

# **Side-Channel Analysis Attack on Kyber PWM**

**IITK SCA Competition - Three Comprehensive Solutions**

Solution	Technique	Innovation Level
Solution 1	Deep Learning CNN + Attention	High
Solution 2	Multi-Model CPA with POI Detection	Medium-High
Solution 3	Transformer + Contrastive Learning	Very High

# Table of Contents

- 1. Problem Overview and Challenge Description
- 2. Dataset Analysis
- 3. Solution 1: Deep Learning-Based SCA Attack
- 4. Solution 2: Advanced Correlation Power Analysis (CPA)
- 5. Solution 3: Differential Deep Learning with Transformers
- 6. Comparison and Recommendations
- 7. Implementation Guide
- Appendix A: Complete Source Code

# 1. Problem Overview and Challenge Description

## 1.1 Introduction to Kyber (ML-KEM)

Kyber is a lattice-based key encapsulation mechanism (KEM) selected by NIST for post-quantum cryptography standardization. It relies on the hardness of the Module Learning with Errors (MLWE) problem. The security of Kyber depends on keeping the secret polynomial vector S1 confidential.

## 1.2 Attack Target: Pointwise Multiplication (PWM)

The target operation is the polynomial pointwise multiplication  $S1 \cdot r$ , where  $S1$  is the secret polynomial and  $r$  is the ciphertext polynomial. The PWM operation processes pairs of coefficients using Montgomery multiplication and twiddle factors.

## 1.3 Challenge Objective

Recover 19 specific coefficients of the secret key  $S1$  at indices: 0, 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196, 210, 224, 238, 252. These indices follow a pattern of every 14th coefficient starting from 0.

## 1.4 PWM Algorithm

The algorithm processes two coefficients at a time computing Montgomery multiplications and modular operations. Key intermediate values that leak through power consumption include:

- $m0 = \text{Montgomery}(A[i], B[i])$  - Primary multiplication target
- $m1 = \text{Montgomery}(A[i+1], B[i+1])$  - Secondary multiplication
- $m3 = \text{Montgomery}(m1, \omega[i/2])$  - Twiddle factor multiplication
- $\text{OUT}[i] = (m0 + m3) \bmod 3329$  - Final output coefficient

## 2. Dataset Analysis

### 2.1 Power Traces (merged\_final\_dataset\_V1.npy)

Property	Value
Shape	20,000 traces × 10,000 time samples
Data Type	float64
Value Range	[-0.058, 0.032]
Mean	-0.003
Std Deviation	0.008

### 2.2 Polynomial Coefficients (polynomials\_final.csv)

Property	Value
Shape	20,000 polynomials × 256 coefficients
Data Type	int64
Value Range	[0, 3328] (mod q=3329)
Unique Values	3329 (full modulus range)

### 2.3 Key Parameters

- Kyber Modulus (q): 3329
- Montgomery Parameter (R):  $2^{12} \text{ mod } 3329$
- Polynomial Degree: 256
- Hardware: ChipWhisperer CW305

## 3. Solution 1: Deep Learning-Based SCA Attack

### 3.1 Overview

This solution employs Convolutional Neural Networks (CNNs) combined with multi-head self-attention mechanisms to extract discriminative features from power traces. The approach is inspired by recent advances in deep learning for side-channel analysis, particularly the work on profiling attacks.

### 3.2 Architecture

The KyberSCANet architecture consists of three main components:

- Convolutional Feature Extractor: Four ConvBlocks with increasing channels ( $64 \rightarrow 128 \rightarrow 256 \rightarrow 256$ ), batch normalization, SELU activation, and average pooling
- Multi-Head Self-Attention: Two attention layers with 8 heads to capture temporal dependencies
- Classification Head: Three fully connected layers with dropout for regularization

### 3.3 Key Innovations

1. SELU Activation: Self-normalizing activation for stable training
2. Attention Mechanism: Captures long-range dependencies in power traces
3. Residual Connections: Enhanced variant (KyberSCANetV2) with skip connections
4. Adaptive Pooling: Handles variable trace lengths efficiently

### 3.4 Leakage Models

The solution implements multiple leakage models to capture different aspects of power consumption:

- Hamming Weight (HW): Counts the number of 1-bits in intermediate values
- Identity: Direct value leakage model
- LSB/MSB: Focuses on least/most significant bits

### 3.5 Attack Strategy

For unprofiled scenarios (secret key unknown), the solution uses correlation-based key recovery: for each key hypothesis, compute expected leakage and correlate with actual traces. The key with highest correlation is selected. For profiled scenarios, the neural network directly classifies traces to recover key coefficients.

## 4. Solution 2: Advanced Correlation Power Analysis (CPA)

### 4.1 Overview

This solution implements a sophisticated Correlation Power Analysis attack with Point of Interest (POI) detection, multiple leakage models, and parallel processing for efficiency. CPA is a classical but highly effective technique that exploits the statistical correlation between hypothetical power consumption and actual measured traces.

### 4.2 Mathematical Foundation

CPA computes Pearson correlation coefficient between hypothetical leakage  $H$  and measured traces  $T$ :

$$\rho(H, T) = \text{Cov}(H, T) / (\sigma_H \times \sigma_T)$$

The correct key hypothesis produces the highest correlation because the hypothetical leakage matches the actual power consumption pattern.

### 4.3 POI Detection Methods

- SNR (Signal-to-Noise Ratio): Identifies time samples with high signal variance relative to noise
- SOST (Sum of Squared T-statistics): Statistical test for class separation
- Variance Analysis: Unsupervised detection based on trace variability
- Difference of Means: Partitions traces and finds distinguishing points

### 4.4 Multi-Model Attack

The MultiModelCPA class combines results from multiple leakage models (HW, identity, LSB, MSB) and multiple intermediate targets ( $m_0, m_1$ ) using a voting and weighted scoring mechanism. This ensemble approach improves robustness against model mismatch.

