ❑ It is still **unclear** how we utilise existing analysis for new signature schemes.

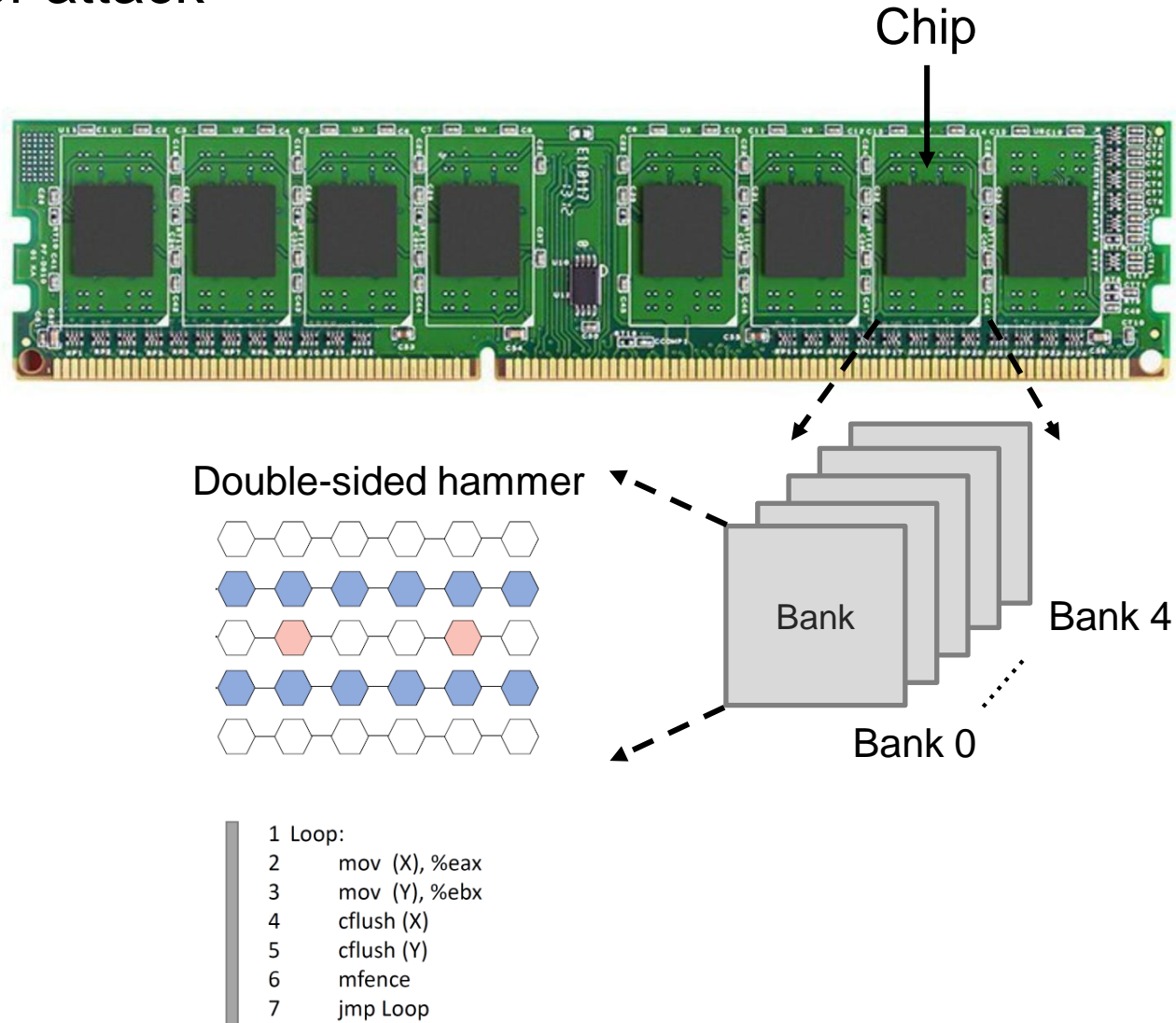


Is there a formal framework ?

