

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

 X23324431
Student ID:
 X23324431 2025
Programme: **Year:**
 Msc Cybersecurity -Group B
Module:
 Khadija Hafeez
Lecturer:
Submission Due Date: 11th August 2025

 Practicum 2
Project Title:

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:

.....
 11th August 2025

Date:

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.

```

C: > Users > Amaka > Downloads > codes for my project---python > simeck.py > Simeck32_64 > _rotl
1 import numpy as np
2 from typing import Tuple
3
4 class Simeck32_64:
5     """
6     Implementation of the Simeck32/64 lightweight block cipher.
7     Encrypts 32-bit blocks using a 64-bit key over 32 rounds.
8     """
9
10    # Number of rounds
11    ROUNDS = 32
12
13    # Round constants for Simeck32/64
14    ROUND_CONSTANTS = [
15        0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
16        0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000, 0x8000,
17        0x001F, 0x003E, 0x007C, 0x00F8, 0x01F0, 0x03E0, 0x07C0, 0x0F80,
18        0x1F00, 0x0E01, 0x1C02, 0x3804, 0x7008, 0xE010, 0xC021, 0x8023
19    ]
20
21    @staticmethod
22    def _rotl(x: int, n: int, bits: int = 16) -> int:
23        """Rotate left n bits of x, considering only 'bits' significant bits."""
24        n = n % bits # Ensure n is within bit width
25        return ((x << n) | (x >> (bits - n))) & ((1 << bits) - 1)
26
27    @classmethod
28    def _f(cls, x: int) -> int:
29        """The non-linear function used in Simeck."""
30        # f(x) = (x <<< 5) & (x <<< 0) ^ (x <<< 1)
31        # For Simeck32/64: a=5, b=0, c=1
32        rotated_5 = cls._rotl(x, 5, 16)
33        rotated_1 = cls._rotl(x, 1, 16)
34        return (rotated_5 & x) ^ rotated_1
35
36    @classmethod
37    def _key_schedule(cls, key: int) -> list:
38        """Generate round keys from the master key."""
39        # Split 64-bit key into four 16-bit words (k3, k2, k1, k0)
40        k = [(key >> (i * 16)) & 0xFFFF for i in range(4)]
41        round_keys = []
42
43        for i in range(cls.ROUNDS):
44            round_keys.append(k[0])
45
46            # Generate next round key
47            tmp = k[0]
48            k[0] = (k[1] ^ cls._f(k[0]) ^ cls.ROUND_CONSTANTS[i] ^ k[3]) & 0xFFFF
49            # Rotate key state
50            k[1], k[2], k[3] = k[2], k[3], tmp
51
52        return round_keys
53
54    @classmethod

```

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

```

84     def hamming_weight(cls, x: int, bits: int = 32) -> int:
85         """
86         Calculate the Hamming weight (number of set bits) of an integer.
87
88         Args:
89             x: Integer to calculate Hamming weight for
90             bits: Number of bits to consider
91
92         Returns:
93             Hamming weight (number of 1s in binary representation)
94         """
95         return bin(x & ((1 << bits) - 1)).count('1')
96
97     @classmethod
98     def encrypt_with_leakage(cls, plaintext: int, key: int) -> Tuple[int, list]:
99         """
100         Encrypt plaintext and simulate side-channel leakage (Hamming weights).
101
102         Args:
103             plaintext: 32-bit integer to encrypt
104             key: 64-bit encryption key
105
106         Returns:
107             Tuple of (ciphertext, leakage_traces) where leakage_traces is a list of
108             Hamming weights of intermediate values during encryption
109         """
110         x = (plaintext >> 16) & 0xFFFF
111         y = plaintext & 0xFFFF
112         round_keys = cls._key_schedule(key)
113
114         leakage = []
115
116         # Initial state leakage
117         leakage.append(cls.hamming_weight(x, 16) + cls.hamming_weight(y, 16))
118
119         for i in range(cls.ROUNDS):
120             # Leakage before round function
121             leakage.append(cls.hamming_weight(x, 16))
122
123             # Apply round function and key addition
124             tmp = x
125             f_out = cls._f(x)
126             leakage.append(cls.hamming_weight(f_out, 16))
127
128             x = y ^ f_out ^ round_keys[i]
129             y = tmp
130
131             # Leakage after round function
132             leakage.append(cls.hamming_weight(x, 16) + cls.hamming_weight(y, 16))
133
134         ciphertext = (x << 16) | y
135         return ciphertext, leakage

