National College of Ireland

Configuration Manual

MSc Research Project

Mitigating Privacy Risks in IoT Healthcare Wearables: Simulated Side-Channel Attacks and Defense via Lightweight Encryption Techniques.

Amaka Ngozi Obibueze
Student ID: x23324431

School of Computing
National College of Ireland

Supervisor: x23324431

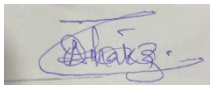
National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Amaka Ngozi Obibueze
Student ID:	X23324431
Programme:	Msc Cybersecurity -Group B **Year:** 2025
Module:	Khadija Hafeez
Lecturer:	
Submission Due Date:	11th August 2025
Project Title:	Practicum 2
Word Count:	**Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Amaka Ngozi Obibueze
Student ID: x23324431

Introduction

This configuration manual provides step-by-step instructions for recreating the IoT wearable security research environment on a new machine. The project evaluates side-channel vulnerabilities of the Simeck32/64 lightweight block cipher and assesses masking-based countermeasures through simulation and analysis scripts written in Python.

1 System Configuration

Machine Configuration:

S/n	Component	Requirement Used
1	Operating System	Windows 11 Pro
2	RAM	16GB
3	Disk Storage	276GB (free space)
4	Graphics Card	128MB
5	Processor	Intel® Core ™ i7-8650U CPU @ 1.90GHz
6	System Type	64-bit operating system, x64-based processor
7	Deepnote	Deepnote for my Implementation

Software stack overview

A. Python Environment:

Python 3.11 (project tested with 3.12)

Dependencies listed in requirements.txt

- NumPy,
- SciPy,
- scikit-learn,
- matplotlib,
- Jupyter.

  Deepnote is a cloud-based Jupyter notebook environment used to run Python programs in research.

Detailed Implementation Walkthrough

The project was implemented in the following stages. “src” represents source code in the explanation below.

1. **The Simulated Environment set up**
   Simulation Environment Setup ARM Cortex-M4 microcontroller operating at 80 MHz Python and its scientific computing libraries. Computer gadget set up and virtual environment.

2. **The Baseline (Unmasked) implementation of the Simeck32/64 cipher (src/Simeck cipher.py):**
   This is a basic lightweight block cipher that operates on 32-bit blocks using a 64-bit key over 32 rounds. The base is an unprotected version of the Simeck32/64 encryption algorithm used for encrypting data, which lacks any security defenses.

