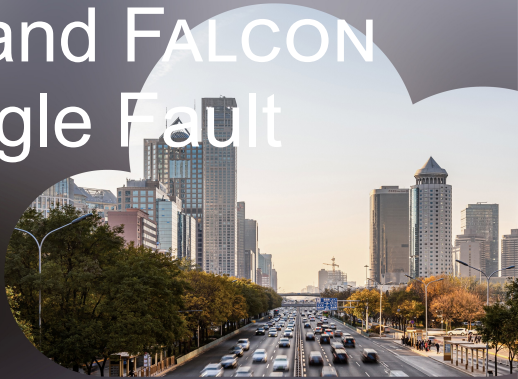
Forging DILITHIUM and FALCON Signatures with Single Fault Injection

Sven Bauer and Fabrizio De Santis



SIEMENS

Reasons for attacking signature verification

Embedded devices verify signatures to

- ensure integrity of data, e.g. software updates, before processing
- authenticate communication

SIEMENS

Reasons for attacking signature verification

Embedded devices verify signatures to

- ensure integrity of data, e.g. software updates, before processing
- authenticate communication

Hence, bypassing signature verification can allow an attacker to

- install arbitrary software on a device
- access confidential data

SIEMENS

How to attack signature verification

How to force unintended behaviour: Fault injection!

SIEMENS

How to attack signature verification

How to force unintended behaviour: Fault injection!

The obvious place:

- Signature verification ends with a 1-bit result (“valid” or “invalid”).
- So, attacking near the end seems most promising.
- Implementers know this and will protect this critical section.

SIEMENS

How to attack signature verification

How to force unintended behaviour: Fault injection!

The obvious place:

- Signature verification ends with a 1-bit result (“valid” or “invalid”).
- So, attacking near the end seems most promising.
- Implementers know this and will protect this critical section.

What about other places in the code? Perhaps overlooked by implementers?

- Example: Attack by Muir and Seifert against RSA signature verification (Reminder: Flip bits of public modulus until it is easy to factor; inject fault to achieve same bit flips during verification)
- We are moving into the post-quantum world.
- Similar attacks against verification of post-quantum signatures?

SIEMENS

DILITHIUM signature verification: quick recap

Notation: $R = \mathbb{Z}[X]/(X^n + 1)$, $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, $n = 256$, $q = 8380417$

Require: A message m, a signature $\mathtt{sig} = (\tilde{\mathbf{c}}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

5: **if** $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

DILITHIUM signature verification: quick recap

Notation: $R = \mathbb{Z}[X]/(X^n + 1)$, $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, $n = 256$, $q = 8380417$

Require: A message m, a signature $\mathtt{sig} = (\tilde{\mathbf{c}}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

5: **if** $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

the obvious place

SIEMENS

DILITHIUM signature verification: quick recap

Notation: $R = \mathbb{Z}[X]/(X^n + 1)$, $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, $n = 256$, $q = 8380417$

Require: A message m, a signature $\mathtt{sig} = (\tilde{\mathbf{c}}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - \boxed{c\mathbf{t}_1} \cdot 2^d, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

5: **if** $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

a less obvious place

the obvious place

SIEMENS

What the attacker does...

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

5: **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

$\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

$\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

$c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

$\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

if $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

    accept

else

    reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

$\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

$\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

$c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

$\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

if $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

    accept

else

    reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ ▷ $\mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

(3) inject fault

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ ▷ $c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} \cancel{- c\mathbf{t}_1}, 2\gamma_2)$ ▷ $\mathbf{w}'_1 \in R_q^k$

5: **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

$\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

$\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

(3) inject fault

$c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

$\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} \cancel{- c\mathbf{t}_1}, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

if $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

  accept

else

  reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

$\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

$\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

(3) inject fault

$c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

$\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} \cancel{- c\mathbf{t}_1}, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q^k$

if $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

  accept

else

  reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

(2) calculate signature

(3) inject fault

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ ▷ $\mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ ▷ $c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ ▷ $\mathbf{w}'_1 \in R_q^k$