```

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

3. Side-Channel Leakage Simulation (src/side_channel.py):

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 > ...
1  """
2  Side-channel analysis and countermeasures for Simeck32/64 implementation.
3  """
4  import numpy as np
5  from typing import List, Tuple, Dict, Optional
6  from sklearn.linear_model import LinearRegression
7  from .simeck import Simeck32_64
8
9  class SideChannelAnalyzer:
10     """
11     Performs side-channel analysis on the Simeck32/64 implementation.
12     """
13
14     def __init__(self, num_traces: int = 1000):
15         """
16         Initialize the side-channel analyzer.
17
18         Args:
19             num_traces: Number of traces to collect for analysis
20         """
21         self.num_traces = num_traces
22         self.traces = []
23         self.plaintexts = []
24         self.ciphertexts = []
25
26     def collect_traces(self, key: int) -> None:
27         """
28         Collect side-channel traces by encrypting random plaintexts.
29
30         Args:
31             key: 64-bit encryption key to use for collection
32         """
33         self.traces = []
34         self.plaintexts = []
35         self.ciphertexts = []
36
37         for _ in range(self.num_traces):
38             # Generate random plaintext
39             pt = np.random.randint(0, 2**32, dtype=np.uint32)
40
41             # Encrypt and collect leakage
42             ct, leakage = Simeck32_64.encrypt_with_leakage(pt, key)
43
44             self.plaintexts.append(pt)
```

Figure 2a: Side Channel Leakage Simulation.

```

46         self.traces.append(leakage)
47
48     def perform_DPA_attack(self, target_bit: int, max_features: int = 10) -> Dict:
49         """
50         Perform a Differential Power Analysis attack on the collected traces.
51
52         Args:
53             target_bit: Which bit of the key to target (0-63)
54             max_features: Maximum number of features to use in the model
55
56         Returns:
57             Dictionary containing attack results and metrics
58         """
59         if not self.traces:
60             raise ValueError("No traces collected. Call collect_traces() first.")
61
62         # Prepare features (Hamming weights at different points in time)
63         X = np.array(self.traces)
64
65         # Target is the target bit of the plaintext
66         # We'll predict each bit of the plaintext to find correlations with leakage
67         if target_bit >= 32:
68             raise ValueError("target_bit must be between 0 and 31")
69
70         # Get the target bit from each plaintext (0 or 1)
71         y = np.array([(pt >> target_bit) & 1 for pt in self.plaintexts])
72
73         # Use linear regression to find correlations
74         model = LinearRegression()
75         model.fit(X, y)
76
77         # Get the most important features
78         if hasattr(model, 'coef_'):
79             top_features = np.argsort(np.abs(model.coef_))[-max_features:]
80         else:
81             top_features = []
82
83         return {
84             'model': model,
85             'score': model.score(X, y),
86             'top_features': top_features.tolist() if len(top_features) > 0 else [],
87             'coefficients': model.coef_.tolist() if hasattr(model, 'coef_') else []
88         }

```

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 > ...
1  #!/usr/bin/env python3
2  """
3  Simple script to generate dataset and run analysis.
4  """
5
6  import sys
7  import os
8  import json
9  import h5py
10 import numpy as np
11 from pathlib import Path
12 from tqdm import tqdm
13
14 # Add the parent directory to the path to import simeck
15 sys.path.append(str(Path(__file__).parent.parent))
16
17 from src.simeck import Simeck32_64
18
19 def generate_dataset(num_traces=1000, trace_length=100, noise_level=0.1):
20     """Generate synthetic power traces for Simeck32/64 encryption."""
21     print(f"Generating {num_traces} traces...")
22
23     traces = np.zeros((num_traces, trace_length), dtype=np.float32)
24     metadata = []
25
26     for i in tqdm(range(num_traces), desc="Generating traces"):
27         # Generate random plaintext and key
28         pt = np.random.randint(0, 2**32, dtype=np.uint32)
29         key = np.random.randint(0, 2**64, dtype=np.uint64)
30
31         # Encrypt and get leakage
32         _, leakage = Simeck32_64.encrypt_with_leakage(pt, key)
33
34         # Pad or truncate leakage to desired length
35         if len(leakage) > trace_length:
36             leakage = leakage[:trace_length]
37         elif len(leakage) < trace_length:
38             leakage = np.pad(leakage, (0, trace_length - len(leakage)), 'edge')
39
40         # Add noise and store
41         traces[i] = leakage + np.random.normal(0, noise_level, trace_length)
42
43         # Store metadata
44         metadata.append({