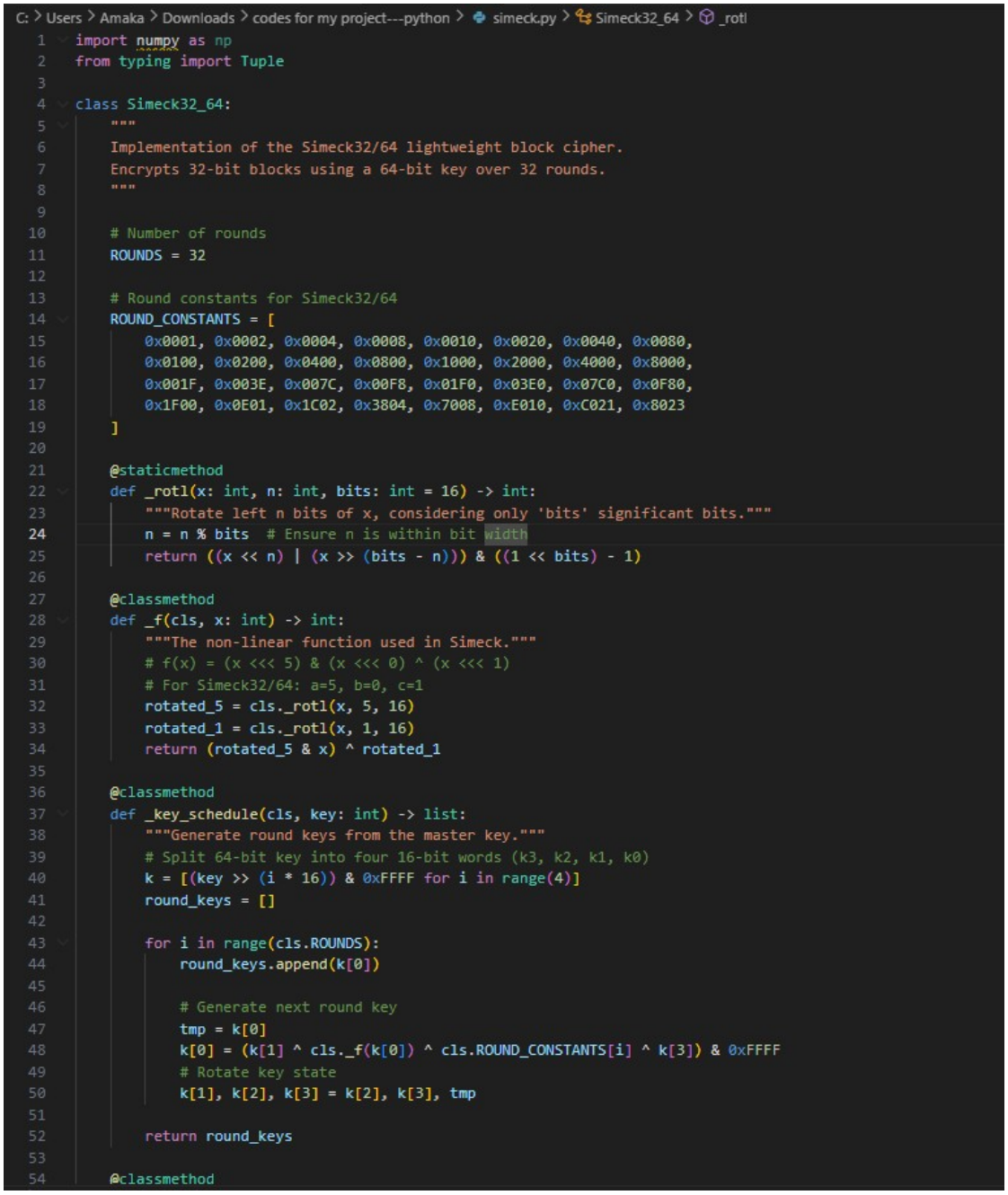


Figure 1a: Implementation of the Simeck32/64 cipher.

```python
    def hamming_weight(cls, x: int, bits: int = 32) -> int:
        """
        Calculate the Hamming weight (number of set bits) of an integer.

        Args:
            x: Integer to calculate Hamming weight for
            bits: Number of bits to consider

        Returns:
            Hamming weight (number of 1s in binary representation)
        """
        return bin(x & ((1 << bits) - 1)).count('1')

    @classmethod
    def encrypt_with_leakage(cls, plaintext: int, key: int) -> Tuple[int, list]:
        """
        Encrypt plaintext and simulate side-channel leakage (Hamming weights).

        Args:
            plaintext: 32-bit integer to encrypt
            key: 64-bit encryption key

        Returns:
            Tuple of (ciphertext, leakage_traces) where leakage_traces is a list of
            Hamming weights of intermediate values during encryption
        """
        x = (plaintext >> 16) & 0xFFFF
        y = plaintext & 0xFFFF
        round_keys = cls._key_schedule(key)

        leakage = []

        # Initial state leakage
        leakage.append(cls.hamming_weight(x, 16) + cls.hamming_weight(y, 16))

        for i in range(cls.ROUNDS):
            # Leakage before round function
            leakage.append(cls.hamming_weight(x, 16))

            # Apply round function and key addition
            tmp = x
            f_out = cls._f(x)
            leakage.append(cls.hamming_weight(f_out, 16))

            x = y ^ f_out ^ round_keys[i]
            y = tmp

            # Leakage after round function
            leakage.append(cls.hamming_weight(x, 16) + cls.hamming_weight(y, 16))

        ciphertext = (x << 16) | y
        return ciphertext, leakage
```

Figure 1b: Implementation of the Simeck32/64 cipher.

3. **<u>Side-Channel Leakage Simulation (src/side_channel.py)</u>:**
   This stage demonstrates how side-channel attacks work by simulating them against the Simeck32/64 encryption method. The Side Channel Analyzer simulates power analysis attacks by collecting 1000 simulated power consumption traces during the encryption process. Afterward, it applies Differential Power Analysis (DPA) coupled with linear

regression to correlate the simulated physical leakage to plaintext bits. In this simulation, the Hamming-weight power model and configurable noise sources were added. Synthetic traces for each intermediate state were generated.