- ❑ Yes! Our framework Achilles targets generalised signature schemes, and can be used for:
  - 1> discovering **new vulnerable parameters** even in studied schemes
  - 2> helping identify **new vulnerable schemes**

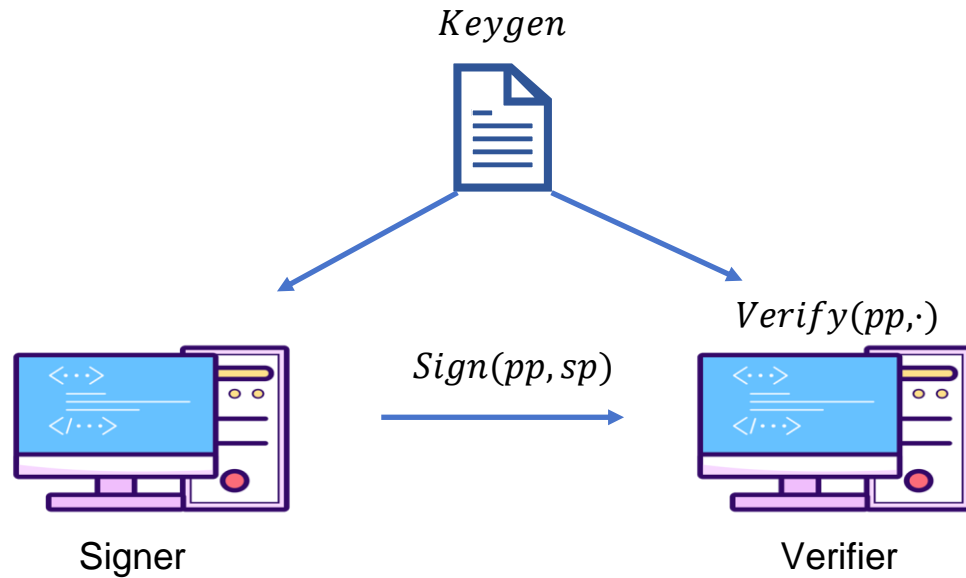
# Background

## Rowhammer attack



# Background

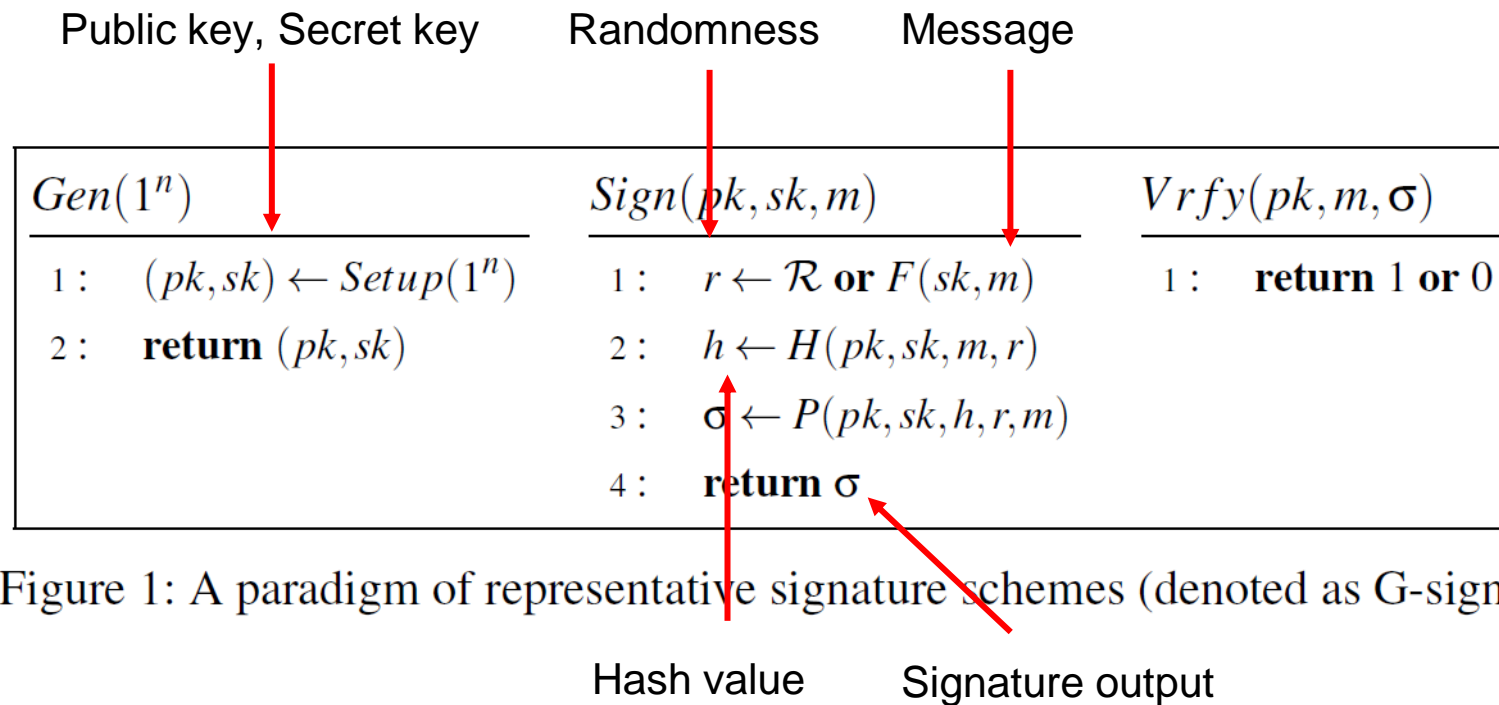
- Signature Schemes: the idea is simple. The verifier verifies the identity of the signer by using public parameters to verify the signature



