Single-Trace Side-Channel Attacks on CRYSTALS-Dilithium: Myth or Reality?

Ruize Wang, **Kalle Ngo**, Joel Gärtner, and Elena Dubrova

KTH Royal Institute of Technology, Stockholm Sweden

{ruize,kngo,jgartner,dubrova}@kth.se

CRYSTALS-Dilithium

- A Digital Signature Scheme part of the CRYSTALS cryptographic suit
- In July of 2022, it was announced that Dilithium will be standardized as ML-DSA
- Secret Key, Public Key, message, signature
- Keygen, Sign, verify functions

Side Channel Analysis

- “In computer security, a **side-channel attack** is any attack based on extra information that can be gathered because of the fundamental way a computer algorithm is implemented, rather than flaws in the design of the algorithm itself” –Wikipedia
- These leakages can occur in a variety of methods including:
  - Timing, cache,
  - Power,
  - Electromagnetic,
  - Acoustic, etc.

Previous Attacks

- Previous attacks on Software implementations
  - Largely focus on recovery on secret key vector $\mathbf{s}_1$
  - Usually during the signing procedure

- Secret key consist of 2 secret vector $\mathbf{s}_1$ and $\mathbf{s}_2$
  - Work has done to show that with some form of forgeries can be done with only knowledge of $\mathbf{s}_1$

Attacks on Signing

- $\mathbf{s}_1$ and $\mathbf{s}_2$ are the secrets
- Manipulation with secrets occur in specific lines in algorithm
- Commonly accepted that recovery of $\mathbf{s}_1$ is sufficient for some form of forgery
- Line 12: z=y + c$\mathbf{s}_1$ is the target of most attacks and therefore also countermeasures

$\mathsf{Sign}(sk, M)$

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) = \mathsf{skDecode}(sk)$
2: $\mathbf{A} \in R_q^{k\times\ell} = \mathsf{ExpandA}(\rho)$
3: $\mu \in \{0,1\}^{512} = \mathsf{H}(tr \mid\mid M)$
4: $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
5: $\rho' \leftarrow \{0,1\}^{512}$
6: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
7: $\quad \mathbf{y} = \mathsf{ExpandMask}(\rho', \kappa)$
8: $\quad \mathbf{w} = \mathbf{A}\mathbf{y}$
9: $\quad \mathbf{w}_1 = \mathsf{HighBits}(\mathbf{w}, 2\gamma_2)$
10: $\quad \tilde{c} \in \{0,1\}^{256} = \mathsf{H}(\mu \mid\mid \mathbf{w}_1)$
11: $\quad c = \mathsf{SampleInBall}(\tilde{c})$
12: $\quad \mathbf{z} = \mathbf{y} + c\mathbf{s}_1$
13: $\quad \mathbf{r}_0 = \mathsf{LowBits}(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
14: $\quad$ **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ or $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) = \perp$
15: $\quad$ **else**
16: $\qquad \mathbf{h} = \mathsf{MakeHint}(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
17: $\qquad$ **if** $||c\mathbf{t}_0||_\infty \geq \gamma_2$ or # of 1's in $\mathbf{h}$ is $> \omega$ **then** $(\mathbf{z}, \mathbf{h}) = \perp$
18: $\quad \kappa = \kappa + \ell$
19: **return** $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Unpacking

- Currently overlooked manipulation of secrets
- Dilithium stores key coefficients as ie.
  - -2,-1,0,1,2 (for Dilithium-2 and Dilithium-5)
- 5 unique values therefore can be represented as a 3 bit value
- We target partial recovery of $\mathbf{s}_1$ and $\mathbf{s}_2$ and solve for the full key in post processing

```
void small_polyeta_unpack(smallpoly *r, uint8_t *a)
/* a is the input byte array of s1 or s2 in the secret key*/
/* r is the corresponding output polynomial coefficients of of s1 or s2*/
unsigned int i;
for (i = 0; i < N/8; ++i) do /* N = 256, ETA = 2 in Dilithium-2 */
    r->coeffs[8*i+0] = (a[3*i+0] >> 0) & 7;
    r->coeffs[8*i+1] = (a[3*i+0] >> 3) & 7;
    r->coeffs[8*i+2] = ((a[3*i+0] >> 6) | (a[3*i+1] << 2)) & 7;
    r->coeffs[8*i+3] = (a[3*i+1] >> 1) & 7;
    r->coeffs[8*i+4] = (a[3*i+1] >> 4) & 7;
    r->coeffs[8*i+5] = ((a[3*i+1] >> 7) | (a[3*i+2] << 1)) & 7;
    r->coeffs[8*i+6] = (a[3*i+2] >> 2) & 7;
    r->coeffs[8*i+7] = (a[3*i+2] >> 5) & 7;

    r->coeffs[8*i+0] = ETA - r->coeffs[8*i+0];
    r->coeffs[8*i+1] = ETA - r->coeffs[8*i+1];
    r->coeffs[8*i+2] = ETA - r->coeffs[8*i+2];
    r->coeffs[8*i+3] = ETA - r->coeffs[8*i+3];
    r->coeffs[8*i+4] = ETA - r->coeffs[8*i+4];
    r->coeffs[8*i+5] = ETA - r->coeffs[8*i+5];
    r->coeffs[8*i+6] = ETA - r->coeffs[8*i+6];
    r->coeffs[8*i+7] = ETA - r->coeffs[8*i+7];
end for
```