C: > Users > Amaka > Downloads > codes for my project---python > side_channel.py > ...

```python
"""
Side-channel analysis and countermeasures for Simeck32/64 implementation.
"""
import numpy as np
from typing import List, Tuple, Dict, Optional
from sklearn.linear_model import LinearRegression
from .simeck import Simeck32_64

class SideChannelAnalyzer:
    """
    Performs side-channel analysis on the Simeck32/64 implementation.
    """

    def __init__(self, num_traces: int = 1000):
        """
        Initialize the side-channel analyzer.

        Args:
            num_traces: Number of traces to collect for analysis
        """
        self.num_traces = num_traces
        self.traces = []
        self.plaintexts = []
        self.ciphertexts = []

    def collect_traces(self, key: int) -> None:
        """
        Collect side-channel traces by encrypting random plaintexts.

        Args:
            key: 64-bit encryption key to use for collection
        """
        self.traces = []
        self.plaintexts = []
        self.ciphertexts = []

        for _ in range(self.num_traces):
            # Generate random plaintext
            pt = np.random.randint(0, 2**32, dtype=np.uint32)

            # Encrypt and collect leakage
            ct, leakage = Simeck32_64.encrypt_with_leakage(pt, key)

            self.plaintexts.append(pt)
```

Figure 2a: Side Channel Leakage Simulation.

```
            self.traces.append(leakage)

    def perform_DPA_attack(self, target_bit: int, max_features: int = 10) -> Dict:
        """
        Perform a Differential Power Analysis attack on the collected traces.

        Args:
            target_bit: Which bit of the key to target (0-63)
            max_features: Maximum number of features to use in the model

        Returns:
            Dictionary containing attack results and metrics
        """
        if not self.traces:
            raise ValueError("No traces collected. Call collect_traces() first.")

        # Prepare features (Hamming weights at different points in time)
        X = np.array(self.traces)

        # Target is the target bit of the plaintext
        # We'll predict each bit of the plaintext to find correlations with leakage
        if target_bit >= 32:
            raise ValueError("target_bit must be between 0 and 31")

        # Get the target bit from each plaintext (0 or 1)
        y = np.array([(pt >> target_bit) & 1 for pt in self.plaintexts])

        # Use linear regression to find correlations
        model = LinearRegression()
        model.fit(X, y)

        # Get the most important features
        if hasattr(model, 'coef_'):
            top_features = np.argsort(np.abs(model.coef_))[-max_features:]
        else:
            top_features = []

        return {
            'model': model,
            'score': model.score(X, y),
            'top_features': top_features.tolist() if len(top_features) > 0 else [],
            'coefficients': model.coef_.tolist() if hasattr(model, 'coef_') else []
        }
```

Figure 2b: Side Channel Leakage Simulation.

4). Dataset Generation (src/run_analysis.py)

- **run_analysis.py**- this generates synthetic power traces and runs the full analysis pipeline. This also creates data/synthetic/ and writes HDF5 traces + JSON metadata.

C: > Users > Amaka > Downloads > codes for my project---python > run_analysis_simple.py > ...

```python
#!/usr/bin/env python3
"""
Simple script to generate dataset and run analysis.
"""

import sys
import os
import json
import h5py
import numpy as np
from pathlib import Path
from tqdm import tqdm

# Add the parent directory to the path to import simeck
sys.path.append(str(Path(__file__).parent.parent))

from src.simeck import Simeck32_64

def generate_dataset(num_traces=1000, trace_length=100, noise_level=0.1):
    """Generate synthetic power traces for Simeck32/64 encryption."""
    print(f"Generating {num_traces} traces...")

    traces = np.zeros((num_traces, trace_length), dtype=np.float32)
    metadata = []

    for i in tqdm(range(num_traces), desc="Generating traces"):
        # Generate random plaintext and key
        pt = np.random.randint(0, 2**32, dtype=np.uint32)
        key = np.random.randint(0, 2**64, dtype=np.uint64)

        # Encrypt and get leakage
        _, leakage = Simeck32_64.encrypt_with_leakage(pt, key)

        # Pad or truncate leakage to desired length
        if len(leakage) > trace_length:
            leakage = leakage[:trace_length]
        elif len(leakage) < trace_length:
            leakage = np.pad(leakage, (0, trace_length - len(leakage)), 'edge')

        # Add noise and store
        traces[i] = leakage + np.random.normal(0, noise_level, trace_length)

        # Store metadata
        metadata.append({
```

Figure 3a: Dataset Generation.