5: **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

In code (example from `pqm4` project, `crypto_sign/dilithium2/m4f/sign.c`)

```
...
poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);

polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
...
```

← skip this call with fault injection

SIEMENS

In code (example from `pqm4` project, `crypto_sign/dilithium2/m4f/sign.c`)

```
282     ...
283     poly_ntt(&cp);
284     polyveck_shiftl(&t1);
285     polyveck_ntt(&t1);
286     polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);
287
288     polyveck_sub(&w1, &w1, &t1);
289     polyveck_reduce(&w1);
290     ...
```

skip this call with fault injection

single fault to skip instruction

In our setup, line 288 compiles to this Cortex-M4 code:

```
...
0x80030c4 <crypto_sign_verify+252> adds r2, #16
0x80030c6 <crypto_sign_verify+254> bl 0x80028f8 <polyveck_sub>
0x80030ca <crypto_sign_verify+258> add.w r0, sp, #10304
...
```


SIEMENS

Countermeasures

- Generic countermeasures to ensure control flow integrity.
- Replace

$$w_1' \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - \mathtt{c}\mathbf{t}_1, 2\gamma_2)$$

with

$$w_1' \leftarrow \mathtt{UseHint}(\mathbf{h}, (\mathbf{Az} + \mathbf{u}) - (\mathtt{c}\mathbf{t}_1 + \mathbf{u}), 2\gamma_2),$$

where $\mathbf{u}$ is a random vector.

SIEMENS

Alternative attack

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

(2) calculate signature

(3) inject fault

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ ▷ $\mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ ▷ $c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ ▷ $\mathbf{w}'_1 \in R_q$

5: **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

Alternative attack

(1) choose message

Require: A message m, a signature $\mathtt{sig} = (\tilde{c}, \mathbf{z}, \mathbf{h})$, a public key $\mathtt{pk} = (\rho, \mathbf{t}_1)$.

Ensure: Accept or reject

(2) calculate signature

(3) inject fault

1: $\mathbf{A} \leftarrow \mathtt{ExpandA}(\rho)$ $\quad \triangleright\ \mathbf{A} \in R_q^{k \times \ell}$

2: $\mu \leftarrow \mathtt{SHAKE256}(\mathtt{SHAKE256}(\rho \| \mathbf{t}_1, 256) \| m, 512)$

3: $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$ $\quad \triangleright\ c \in B_\tau \subset R$

4: $\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1, 2\gamma_2)$ $\quad \triangleright\ \mathbf{w}'_1 \in R_q$

5: **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{SHAKE256}(\mu \| \mathbf{w}'_1, 256)$ and the number of 1s in $\mathbf{h}$ is $\leq \omega$ **then**

6: accept

7: **else**

8: reject

SIEMENS

In code (example from `pqm4` project, `crypto_sign/dilithium2/m4f/sign.c`)

```
...
/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
polyveck_pack_w1(buf, &w1);

/* Call random oracle and verify challenge */
shake256_inc_init(&state);
shake256_inc_absorb(&state, mu, CRHBYTES);
shake256_inc_absorb(&state, buf, K*POLYW1_PACKEDBYTES);
shake256_inc_finalize(&state);
...
```