Key Recovery

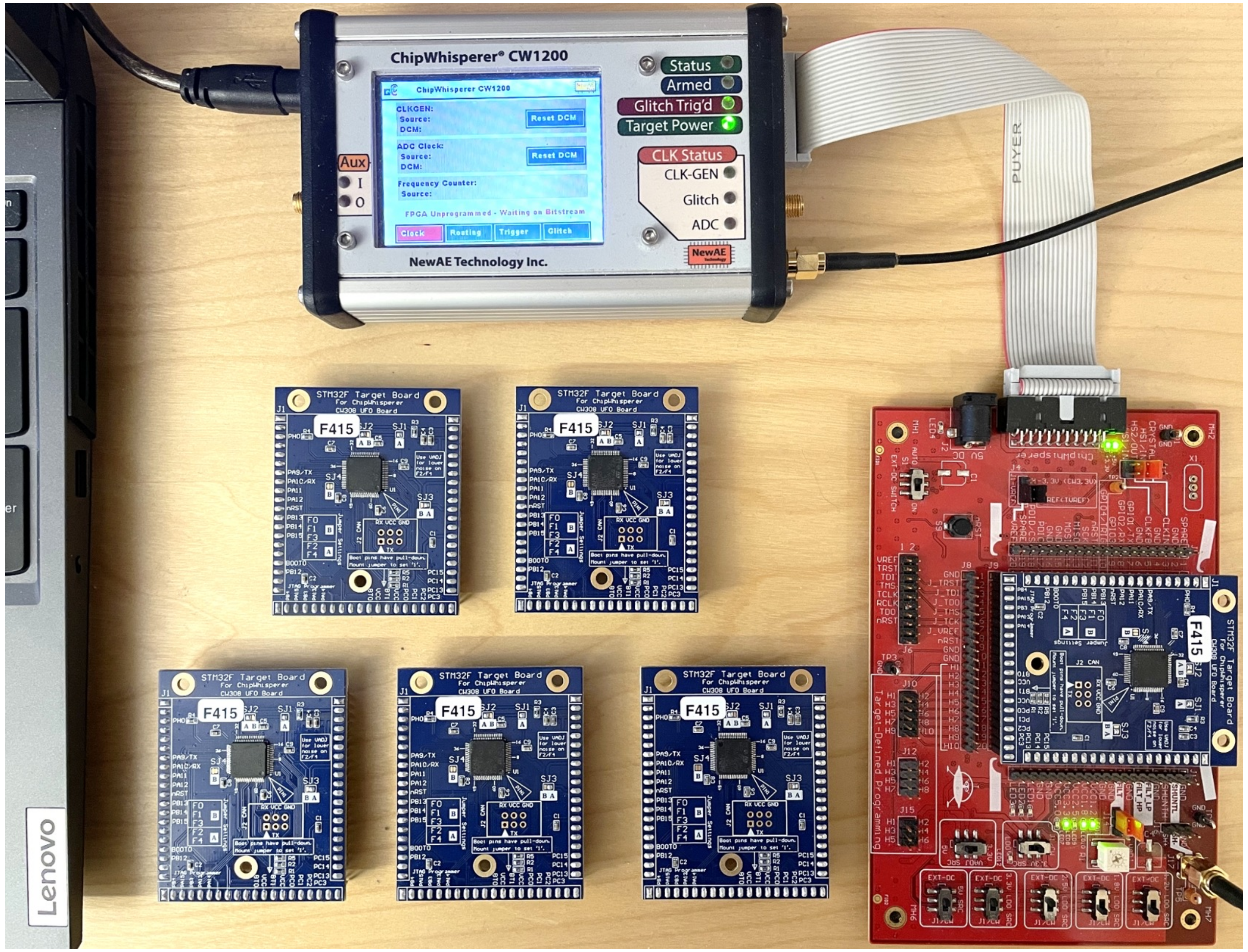
- Key unpacking has no user input (non-differential attack)
- We target partial recovery of $\mathbf{s}_1$ and $\mathbf{s}_2$ and then solve for the full key in post processing
- “This compression is an optimization for performance, not security. The low order bits of $\mathbf{t}$ can be reconstructed from a small number of signatures and, therefore, need not be regarded as secret. ”