```

Figure 3a: Dataset Generation.

```

C: > Users > Amaka > Downloads > codes for my project---python > run_analysis_simple.py > ...
19 def generate_dataset(num_traces=1000, trace_length=100, noise_level=0.1):
45     'plaintext': int(pt),
46     'ciphertext': int(Simeck32_64.encrypt(pt, key)),
47     'key': int(key),
48     'leakage_points': [int(x) for x in leakage]
49     })
50
51     return traces, metadata
52
53 def save_dataset(traces, metadata, output_dir):
54     """Save traces and metadata to disk."""
55     output_dir = Path(output_dir)
56     output_dir.mkdir(parents=True, exist_ok=True)
57
58     # Save traces to HDF5
59     traces_path = output_dir / 'traces.h5'
60     with h5py.File(traces_path, 'w') as f:
61         f.create_dataset('traces', data=traces)
62     print(f"Saved traces to {traces_path}")
63
64     # Save metadata to JSON
65     metadata_path = output_dir / 'metadata.json'
66     with open(metadata_path, 'w') as f:
67         json.dump(metadata, f)
68     print(f"Saved metadata to {metadata_path}")
69
70 def main():
71     # Set up paths
72     data_dir = Path("data/synthetic")
73     data_dir.mkdir(parents=True, exist_ok=True)
74
75     # Generate and save dataset
76     traces, metadata = generate_dataset()
77     save_dataset(traces, metadata, data_dir)
78
79     print("\nDataset generation complete!")
80     print(f"- Number of traces: {len(traces)}")
81     print(f"- Trace length: {traces.shape[1]}")
82     print(f"- Output directory: {data_dir.absolute()}")
83
84 if __name__ == "__main__":
85     main()
86

```

Figure 3b: Dataset Generation.

5). Implementation of Countermeasures (Masking Defense)

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 > ...
1  """
2  Masked implementation of the Simeck32/64 lightweight block cipher.
3  Uses first-order Boolean masking for side-channel attack resistance.
4  """
5
6  import numpy as np
7  from typing import Tuple, List
8
9  class Simeck32_64_Masked:
10     """
11     Masked implementation of the Simeck32/64 block cipher.
12     Uses first-order Boolean masking for side-channel resistance.
13     """
14
15     # Round constants for key schedule
16     ROUND_CONSTANTS = [
17         0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
18         0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000, 0x8000,
19         0x001F, 0x003E, 0x007C, 0x00F8, 0x01F0, 0x03E0, 0x07C0, 0x0F80,
20         0x1F00, 0x0E01, 0x1C02, 0x3804, 0x7008, 0xE010, 0xC021, 0x8023
21     ]
22
23     def __init__(self, seed: int = 42):
24         """Initialize the masked cipher with a random seed for masking."""
25         self.rng = np.random.RandomState(seed)
26
27     @staticmethod
28     def _rotate_left(x: int, n: int, bits: int = 16) -> int:
29         """Rotate left n bits of a bits-bit number."""
30         return ((x << n) | (x >> (bits - n))) & ((1 << bits) - 1)
31
32     @staticmethod
33     def _rotate_right(x: int, n: int, bits: int = 16) -> int:
34         """Rotate right n bits of a bits-bit number."""
35         return ((x >> n) | (x << (bits - n))) & ((1 << bits) - 1)
36
37     def _generate_mask(self, bits: int = 16) -> int:
38         """Generate a random mask of specified bit length."""
39         return self.rng.randint(0, 1 << bits, dtype=np.uint32)
40
41     def _mask(self, x: int, m: int) -> int:
42         """Apply mask to value."""
43         return x ^ m
44

```

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