skip this call with fault injection

SIEMENS

FALCON signature verification: quick recap

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

```
c ← HashToPoint(r||m)                                   ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)                                     ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q                                   ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ||(s_1, s_2)||^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```


SIEMENS

FALCON signature verification: quick recap

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

1: $c \leftarrow \texttt{HashToPoint}(r \| m)$ ▷ $c \in \mathbb{Z}_q[x]$

2: $s_2 \leftarrow \texttt{Decompress}(s)$ ▷ $s_2 \in \mathbb{Z}[x]$

3: **if** $s_2 = \perp$ **then**

4: reject

5: $s_1 \leftarrow c - s_2 h \bmod q$ ▷ coefficients of s_1 normalized to be between $\lceil -\frac{q}{2} \rceil$ and $\lfloor \frac{q}{2} \rfloor$

6: **if** $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$ **then**

7: accept

8: **else**

9: reject

the obvious place

SIEMENS

FALCON signature verification: quick recap

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

1: $c \leftarrow \texttt{HashToPoint}(r \| m)$ ▷ $c \in \mathbb{Z}_q[x]$

2: $s_2 \leftarrow \texttt{Decompress}(s)$ ▷ $s_2 \in \mathbb{Z}[x]$

3: **if** $s_2 = \perp$ **then**

4: reject

a less obvious place

5: $s_1 \leftarrow \boxed{c} - s_2 h \bmod q$ ▷ coefficients of s_1 normalized to be between $\lceil -\frac{q}{2} \rceil$ and $\lfloor \frac{q}{2} \rfloor$

6: **if** $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$ **then**

7: accept

8: **else**

9: reject

the obvious place

SIEMENS

What the attacker does...

Require: A message m, a signature $\mathtt{sig} = (r, s)$, a public key $\mathtt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

```
c ← HashToPoint(r‖m)                                   ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)                                    ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q                                  ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ‖(s_1, s_2)‖^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```


SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

```
c ← HashToPoint(r||m)                                   ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)                                     ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q                                   ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ||(s_1, s_2)||^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```


SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

```
c ← HashToPoint(r‖m)                                   ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)                                    ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q                                  ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ‖(s_1, s_2)‖^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```


SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

$c \leftarrow \texttt{HashToPoint}(r \| m)$ ▷ $c \in \mathbb{Z}_q[x]$

$s_2 \leftarrow \texttt{Decompress}(s)$ ▷ $s_2 \in \mathbb{Z}[x]$

if $s_2 = \perp$ **then**

    reject

(3) inject fault

$s_1 \leftarrow c - s_2 h \bmod q$ ▷ coefficients of s_1 normalized to be between $\lceil -\frac{q}{2} \rceil$ and $\lfloor \frac{q}{2} \rfloor$

if $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$ **then**

    accept

else

    reject

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

(2) let $s = \texttt{Compress}(0)$

```
c ← HashToPoint(r‖m)                          ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)                           ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q                         ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ‖(s_1, s_2)‖^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```

(3) inject fault

SIEMENS

What the attacker does...

(1) choose message

Require: A message m, a signature $\texttt{sig} = (r, s)$, a public key $\texttt{pk} = h \in \mathbb{Z}_q[x]/(\phi)$

Ensure: Accept or reject

(2) let $s = \texttt{Compress}(0)$

```
c ← HashToPoint(r||m)                          ▷ c ∈ Z_q[x]
s_2 ← Decompress(s)   = 0                      ▷ s_2 ∈ Z[x]
if s_2 = ⊥ then
    reject
s_1 ← c − s_2 h mod q   = 0                    ▷ coefficients of s_1 normalized to be between ⌈−q/2⌉ and ⌊q/2⌋
if ||(s_1, s_2)||^2 ≤ ⌊β^2⌋ then
    accept
else
    reject
```

(3) inject fault

$= 0$

SIEMENS

In code (example from `pqm4` project, `crypto_sign/falcon-512/m4-ct/vrfy.c`)

```
...
/*
 * Compute -s1 = s2*h - c0 mod phi mod q (in tt[]).
 */
mq_NTT(tt, logn);
mq_poly_montymul_ntt(tt, h, logn);
mq_iNTT(tt, logn);
mq_poly_sub(tt, c0, logn);
...
```


SIEMENS

In code (example from `pqm4` project, `crypto_sign/falcon-512/m4-ct/vrfy.c`)