Two Methods:

- Simple Linear Algebra – Assuming knowledge of the low order bits of $\mathbf{t}$, $\mathbf{t}_0$
  - $\mathbf{t} = A\mathbf{s}_1 + \mathbf{s}_2$
- Lattice Reduction – No knowledge of $\mathbf{t}$ required
  - Recover a larger portion of $\mathbf{s}_1$ and then solve

Experiment

- Equipment
    - Chipwhisperer
    - ST's STM32F415 ARM Cortex-M4
- Publicly available Dilithium-2 implementation by Abdulrahman et al.
- 5 profiling devices
- 1 attack device



Trace Capture

- Profiling traces to train neural network can take up to 5 hours
  - Under the attacker's control
  - Does NOT have the secret key
  - Train on randomly generated keys
- Test set are traces from the DUT
  - Never seen by the neural network during training
  - Attacker needs up to 6 minutes with it

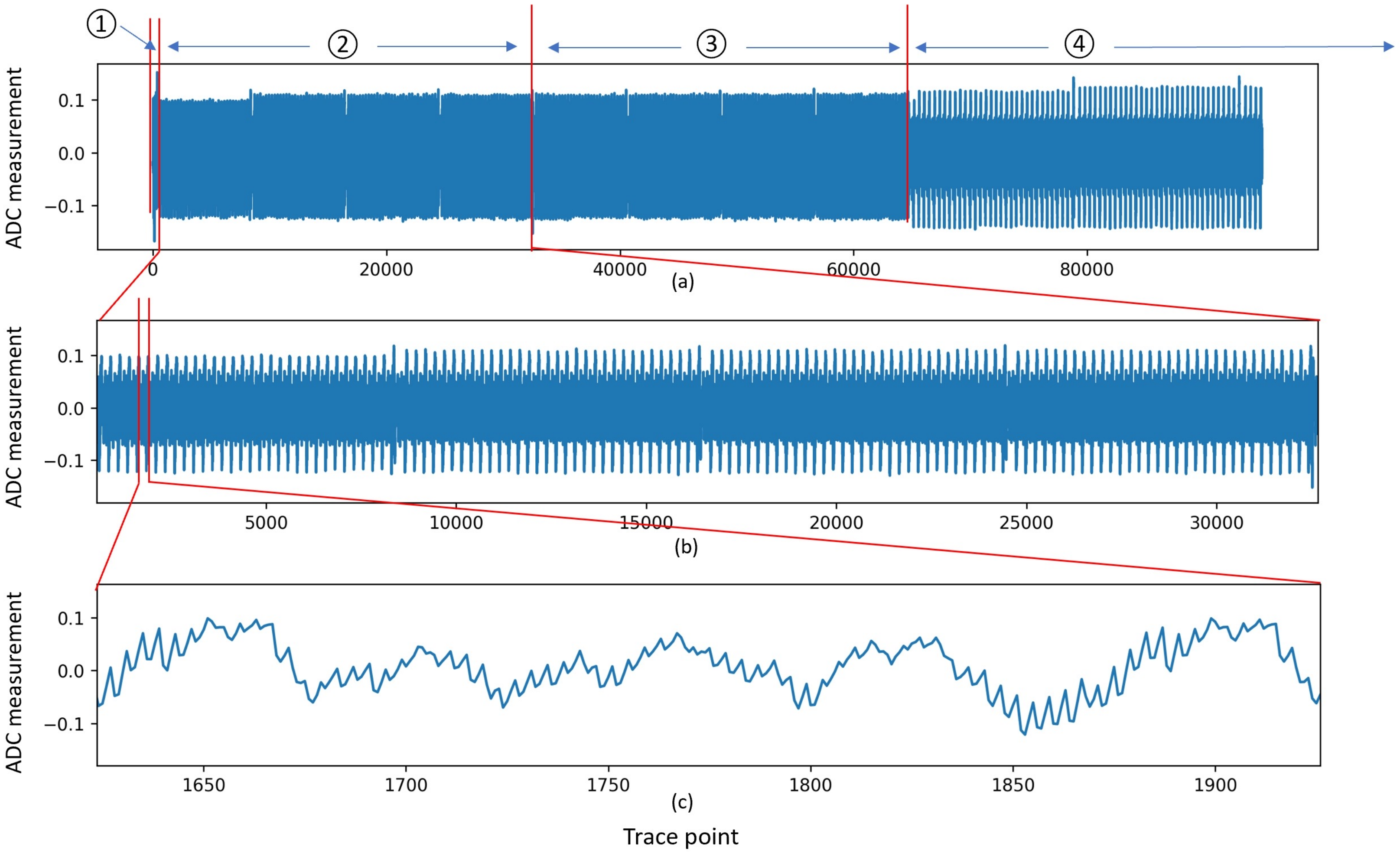
Training set	Time for capturing 5×2.5K traces
	4.8 hrs