# Achilles – Step 1

## □ A formal Treatment of Signature Schemes

- It is based on the general definition of signature schemes
- It includes some inevitable key parameters in signature schemes



# Achilles – Step 2

- A Game-based model against  $G$ -sign
  - $OSign$ : Sign a message and return the signature

$OSign(pk, sk, m)$

---

1:  $\sigma \leftarrow Sign(pk, sk, m)$   
2:  $M \leftarrow M \cup \{m\}$   
3: **return**  $\sigma$

Figure 2: Formalizing Rowhammer faults against G-sign.

# Achilles – Step 2

- A Game-based model against  $G$ -sign
  - $OSign$ : Sign a message and return the signature
  - $f_i(x)$ : formalization of Rowhammer fault (a random model)

$OSign(pk, sk, m)$	$f_i(x)$
1: $\sigma \leftarrow Sign(pk, sk, m)$	1: $f_i^{set} \leftarrow \emptyset$
2: $M \leftarrow M \cup \{m\}$	2: $k \leftarrow RNG(0,  x  - 1)$
3: <b>return</b> $\sigma$	3: $x[k] \leftarrow x[k] \oplus 1$
	4: $f_i^{set} \leftarrow f_i^{set} \cup \{k, x[k]\}$
	5: <b>return</b> $x$

Figure 2: Formalizing Rowhammer faults against G-sign.

# Achilles – Step 2

## □ A Game-based model against $G$ -sign

- $OSign$ : An oracle which signs a message and return the signature
- $f_i(x)$ : formalization of Rowhammer fault (a random model)
- $OFSign$ : An oracle which allows fault occurs in the key parameters.



$OSign(pk, sk, m)$	$f_i(x)$	$OFSign(i, pk, sk, m)$		
1: $\sigma \leftarrow Sign(pk, sk, m)$	1: $f_i^{set} \leftarrow \emptyset$	1: $\forall j \neq i, \text{set } f_j \text{ as a void function}$	 <p><math>i</math> is specified by the attacker, indicated which parameter is faulted</p>	
2: $M \leftarrow M \cup \{m\}$	2: $k \leftarrow RNG(0,  x  - 1)$	2: $r \leftarrow \mathcal{R} \text{ or } F(f_1(sk), f_2(m))$		
3: <b>return</b> $\sigma$	3: $x[k] \leftarrow x[k] \oplus 1$	3: $h \leftarrow H(f_3(pk), f_4(sk), f_5(m), f_6(r))$		
	4: $f_i^{set} \leftarrow f_i^{set} \cup \{k, x[k]\}$	4: $\sigma \leftarrow P(f_7(pk), f_8(sk), f_9(h), f_{10}(r), f_{11}(m))$		
	5: <b>return</b> $x$	5: $F_f \leftarrow \bigcup f_i^{set}$		
		6: $MF \leftarrow MF \cup \{m\}$		 <p>Store the fault information in <math>F_f</math> and <math>MF</math></p>
		7: <b>return</b> $\sigma$		