C: > Users > Amaka > Downloads > codes for my project---python > run_analysis_simple.py > ...

```python
def generate_dataset(num_traces=1000, trace_length=100, noise_level=0.1):
                'plaintext': int(pt),
                'ciphertext': int(Simeck32_64.encrypt(pt, key)),
                'key': int(key),
                'leakage_points': [int(x) for x in leakage]
            })

    return traces, metadata

def save_dataset(traces, metadata, output_dir):
    """Save traces and metadata to disk."""
    output_dir = Path(output_dir)
    output_dir.mkdir(parents=True, exist_ok=True)

    # Save traces to HDF5
    traces_path = output_dir / 'traces.h5'
    with h5py.File(traces_path, 'w') as f:
        f.create_dataset('traces', data=traces)
    print(f"Saved traces to {traces_path}")

    # Save metadata to JSON
    metadata_path = output_dir / 'metadata.json'
    with open(metadata_path, 'w') as f:
        json.dump(metadata, f)
    print(f"Saved metadata to {metadata_path}")

def main():
    # Set up paths
    data_dir = Path("data/synthetic")
    data_dir.mkdir(parents=True, exist_ok=True)

    # Generate and save dataset
    traces, metadata = generate_dataset()
    save_dataset(traces, metadata, data_dir)

    print("\nDataset generation complete!")
    print(f"- Number of traces: {len(traces)}")
    print(f"- Trace length: {traces.shape[1]}")
    print(f"- Output directory: {data_dir.absolute()}")

if __name__ == "__main__":
    main()
```

Figure 3b: Dataset Generation.

5). <u>Implementation of Countermeasures (Masking Defense)</u>

Implemented first- and second-order Boolean masking using ISW scheme. Mask-refreshing was added to thwart higher-order attacks.

For First-order and Second-order masked implementation:

C: > Users > Amaka > Downloads > codes for my project---python > simeck_masked.py > ...

```python
"""
Masked implementation of the Simeck32/64 lightweight block cipher.
Uses first-order Boolean masking for side-channel attack resistance.
"""

import numpy as np
from typing import Tuple, List

class Simeck32_64_Masked:
    """
    Masked implementation of the Simeck32/64 block cipher.
    Uses first-order Boolean masking for side-channel resistance.
    """

    # Round constants for key schedule
    ROUND_CONSTANTS = [
        0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
        0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000, 0x8000,
        0x001F, 0x003E, 0x007C, 0x00F8, 0x01F0, 0x03E0, 0x07C0, 0x0F80,
        0x1F00, 0x0E01, 0x1C02, 0x3804, 0x7008, 0xE010, 0xC021, 0x8023
    ]

    def __init__(self, seed: int = 42):
        """Initialize the masked cipher with a random seed for masking."""
        self.rng = np.random.RandomState(seed)

    @staticmethod
    def _rotate_left(x: int, n: int, bits: int = 16) -> int:
        """Rotate left n bits of a bits-bit number."""
        return ((x << n) | (x >> (bits - n))) & ((1 << bits) - 1)

    @staticmethod
    def _rotate_right(x: int, n: int, bits: int = 16) -> int:
        """Rotate right n bits of a bits-bit number."""
        return ((x >> n) | (x << (bits - n))) & ((1 << bits) - 1)

    def _generate_mask(self, bits: int = 16) -> int:
        """Generate a random mask of specified bit length."""
        return self.rng.randint(0, 1 << bits, dtype=np.uint32)

    def _mask(self, x: int, m: int) -> int:
        """Apply mask to value."""
        return x ^ m

```

Figure 4a: Masking Implementation of Simeck 32/64 block Ciphers (First-Order).