```

C: > Users > Amaka > Downloads > codes for my project---python > simeck_ho_masked.py > ...
10 class Simeck32_64_HOMasked:
11     def __init__(self, x_shares, new_masks):
12         self.x_shares = x_shares
13         self.masks = new_masks
14
15     def _masked_round_function(self, x_shares: List[int], k_shares: List[int],
16                               masks: List[List[int]]) -> Tuple[List[int], List[List[int]]]:
17         """Second-order masked version of the round function."""
18         # Extract masks
19         m_x = masks[0]
20         m_k = masks[1]
21
22         # Generate fresh masks for this round
23         m_a = self._generate_masks(2)
24         m_b = self._generate_masks(2)
25         m_out = self._generate_masks(2)
26
27         # 1. Rotate left by 5 (linear operation)
28         rot5_shares = [
29             self._rotate_left(x_shares[0], 5) ^ m_a[0],
30             self._rotate_left(x_shares[1], 5) ^ m_a[1]
31         ]
32
33         # 2. Rotate left by 1 (for AND operation)
34         rot1_shares = [
35             self._rotate_left(x_shares[0], 1) ^ m_b[0],
36             self._rotate_left(x_shares[1], 1) ^ m_b[1]
37         ]
38
39         # 3. Securely compute x & rot(x,1)
40         and_shares, new_masks = self._sec_and(x_shares, rot1_shares, [m_x, m_b])
41
42         # 4. Final round function
43         f_shares = [
44             (and_shares[0] ^ rot5_shares[0] ^ k_shares[0] ^ m_out[0]) & 0xFFFF,
45             (and_shares[1] ^ rot5_shares[1] ^ k_shares[1] ^ m_out[1]) & 0xFFFF
46         ]
47
48         # Update masks
49         masks_out = [m_out, m_out] # Same mask for both shares
50
51         return f_shares, masks_out
52
53     def _key_schedule(self, key: int) -> List[Tuple[int, int]]:
54         """Generate masked round keys."""
55         # Unmasked key schedule (same as original)
56         k = [0] * 4

```

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