Figure 2: Formalizing Rowhammer faults against G-sign.

# Achilles – Step 2

## □ A Game-based model against $G\text{-sign}$

- $OSign$ : An oracle which signs a message and return the signature
- $f_i(x)$ : formalization of Rowhammer fault (a random model)
- $OFSign$ : An oracle which allows fault occurs in the key parameters.
- $Exp^F$ : A Game experiment indicated two kinds of capabilities of the attacker.

```
ExpF( $\mathcal{A}$ )
-----
1:  $M \leftarrow \emptyset, MF \leftarrow \emptyset, F_f \leftarrow \emptyset$ 
2:  $(sk, pk) \leftarrow Gen()$ 
3:  $(m^*, \sigma^*, f^*) \leftarrow \mathcal{A}^{OSign(\cdot), OFSign(i, \cdot)}(pk)$ 
4:  $v \leftarrow Vrfy(pk, m^*, \sigma^*)$ 
5: return 1 if  $v == 1 \wedge m^* \notin M \wedge m^* \notin MF$ 
6: return 2 if  $v == 1 \wedge m^* \notin M \wedge m^* \in MF$ 
    $\wedge f^* \in F_f$ 
7: return 0, otherwise
```

Line 5 : The attacker can output a valid signature on a **new** message

💡 The fault occurs on the public parameters (pp)

Line 6 : The attacker can output a valid signature on an existed message which **was queried** in  $OFSign$

💡 The fault occurs on the secret parameters (sp)

Figure 3: A game-based experiment modelling post-Rowhammer analysis.

# Achilles – Step 3

- We design algorithms to recover the secrets when the fault occurs on  $sp$  or  $pp$

---

**Algorithm 1:** Post-Rowhammer analysis via DFA

---

1 Initially,  $S = (Gen, Sign, Vrfy)$  is a signature scheme.  
 $OSign$  and  $f_i(x)$  are function oracles defined in **Figure 2**.

2 Parse  $S$  to G-sign

3 **foreach**  $p \in \{m, pk, h\}$  **do**

    // get a valid and faulty signature.

4      $\sigma \leftarrow OSign(\cdot)$

5     Use  $i$  to denote the index for parameter  $p$

6      $\sigma' \leftarrow OFSign(i, \cdot)$

7     return  $sk' = (\sigma - \sigma') / g(pp)$

8 **end**

---

→ enumerate all possible public parameters

→ get the representation of the faulty signature

↓  
judge if it is exploitable by differential analysis

# Achilles – Step 3

- We design algorithms to recover the secrets when the fault occurs on  $sp$  or  $pp$

---

**Algorithm 2:** Post-Rowhammer analysis via SCA

---

```
3 foreach  $p \in \{sk, r\}$  do
  // get a valid and faulty signature.
4   Use  $i$  to denote the index for parameter  $p$ 
5   do
6      $\sigma' \leftarrow OFSign(i, \cdot)$ 
  // enumerate the one-bit fault.
7      $\Delta \leftarrow 0, j \leftarrow 0$ 
8     do
9       if  $Vrfy(pk \cdot g(pp, \Delta d), m, \sigma') = 1$  then
10        |  $N++$ 
11        |  $sk' = sk' \oplus \Delta$ 
12        | end
13        |  $\Delta \leftarrow \Delta \ll 1, j \leftarrow j + 1$ 
14      while  $j < |sk'|$ 
15    while  $|sk'| - recovered\_bits > 50$ 
16 end
17 return  $sk'$ 
```