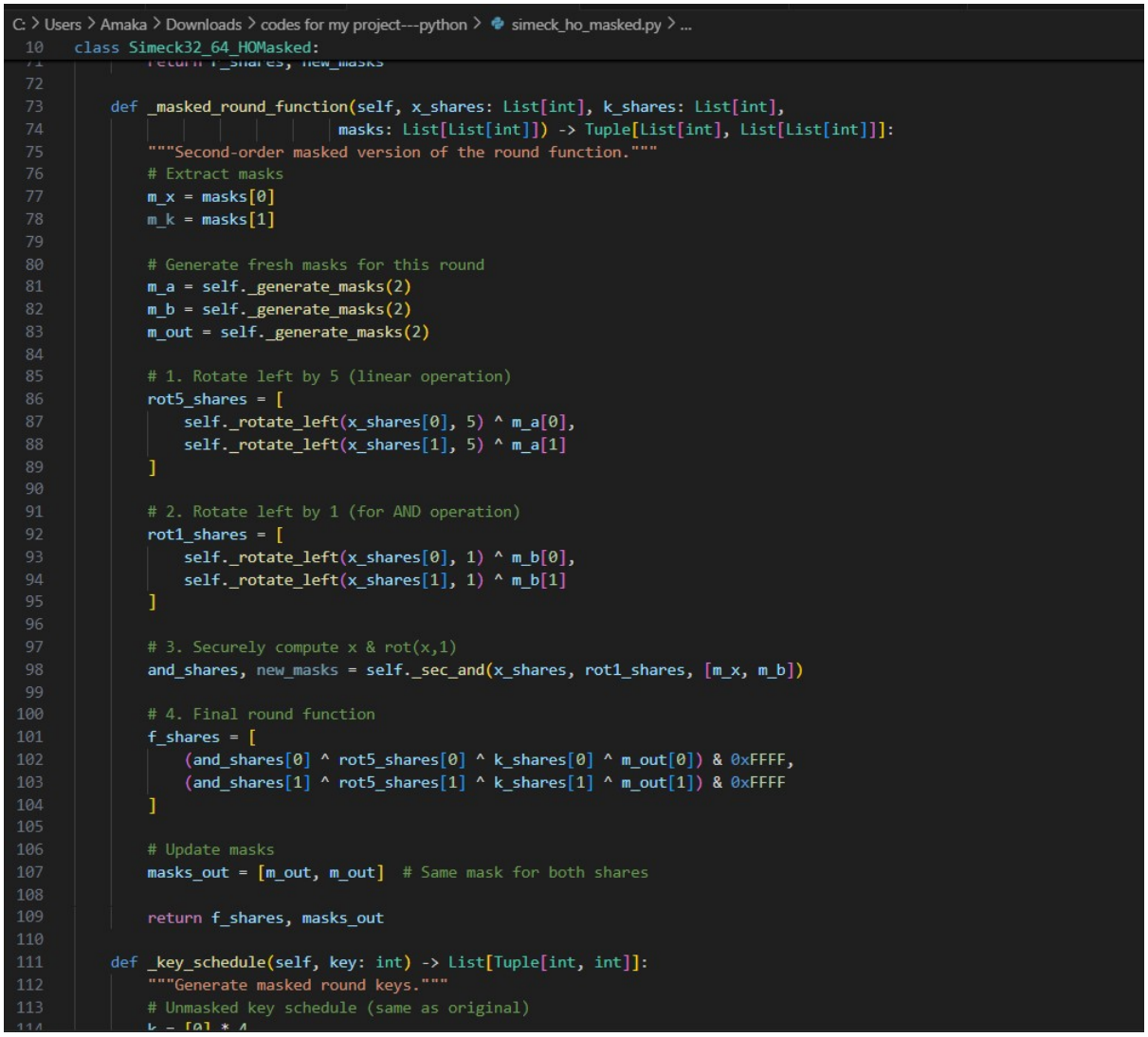


Figure 4b: Masking Implementation of Simeck 32/64 block Ciphers (Second-Order).

```
class Simeck32_64_Masked:
    def encrypt_with_leakage(cls, plaintext: int, key: int) -> Tuple[int, List[int]]:
                next_l_m = current_r_m

                # Update masks
                next_m_r = current_m_l
                next_m_l = new_m_l

                # Prepare for next round
                current_l_m, current_r_m = next_l_m, next_r_m
                current_m_l, current_m_r = next_m_l, next_m_r

            # Final state
            l = cipher._unmask(current_l_m, current_m_l)
            r = cipher._unmask(current_r_m, current_m_r)

            ciphertext = (l << 16) | r

            return ciphertext, leakage

# Test the implementation
if __name__ == "__main__":
    # Test vector from the original Simeck paper
    plaintext = 0x65656877
    key = 0x1918111009080100
    expected_ciphertext = 0x770d2c36

    # Test masked implementation
    cipher = Simeck32_64_Masked()
    ciphertext = cipher.encrypt(plaintext, key)

    print(f"Plaintext:  {plaintext:08x}")
    print(f"Key:        {key:016x}")
    print(f"Expected:   {expected_ciphertext:08x}")
    print(f"Got:        {ciphertext:08x}")
    print(f"Correct:    {ciphertext == expected_ciphertext}")

    # Test with leakage simulation
    print("\nTesting with leakage simulation...")
    ciphertext, leakage = Simeck32_64_Masked.encrypt_with_leakage(plaintext, key)
    print(f"Ciphertext: {ciphertext:08x}")
    print(f"Leakage trace length: {len(leakage)}")
    print(f"First 10 leakage points: {leakage[:10]}")
```

Figure 4c: Masking Implementation of Simeck 32/64 block Ciphers Tested.