```
...
/*
 * Compute -s1 = s2*h - c0 mod phi mod q (in tt[]).
 */
mq_NTT(tt, logn);
mq_poly_montymul_ntt(tt, h, logn);
mq_iNTT(tt, logn);
mq_poly_sub(tt, c0, logn);
...
```

skip this call with fault injection

SIEMENS

In code (example from `pqm4` project, `crypto_sign/falcon-512/m4-ct/vrfy.c`)

```
...
/*
 * Compute -s1 = s2*h - c0 mod phi mod q (in tt[]).
 */
mq_NTT(tt, logn);
mq_poly_montymul_ntt(tt, h, logn);
mq_iNTT(tt, logn);
mq_poly_sub(tt, c0, logn);
...
```

skip this call with fault injection

or force logn to zero

SIEMENS

In code (example from `pqm4` project, `crypto_sign/falcon-512/m4-ct/vrfy.c`)

```
664     ...
665     /*
666      * Compute -s1 = s2*h - c0 mod phi mod q (in tt[]).
667      */
668     mq_NTT(tt, logn);
669     mq_poly_montymul_ntt(tt, h, logn);
670     mq_iNTT(tt, logn);
671     mq_poly_sub(tt, c0, logn);
672     ...
```

skip this call with fault injection

or force logn to zero

In our setup, the function call in line 671 is inlined by the compiler!

SIEMENS

The inlined function and how to adapt the attack

```
...
0x800d834 <verify_raw+136> bl      0x800d62c <mq_iNTT>
0x800d83a <verify_raw+142> movw    r1, #12289
0x800d83e <verify_raw+146> ldrh.w  r3, [r4, #2]!
0x800d842 <verify_raw+150> ldrh.w  r2, [r6, #2]!
0x800d846 <verify_raw+154> subs    r3, r3, r2
0x800d848 <verify_raw+156> and.w   r2, r1, r3, asr #31
0x800d84c <verify_raw+160> add     r3, r2
0x800d84e <verify_raw+162> cmp     r9, r4
0x800d850 <verify_raw+164> strh    r3, [r4, #0]
0x800d852 <verify_raw+166> bne.n   0x800d83e <verify_raw+146>
...
```


SIEMENS

The inlined function and how to adapt the attack

```
...
0x800d834 <verify_raw+136> bl      0x800d62c <mq_iNTT>
0x800d83a <verify_raw+142> movw    r1, #12289
0x800d83e <verify_raw+146> ldrh.w  r3, [r4, #2]!
0x800d842 <verify_raw+150> ldrh.w  r2, [r6, #2]!
0x800d846 <verify_raw+154> subs    r3, r3, r2
0x800d848 <verify_raw+156> and.w   r2, r1, r3, asr #31
0x800d84c <verify_raw+160> add     r3, r2
0x800d84e <verify_raw+162> cmp     r9, r4
0x800d850 <verify_raw+164> strh    r3, [r4, #0]
0x800d852 <verify_raw+166> bne.n   0x800d83e <verify_raw+146>
...
```

1. Choose signature component r such that first coefficient of $c = \texttt{HashToPoint}(r \| m)$ is 0.
2. Skip `bne.n` with a single fault.

SIEMENS

Countermeasures

Similar as for DILITHIUM:

- Generic countermeasures to ensure control flow integrity.
- Replace

$$s_1 \leftarrow c - s_2 h \bmod q$$

with

$$s_1 \leftarrow (c + u) - (s_2 h + u) \bmod q$$

where u is a random vector.

SIEMENS

Summary

- A successful fault attack against signature verification can completely compromise the security of a device.
- There are places in signature verification functions which are obvious targets for a fault attack.
- An experienced developer will protect these places with suitable countermeasures.
- There are also less obvious places to attack.
- These are easily overlooked when hardening the code of a signature verification function.
- We have identified some places like this in DILITHIUM and FALCON.

SIEMENS

Contact

Sven Bauer, Fabrizio De Santis
Siemens AG Technology

Otto-Hahn-Ring 6
81739 München
Germany

E-mail svenbauer[at]siemens.com, fabrizio.desantis[at]siemens.com

SIEMENS