Test set	Time for capturing N traces		
	$N = 1$	$N = 100$	$N = 1000$
	1.2 sec	36.6 sec	358.2 sec

Trace

- The unpack_sk() procedure

1. ρ,tr and K
2. s_1
3. s_2
4. t_0



Coefficient Recovery

- Train 8 NN models (40 mins laptop PC)
- With each random key, sign random message while capturing trace.

```
void small_polyeta_unpack(smallpoly *r, uint8_t *a)
/* a is the input byte array of s1 or s2 in the secret key*/
/* r is the corresponding output polynomial coefficients of of s1 or s2*/
unsigned int i;
for (i = 0; i < N/8; ++i)  do /* N = 256, ETA = 2 in Dilithium-2 */
    r->coeffs[8*i+0] = (a[3*i+0] >> 0) & 7;
    r->coeffs[8*i+1] = (a[3*i+0] >> 3) & 7;
    r->coeffs[8*i+2] = ((a[3*i+0] >> 6) | (a[3*i+1] << 2)) & 7;
    r->coeffs[8*i+3] = (a[3*i+1] >> 1) & 7;
    r->coeffs[8*i+4] = (a[3*i+1] >> 4) & 7;
    r->coeffs[8*i+5] = ((a[3*i+1] >> 7) | (a[3*i+2] << 1)) & 7;
    r->coeffs[8*i+6] = (a[3*i+2] >> 2) & 7;
    r->coeffs[8*i+7] = (a[3*i+2] >> 5) & 7;

    r->coeffs[8*i+0] = ETA - r->coeffs[8*i+0];
    r->coeffs[8*i+1] = ETA - r->coeffs[8*i+1];
    r->coeffs[8*i+2] = ETA - r->coeffs[8*i+2];
    r->coeffs[8*i+3] = ETA - r->coeffs[8*i+3];
    r->coeffs[8*i+4] = ETA - r->coeffs[8*i+4];
    r->coeffs[8*i+5] = ETA - r->coeffs[8*i+5];
    r->coeffs[8*i+6] = ETA - r->coeffs[8*i+6];
    r->coeffs[8*i+7] = ETA - r->coeffs[8*i+7];
end for
```