→ enumerate all possible secret parameters

→ get the representation of the faulty signature

→ try to correct the faulty signature to a valid one

→ use  $\Delta$  as bit leakage

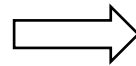
---

# Achilles – Step 3

- ❑ To use the algorithms, the attacker need first instantiates with a specified scheme
- ❑ An *EdDSA* example:

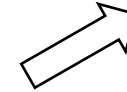
*KeyGen:*  
 $sk \leftarrow s$   
 $pk \leftarrow (P, B)$   
 $[P = sB]$

*Sign:*  
 $r = H(m)$   
 $R = rB$   
 $h = H(R||P||m)$   
 $\sigma = r + hs$

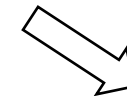


Scheme	<i>pp</i>				<i>sp</i>
EdDSA	<i>h</i>	<i>m</i>	<i>P, B</i>	<i>r, R</i>	<i>s</i>
G-sign	<i>h</i>	<i>m</i>	<i>pk</i>	<i>r</i>	<i>sk</i>

Table 2: Classification of potentially vulnerable parameters in EdDSA.



*DFA:* when *h* is faulted, the faulty signature  $\sigma' = r + (h + \Delta)s$  is exploitable



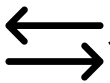
*SCA:* when *s* is faulted, the faulty signature  $\sigma' = r + h(s + \Delta)$  is exploitable

# Achilles – AutoVuln

- The process can be automated by using symbolic operations.

```
./AutoVuln -pk_list = [] -sk_list = [] -h_list = [] -m_list  
= [] -r_list = [] -sign_list = [] -vrfy_list = []
```

description of the **symbols**



Potentially vulnerable  
parameters



description of the **equations**

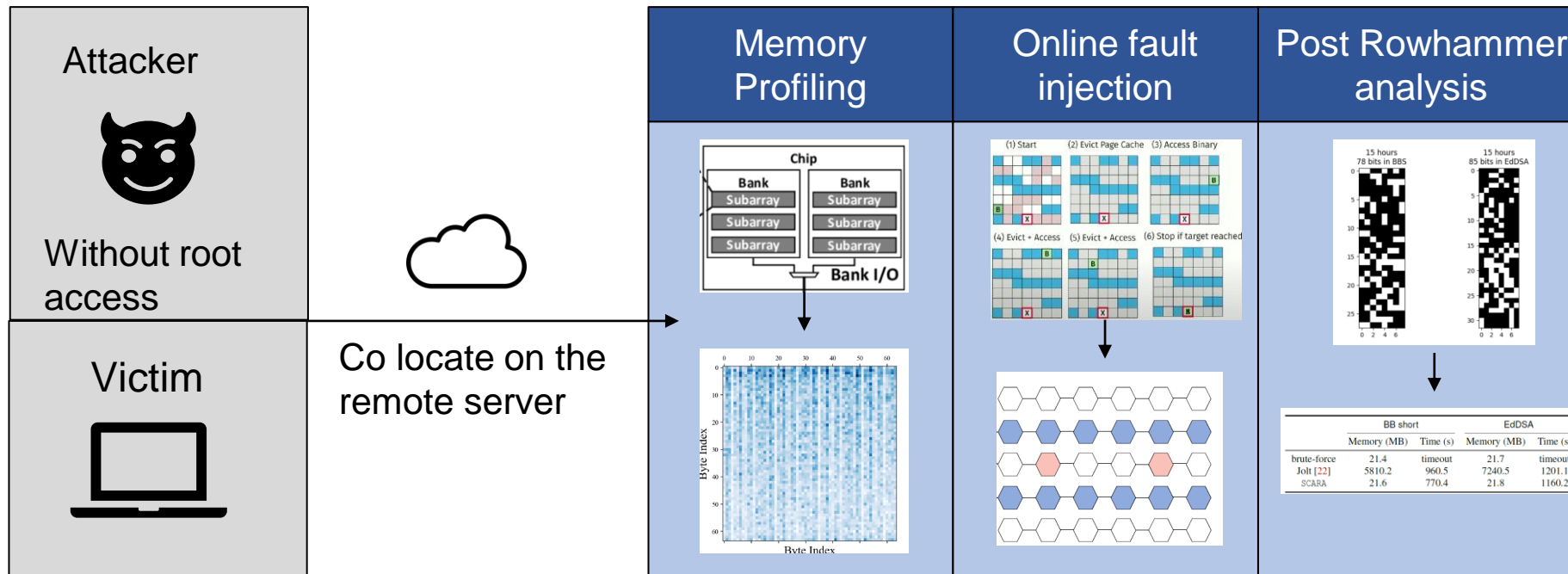
Result:

Scheme	Library	Time	Identified Potentially Vulnerable Parameter					
			<i>pk</i>	<i>h</i>	<i>m</i>	<i>sk</i>	<i>r</i>	
BB short	relic-toolkit-0.6.0	64.0 ms					s	
EdDSA	wolfSSL-5.6.6	75.2 ms	R,P	h	m		s	
ML-DSA	liboqs-0.10.0	85.0 ms	$\mu,w$	c	m		$s_1$	
Elgmal	cryptopp-8.9	66.9 ms					x	k
RSA	wolfSSL-5.6.6	54.5 ms					s	
BLS	bls-signatures-2.0.3	52.2 ms					x	

Table 3: The execution time and potentially vulnerable parameters identified automatically by AutoVuln.

# Evaluation

- We follow the idea of including offline memory profiling, online fault injection and post Rowhammer analysis for performing an attack.



# Evaluation

- Memory profiling:

- We first run DRAMdig to get the bank function of our DRAM, from which we can know if two addresses are in the same bank

The bank function of our DRAM: (17,21), (16,20), (15,19), (14,18), (6,13)

- Then we profile 12GB of physical memory

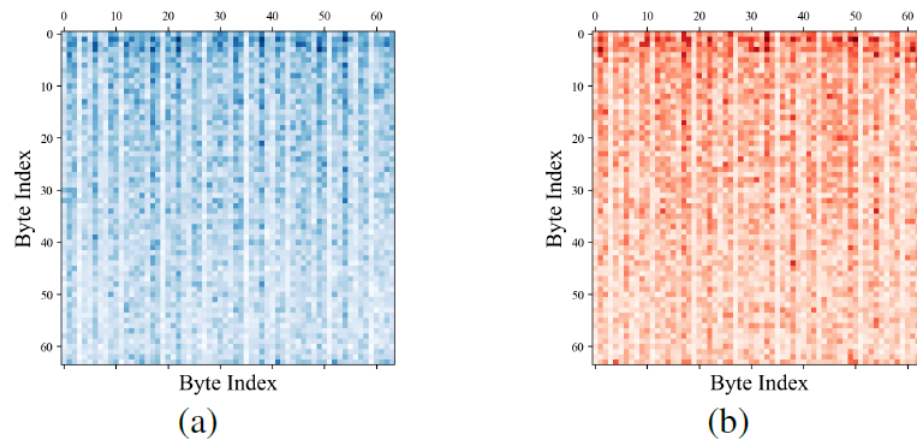


Figure 4: The distribution of flippable-bit offsets over 4 KB-aligned pages. Bit flips from 1 to 0 (blue) and bit flips from 0 to 1 (red) accumulated over 4 KB pages.

# Evaluation

## ❑ Online Fault Injection:

- ❑ We utilize the behaviour of Linux buddy allocator (first-in-last-out), which allows us to do memory massaging by releasing vulnerable pages and coerce the victim to use these pages to store the parameters.
- ❑ **Challenge:** the fault must occur in a limited **time window** between a parameter's initiation and its subsequent use in computations
- ❑ Our solution: we opt for **OS signals**. The attacker process sends the registered signal to trick the OS kernel into switching the victim process off the core.