### 4.5 Optimization Techniques

- Parallel Processing: Uses joblib for multi-core correlation computation
- POI Selection: Reduces dimensionality from 10,000 to ~2,000 informative samples
- Incremental Computation: Efficient online correlation updates
- Memory Management: Processes traces in batches to handle large datasets

## 5. Solution 3: Differential Deep Learning with Transformers

### 5.1 Overview

This cutting-edge solution combines transformer architectures (inspired by Vision Transformers) with contrastive pre-training and differential analysis. It represents the state-of-the-art in deep learning for side-channel analysis, leveraging self-supervised learning to extract robust features without labeled data.

### 5.2 Transformer Architecture

The KyberTransformerSCA model adapts the Vision Transformer (ViT) architecture for power traces:

- Patch Embedding: Divides 10,000-sample trace into 100 patches of 100 samples each
- Positional Encoding: Sinusoidal encoding to preserve temporal information
- CLS Token: Learnable token for aggregating global information
- Transformer Encoders: 6 layers with 8-head self-attention and feed-forward networks
- Classification Head: Projects CLS token to key space (3329 classes)

### 5.3 Contrastive Pre-training

Before supervised training, the model is pre-trained using contrastive learning (SimCLR-style). This self-supervised approach learns meaningful representations by:

1. Creating augmented views of each trace (noise, scaling, shifting)
2. Training the model to maximize similarity between views of the same trace
3. Minimizing similarity between different traces
4. Using NT-Xent loss with temperature scaling

### 5.4 Hybrid CPA-DL Attack

The solution includes a hybrid approach that combines classical CPA with deep learning:

- Phase 1: CPA identifies top-k candidate keys based on correlation
- Phase 2: Mutual information analysis verifies candidates
- Phase 3: Deep learning model refines selection among candidates
- Final: Combined scoring selects the most likely key

### 5.5 Cluster-Based Pseudo-Labeling

For unsupervised scenarios, the solution generates pseudo-labels through clustering: traces are clustered using K-means on PCA-reduced features, and each cluster is assigned a key hypothesis based on intra-cluster CPA. This enables supervised training without ground truth labels.

## 6. Comparison and Recommendations

### 6.1 Solution Comparison Matrix

Criterion	Solution 1	Solution 2	Solution 3
Computational Cost	High	Medium	Very High
Data Efficiency	Medium	High	Medium
Noise Robustness	High	Medium	Very High
Implementation Complexity	Medium	Low	High
Innovation Level	High	Medium	Very High
GPU Required	Yes	No	Yes
Profiled Attack	Yes	No	Yes
Unprofiled Attack	Yes	Yes	Yes

### 6.2 Recommendations

- **For Quick Results:** Use Solution 2 (CPA) - fastest to implement and run, requires no GPU, and provides competitive results.
- **For Best Accuracy:** Use Solution 3 (Transformer) - state-of-the-art architecture with contrastive pre-training offers best noise robustness.
- **For Balanced Approach:** Use Solution 1 (CNN+Attention) - good trade-off between computational cost and accuracy, easier to tune than transformers.
- **For Ensemble:** Combine all three solutions - run CPA first for quick candidates, then verify with DL models for highest confidence.

## 7. Implementation Guide

### 7.1 Requirements

- Python 3.8+
- NumPy, Pandas, SciPy
- PyTorch 1.10+ (with CUDA for GPU acceleration)
- scikit-learn
- joblib (for parallel processing)
- tqdm (for progress bars)
- matplotlib (for visualization)

### 7.2 Installation

```
pip install numpy pandas scipy torch scikit-learn joblib tqdm matplotlib
```

### 7.3 Running the Solutions

```
# Solution 1: Deep Learning Attack
python solution1_deep_learning_sca.py

# Solution 2: CPA Attack
python solution2_cpa_attack.py

# Solution 3: Transformer Attack
python solution3_differential_deep_learning.py
```

### 7.4 Output Format

Each solution outputs a CSV file with recovered keys in the format:

Index	Recovered_Key	Correlation/Confidence
0	xxxx	0.xxxx
14	xxxx	0.xxxx
...	...	...
252	xxxx	0.xxxx

# Appendix A: Key Code Snippets

## A.1 Montgomery Multiplication

```
def montgomery_multiply(a, b):
    '''Montgomery multiplication: a * b * R^-1 mod q'''
    Q = 3329 # Kyber modulus
    R_INV = pow(pow(2, 12, Q), -1, Q)
    return (a * b * R_INV) % Q
```

## A.2 Hamming Weight Leakage Model

```
def hamming_weight(x):
    '''Compute Hamming weight (number of 1s)'''
    return bin(int(x) & 0xFFFF).count('1')
```

## A.3 CPA Core Function

```
def cpa_attack(traces, r_coeffs, target_idx):
    correlations = np.zeros(Q)
    for s_guess in range(Q):
        hyp_leak = [hamming_weight(
            montgomery_multiply(s_guess, r_coeffs[i]))
            for i in range(len(r_coeffs))]
        corr = np.corrcoef(hyp_leak, traces.T)[0, 1:]
        correlations[s_guess] = np.max(np.abs(corr))
    return np.argmax(correlations)
```

## A.4 Transformer Attention

```
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, embed_dim, num_heads=8):
        super().__init__()
        self.attention = nn.MultiheadAttention(
            embed_dim, num_heads, batch_first=True
        )
        self.norm = nn.LayerNorm(embed_dim)

    def forward(self, x):
        attn_out, _ = self.attention(x, x, x)
        return self.norm(x + attn_out)
```