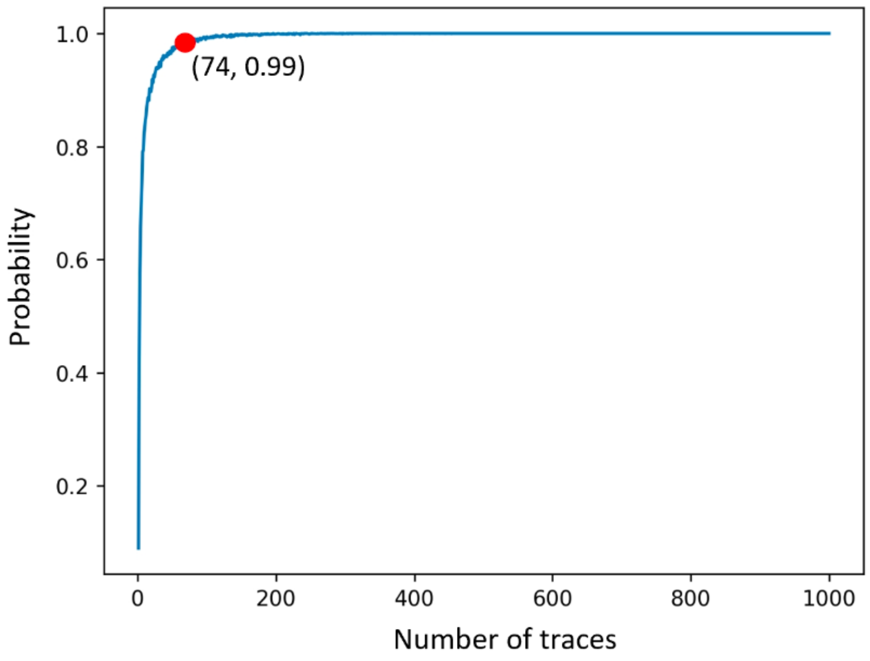
Table 3: Empirical probability to recover a single coefficient of $\mathbf{s}_1$, $\mathbf{s}_1[j]$, by power analysis using N traces; each entry in the middle column is a mean probability over all $\mathbf{s}_1[j]$ with the same j mod 8, for $j \in \{0, 1, \cdots, 1023\}$.

N	j mod 8								Avg.
	0	1	2	3	4	5	6	7	
1	0.942	0.925	0.975	0.967	0.945	0.920	0.924	0.784	0.923
10	0.980	0.951	0.999	0.993	0.988	0.951	0.956	0.863	0.960
100	0.983	0.952	0.999	0.995	0.991	0.954	0.959	0.870	0.963
1000	0.983	0.954	0.999	0.995	0.991	0.956	0.960	0.871	0.964

Linear Algebra

Probability to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$				LA post-processing
$N = 1$	$N = 10$	$N = 100$	$N = 1000$	CPU time
0.09	0.83	0.99	1	2 sec

- system of the linear equations
  - $\mathbf{t} = A\mathbf{s}_1 + \mathbf{s}_2$
- We can observe the NN output score vector to infer more information

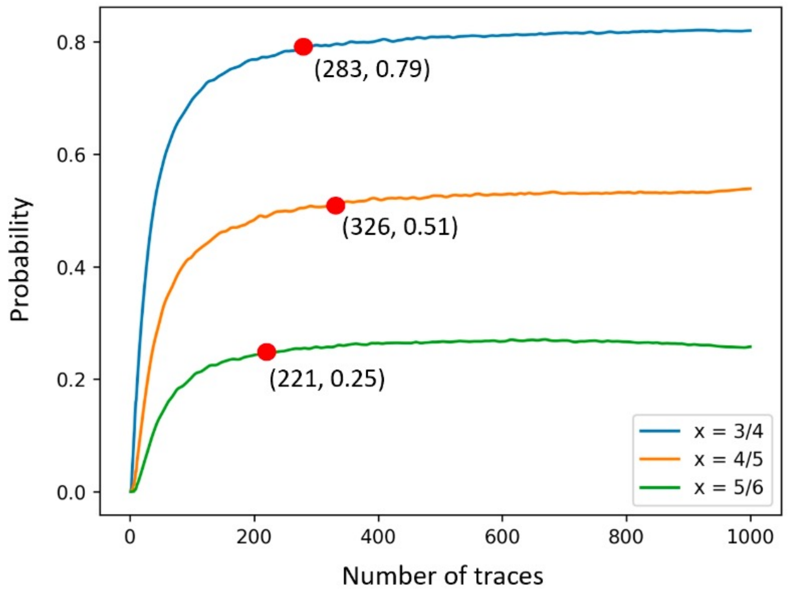


Lattice Reduction

- With 4/5 of s1 recovered by power analysis
  - 6 hours for BKZ Lattice reduction

- With 6 mins of access to DUT, attacker has 54% chance recovering Secret Key through Lattice Reduction in 6 hours

Fraction x	Probability to recover $\lfloor x \cdot 1024 \rfloor$ coeff. of $\mathbf{s}_1$				BKZ post-processing	
	$N = 1$	$N = 10$	$N = 100$	$N = 1000$	β	α
3/4	0	0.16	0.70	0.82	160	46.7
4/5	0	0.04	0.42	0.54	115	33.6
5/6	0	0.01	0.20	0.26	86	25.2



Conclusion

- Consider that the unpacking function also needs protection
- Countermeasures:
  - Statically masking the stored key
  - Constant-weight encoding
  - Shuffling the key unpacking loop