# Evaluation

## □ Post Rowhammer Analysis:

- From the faulty signatures, we are able to recover the secrets.
- In DFA, we only need one faulty signature for full recovery (EdDSA).
- In SCA, we need hundreds of faulty signatures. We also need to find the remaining bits.

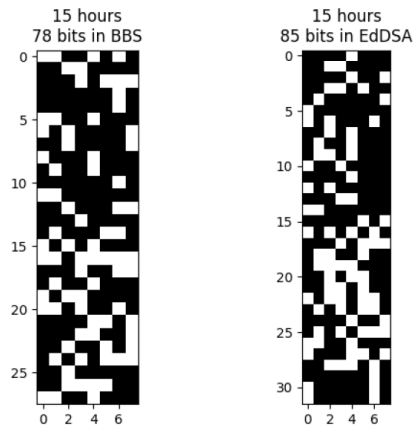


Figure 5: The number of recovered bits in 15 hours for the secret key in BB short and EdDSA is represented in the form of pixels, where white cells indicate recovered bits, and black cells indicate non-recovered bits.

	BB short		EdDSA	
	Memory (MB)	Time (s)	Memory (MB)	Time (s)
brute-force	21.4	timeout	21.7	timeout
Jolt [22]	5810.2	960.5	7240.5	1201.1
SCARA	21.6	770.4	21.8	1160.2

Table 4: The memory and time overhead when recovering the remaining secret bits using brute-force, Jolt [22] and our SCARA, respectively.

# Countermeasure


## ❑ Specific defenses of signature schemes

- ❑ Leakage resilient cryptographic schemes
- ❑ Verifying after signing
- ❑ Redundancy check

## ❑ WolfSSL has adopted our suggestion and implement “verifying after signing” and “Redundancy check” for RSA and EdDSA.

<a href="#">LINK</a>	<a href="#">CVE-2024-2881</a>	Medium	Fault injection attack with EdDSA signature operations. This affects ed25519 sign operations where the system could be susceptible to Rowhammer attacks. Thanks to Junkai Liang, Zhi Zhang, Xin Zhang, Qingni Shen for the report (Peking University, The University of Western Australia). Fixed in this GitHub pull request #7212	0 days	5.7.0
<a href="#">LINK</a>	<a href="#">CVE-2024-1545</a>	Medium	Fault Injection vulnerability in RsaPrivateDecryption function that potentially allows an attacker that has access to the same system with a victims process to perform a Rowhammer fault injection. Thanks to Junkai Liang, Zhi Zhang, Xin Zhang, Qingni Shen for the report (Peking University, The University of Western Australia)." Fixed in this GitHub pull request #7167	9 days	5.7.0

### EdDsa: check private value after sign #7212

 Merged dgarske merged 1 commit into wolfSSL:master from SparkiDev:eddsa\_check\_priv on Feb 14, 2024

 Conversation 4  Commits 1  Checks 0  Files changed 3



SparkiDev commented on Feb 5, 2024 • edited

Contributor ...

#### Description

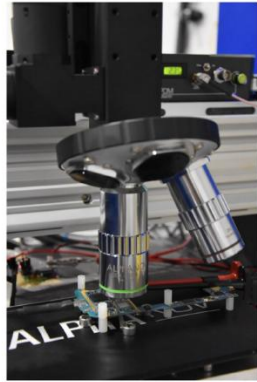
Check the private value hasn't changed during signing with EdDSA.

Fixes zd#17438

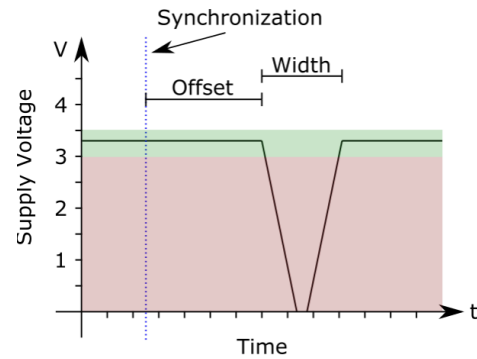
In our work, we have shown that these countermeasures can be bypassed by faulting the opcode

# Physical Fault Injection

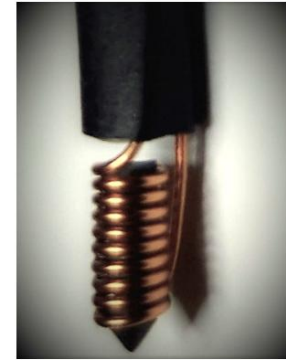
- ❑ Can physical fault injection achieve similar results ?