6). Attack Implementations (side_channel.py, test_side_channel.py)

The code displays the degree of the attack after the defense measure, such as masking, has been implemented. Meanwhile, “test_side_channel.py” code tests for side-channel analysis and also the success of the masking defense.

7) Visualization & Reporting (src/visualization.py, notebooks)

This stage involves the publication of high-quality graph plots that will be incorporated into figures and documents.

C: > Users > Amaka > Downloads > codes for my project---python > visualization.py > ...

```python
"""
Visualization utilities for side-channel analysis results.
"""
import matplotlib.pyplot as plt
import numpy as np
from typing import List, Dict, Any

def plot_leakage_traces(traces: List[List[int]], title: str = "Side-Channel Leakage Traces") -> None:
    """
    Plot multiple side-channel leakage traces.

    Args:
        traces: List of leakage traces, where each trace is a list of Hamming weights
        title: Plot title
    """
    plt.figure(figsize=(12, 6))

    # Plot a subset of traces to avoid overcrowding
    max_traces = min(50, len(traces))
    for i in range(max_traces):
        plt.plot(traces[i], alpha=0.5, linewidth=0.5)

    # Plot mean trace in bold
    mean_trace = np.mean(traces, axis=0)
    plt.plot(mean_trace, 'k-', linewidth=2, label='Mean Trace')

    plt.title(title)
    plt.xlabel('Time Sample')
    plt.ylabel('Hamming Weight')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    return plt.gcf()

def plot_correlation_analysis(analyzer_results: Dict[str, Any], title: str = "Correlation Analysis") -> None:
    """
    Plot correlation analysis results from the side-channel analyzer.

    Args:
        analyzer_results: Dictionary containing analysis results from SideChannelAnalyzer
        title: Plot title
    """
```

Figure 5a: Visualization and Reporting.

C: > Users > Amaka > Downloads > codes for my project---python > visualization.py > ...

```python
def plot_correlation_analysis(analyzer_results: Dict[str, Any], title: str = "Correlation Analysis") -> None:

    coeffs = analyzer_results['coefficients']

    plt.figure(figsize=(12, 4))

    # Plot absolute values of coefficients
    abs_coeffs = np.abs(coeffs)
    plt.bar(range(len(abs_coeffs)), abs_coeffs, width=1.0)

    # Highlight top features if available
    if 'top_features' in analyzer_results and analyzer_results['top_features']:
        top_features = analyzer_results['top_features']
        plt.scatter(
            top_features,
            [abs_coeffs[i] for i in top_features],
            color='red',
            zorder=5,
            label=f'Top {len(top_features)} Features'
        )

    plt.title(f"{title}\nCorrelation Coefficients (R² = {analyzer_results['score']:.4f})")
    plt.xlabel('Feature Index')
    plt.ylabel('Absolute Correlation')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    return plt.gcf()

def compare_leakage(original_traces: List[List[int]],
                    masked_traces: List[List[int]],
                    title: str = "Leakage Comparison") -> None:
    """
    Compare original and masked leakage traces.

    Args:
        original_traces: Leakage traces without masking
        masked_traces: Leakage traces with masking
        title: Plot title
    """
    # Calculate mean traces
    mean_original = np.mean(original_traces, axis=0)
    mean_masked = np.mean(masked_traces, axis=0)

```

Figure 5b: Visualization and Reporting.

C: > Users > Amaka > Downloads > codes for my project---python > visualization.py > ...

```
 73    def compare_leakage(original_traces: List[List[int]],
106        ax2.set_xlabel('Time Sample')
107        ax2.set_ylabel('Variance')
108        ax2.legend()
109        ax2.grid(True, alpha=0.3)
110
111        plt.tight_layout()
112        return fig
113
114    def plot_attack_success_rates(success_rates: List[float],
115                                  labels: List[str],
116                                  title: str = "Attack Success Rates") -> None:
117        """
118        Plot the success rates of different attack scenarios.
119
120        Args:
121            success_rates: List of success rates (0-1)
122            labels: Labels for each scenario
123            title: Plot title
124        """
125        if len(success_rates) != len(labels):
126            raise ValueError("success_rates and labels must have the same length")
127
128        plt.figure(figsize=(10, 6))
129
130        # Create bar plot
131        x = range(len(success_rates))
132        bars = plt.bar(x, success_rates, width=0.6)
133
134        # Add value labels on top of bars
135        for bar in bars:
136            height = bar.get_height()
137            plt.text(bar.get_x() + bar.get_width()/2., height,
138                     f'{height:.2%}',
139                     ha='center', va='bottom')
140
141        plt.xticks(x, labels, rotation=45, ha='right')
142        plt.ylim(0, 1.1)
143        plt.title(title)
144        plt.ylabel('Success Rate')
145        plt.grid(True, alpha=0.3, axis='y')
146        plt.tight_layout()
147        return plt.gcf()
148
```

Figure 5c: Visualization and Reporting.

Optional Packages

PyPDF2 – This open-source pure-Python PDF library is used only for documentation tooling and is not required for cryptographic analysis.

Installation (Virtual-Environment Setup Commands)

First, open the deepnote- "Deepnote for my Implementation"

The following stages are needed to replicate the research environment

Install Python 3.12 (example for Windows 11 pro)
brew install python@3.12

Stage 1 — To Clone Repository

- git clone <repository-url>
- cd iot-wearable-security

Stage 2 — To Create Virtual Environment (Recommended)

- python3.12 -m venv venv
- source venv/bin/activate *# Windows: venv\Scripts\activate.bat*

Stage 3 — To Install Dependencies (*Install project libraries)*

- pip install -r requirements.txt *# numpy, scipy, sklearn, matplotlib, jupyter*

```
(venv) root@deepnote:~/work # cat requirements.txt
numpy>=1.21.0
matplotlib>=3.4.0
scikit-learn>=0.24.0
jupyter>=1.0.0
pytest>=6.2.0
h5py>=3.1.0
tqdm>=4.60.0
(venv) root@deepnote:~/work #
```

Figure 6a: python dependencies installation.

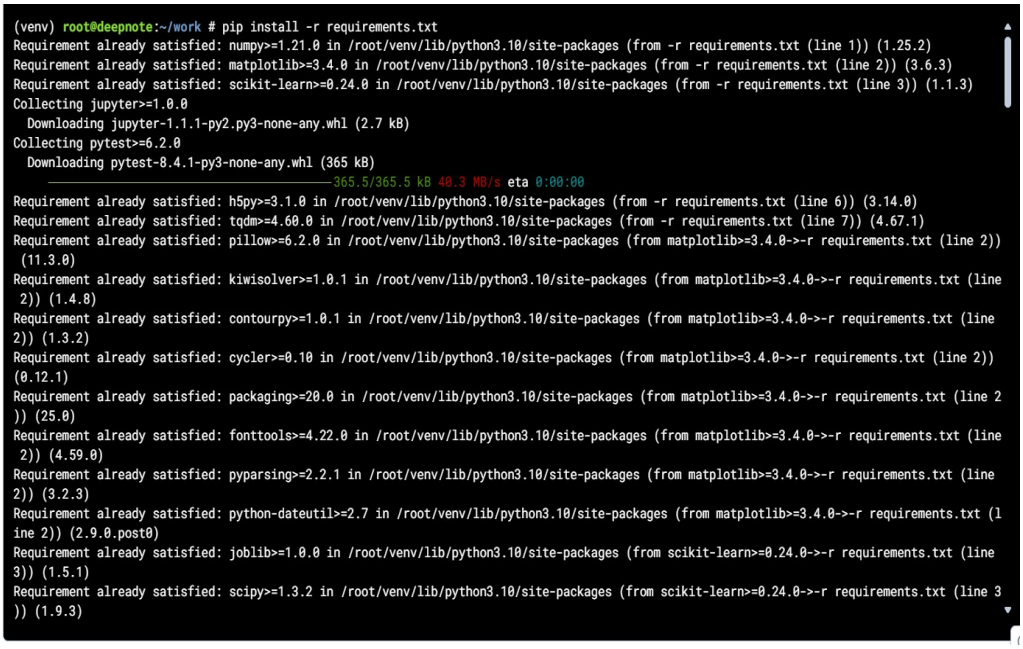


Figure 6b: python dependencies installation.

***Note**: If you plan to regenerate this manual or parse the template PDF, install:*

- pip install PyPDF2

Stage 4 — Verify installation

```
python - << 'PY'
    import importlib, pkg_resources, sys
missing = []
for req in pkg_resources.parse_requirements(open('requirements.txt')):
  try:
    importlib.import_module(req.key)
  except ImportError:
    missing.append(req.key)
if missing:
  sys.exit(f'Missing packages: {missing}')
print('Environment OK!')
PY
```

Stage 5 — Generate Synthetic Dataset & Run Analysis

- python run_analysis.py

Outputs are written to: * data/synthetic/ — generated HDF5 traces and metadata.
* results/ — plots and attack metrics.