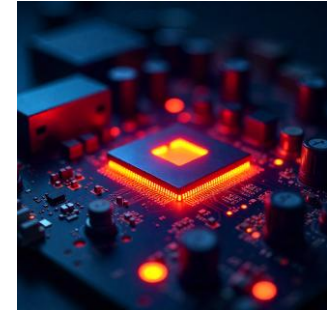
Optical radiation



Voltage/Clock Glitch



Electromagnetic pulses



Temperature attack

- ❑ The temporal and spatial precision of these attacks does not meet our requirements. These attacks do not allow remote attackers.

# Thank you !

Questions ?

<https://github.com/liang-junkai/Achilles>

For details and further info:

Junkai Liang  
(ljknjupku@gmail.com)

## More Insights, Code and Data In Website

The screenshot shows a Zenodo record page for the artifact "Achilles: A Formal Framework of Leaking Secrets from Signature Schemes via Rowhammer". The page is published on March 21, 2025, and is version v4. It features a blue header with the Zenodo logo, a search bar, and navigation links for "Communities" and "My dashboard". The main content area includes the title, authors (Liang, Junkai; Zhang, Xin; Zhang, Zhi), and a detailed abstract. The abstract discusses the vulnerability of signature schemes to Rowhammer attacks and the development of the Achilles framework. On the right side, there are buttons for "Edit", "New version", and "Share", along with statistics showing 84 views and 45 downloads. A "Versions" table lists the artifact's history, including versions v1, v2, v3, and v4, with their respective dates and identifiers.

Published March 21, 2025 | Version v4

Software Open

### Artifact for "Achilles: A Formal Framework of Leaking Secrets from Signature Schemes via Rowhammer"

Liang, Junkai (Researcher)<sup>1</sup>; Zhang, Xin (Project leader)<sup>1</sup>; Zhang, Zhi (Researcher)<sup>2</sup>

Show affiliations

Signature schemes are a fundamental component of cybersecurity infrastructure. While they are designed to be mathematically secure against cryptographic attacks, they are vulnerable to Rowhammer fault-injection attacks. Since all existing attacks are ad-hoc in that they target individual parameters of specific signature schemes, it remains unclear about the impact of Rowhammer on signature schemes as a whole.

In this paper, we present Achilles, a formal framework that aids in leaking secrets in various real-world signature schemes via Rowhammer. Particularly, Achilles can be used to find potentially more vulnerable parameters in schemes that have been studied before and also new schemes that are potentially vulnerable. Achilles mainly describes a formal procedure where Rowhammer faults are induced to key parameters of a generalized signature scheme, called G-sign, and a post-Rowhammer analysis is then performed for secret recovery on it. To illustrate the viability of Achilles, we have evaluated six signature schemes (with five CVEs assigned to track their respective Rowhammer vulnerability), covering traditional and post-quantum signatures with different mathematical problems. Based on the analysis with Achilles, all three schemes are proved to be vulnerable, and two new vulnerable parameters are identified for EdDSA. Further, we demonstrate a successful Rowhammer attack against each of these schemes, using recent cryptographic libraries including `_woffs1_`, `_relic_`, and `_liboqs_`. The artifact contains 3 main components: 1) the attacker's code related to fault injection. 2) the attacker's code that analyses the faulty signature. 3) the automation tool for Achilles which analyses the signature schemes.

#### Files

Version	Date
Version v4 10.5281/zenodo.15062399	Mar 21, 2025
Version v3 10.5281/zenodo.15050315	Mar 19, 2025
Version v2 10.5281/zenodo.15032326	Mar 16, 2025
Version v1 10.5281/zenodo.14735640	Jan 24, 2025