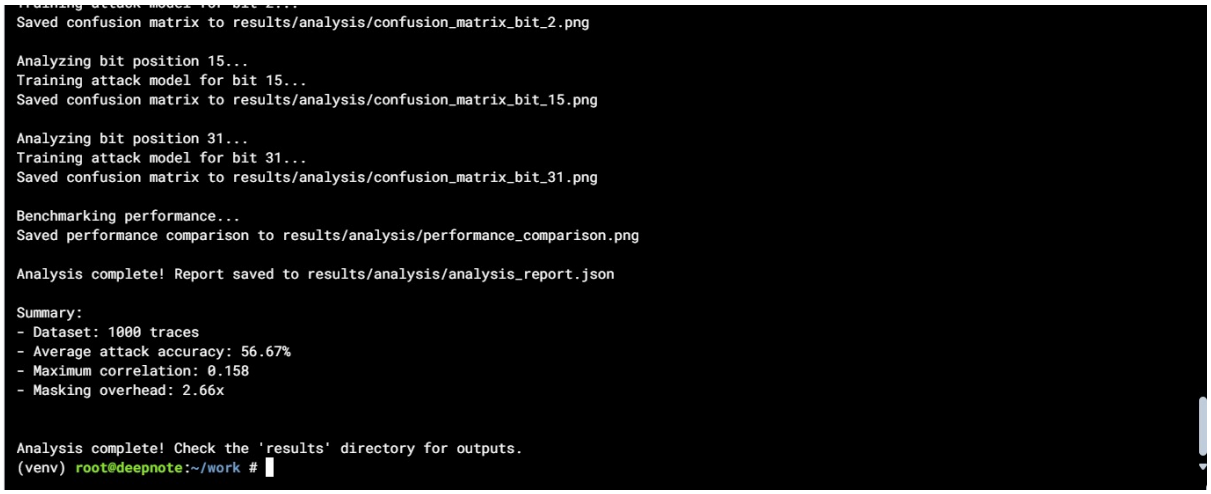


Figure 7: Traces generation andattack success.

Stage 6 —Launch and Explore Notebook (Optional)

- Jupyter notebook notebooks/demo.ipynb

Run the cells to visualize leakage, perform CPA attacks, and compare masked vs unmasked implementations

Reproducing Experimental Results

1. Ensure **Stage 1–5** complete without errors.

2. Open the resulting figures under results/ and compare with those in docs/figures.

3. Key metrics to reproduce:

   SNR curves — should show ≥ 15 dB reduction when masking is enabled.

   Attack success rate — CPA attack should fail (< 5 % success) against first-order masking after 1000 traces.

   Minor numerical differences are expected due to randomness; overall trends must match.

Experiment Navigation

Artifact	Location
Synthetic traces (HDF5)	data/synthetic/traces.h5
Analysis results	results/
Publication-ready figures	docs/figures/
Interactive notebook	notebooks/demo.ipynb

Code Overview (What Each File Does)

File	Purpose
src/simeck.py	Pure-Python Simeck32/64 cipher implementation
src/side_channel.py	Leakage model, masking, attack algorithms
src/visualization.py	Plotting helpers used by notebook & scripts
run_analysis.py	End-to-end driver: dataset → analysis → figures
notebooks/demo.ipynb	Interactive demo replicating paper figures

References

Collette, A. (2015). *HDF5 for Python — h5py 3.14.0 documentation*. [online] H5py.org. Available at: https://docs.h5py.org/.

Deepnote (2025). *Deepnote - Data science notebook for teams*. [online] Deepnote. Available at: https://deepnote.com/.

GNS3 (2025). *Downloading the GNS3 VM | GNS3 Documentation*. [online] mother.github.io. Available at: https://docs.gns3.com/docs/getting-started/installation/download-gns3-vm/.

Matplotlib (2024). *Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation*. [online] Matplotlib.org. Available at: https://matplotlib.org/.

NumPy (2022). *Overview — NumPy v1.19 Manual*. [online] numpy.org. Available at: https://numpy.org/doc/stable/.

Scikit-learn (2024). *scikit-learn: Machine Learning in Python*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/.

Scipy.org. (2015). *Numpy and Scipy Documentation — Numpy and Scipy documentation*. [online] Available at: https://docs.scipy.org/doc/.