```

9  class Simeck32_64_Masked:
171 def encrypt_with_leakage(cls, plaintext: int, key: int) -> Tuple[int, List[int]]:
220     next_l_m = current_r_m
221
222     # Update masks
223     next_m_r = current_m_l
224     next_m_l = new_m_l
225
226     # Prepare for next round
227     current_l_m, current_r_m = next_l_m, next_r_m
228     current_m_l, current_m_r = next_m_l, next_m_r
229
230     # Final state
231     l = cipher._unmask(current_l_m, current_m_l)
232     r = cipher._unmask(current_r_m, current_m_r)
233
234     ciphertext = (l << 16) | r
235
236     return ciphertext, leakage
237
238 # Test the implementation
239 if __name__ == "__main__":
240     # Test vector from the original Simeck paper
241     plaintext = 0x65656877
242     key = 0x1918111009080100
243     expected_ciphertext = 0x770d2c36
244
245     # Test masked implementation
246     cipher = Simeck32_64_Masked()
247     ciphertext = cipher.encrypt(plaintext, key)
248
249     print(f"Plaintext:  {plaintext:08x}")
250     print(f"Key:        {key:016x}")
251     print(f"Expected:    {expected_ciphertext:08x}")
252     print(f"Got:         {ciphertext:08x}")
253     print(f"Correct:     {ciphertext == expected_ciphertext}")
254
255     # Test with leakage simulation
256     print("\nTesting with leakage simulation...")
257     ciphertext, leakage = Simeck32_64_Masked.encrypt_with_leakage(plaintext, key)
258     print(f"Ciphertext: {ciphertext:08x}")
259     print(f"Leakage trace length: {len(leakage)}")
260     print(f"First 10 leakage points: {leakage[:10]}")
261

```

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 > ...

1  """
2  Visualization utilities for side-channel analysis results.
3  """
4  import matplotlib.pyplot as plt
5  import numpy as np
6  from typing import List, Dict, Any
7
8  def plot_leakage_traces(traces: List[List[int]], title: str = "Side-Channel Leakage Traces") -> None:
9      """
10      Plot multiple side-channel leakage traces.
11
12      Args:
13          traces: List of leakage traces, where each trace is a list of Hamming weights
14          title: Plot title
15      """
16      plt.figure(figsize=(12, 6))
17
18      # Plot a subset of traces to avoid overcrowding
19      max_traces = min(50, len(traces))
20      for i in range(max_traces):
21          plt.plot(traces[i], alpha=0.5, linewidth=0.5)
22
23      # Plot mean trace in bold
24      mean_trace = np.mean(traces, axis=0)
25      plt.plot(mean_trace, 'k-', linewidth=2, label='Mean Trace')
26
27      plt.title(title)
28      plt.xlabel('Time Sample')
29      plt.ylabel('Hamming Weight')
30      plt.legend()
31      plt.grid(True, alpha=0.3)
32      plt.tight_layout()
33      return plt.gcf()
34
35  def plot_correlation_analysis(analyzer_results: Dict[str, Any], title: str = "Correlation Analysis") -> None:
36      """
37      Plot correlation analysis results from the side-channel analyzer.
38
39      Args:
40          analyzer_results: Dictionary containing analysis results from SideChannelAnalyzer
41          title: Plot title
42      """
43

```

Figure 5a: Visualization and Reporting.

```

C:\Users\Amaka > Downloads > codes for my project---python > visualization.py > ...
35 def plot_correlation_analysis(analyzer_results: Dict[str, Any], title: str = "Correlation Analysis") -> None:
45
46     coeffs = analyzer_results['coefficients']
47
48     plt.figure(figsize=(12, 4))
49
50     # Plot absolute values of coefficients
51     abs_coeffs = np.abs(coeffs)
52     plt.bar(range(len(abs_coeffs)), abs_coeffs, width=1.0)
53
54     # Highlight top features if available
55     if 'top_features' in analyzer_results and analyzer_results['top_features']:
56         top_features = analyzer_results['top_features']
57         plt.scatter(
58             top_features,
59             [abs_coeffs[i] for i in top_features],
60             color='red',
61             zorder=5,
62             label=f'Top {len(top_features)} Features'
63         )
64
65     plt.title(f"{title}\nCorrelation Coefficients ( $R^2$  = {analyzer_results['score']:.4f})")
66     plt.xlabel('Feature Index')
67     plt.ylabel('Absolute Correlation')
68     plt.legend()
69     plt.grid(True, alpha=0.3)
70     plt.tight_layout()
71     return plt.gcf()
72
73 def compare_leakage(original_traces: List[List[int]],
74                    masked_traces: List[List[int]],
75                    title: str = "Leakage Comparison") -> None:
76     """
77     Compare original and masked leakage traces.
78
79     Args:
80         original_traces: Leakage traces without masking
81         masked_traces: Leakage traces with masking
82         title: Plot title
83     """
84     # Calculate mean traces
85     mean_original = np.mean(original_traces, axis=0)
86     mean_masked = np.mean(masked_traces, axis=0)
87

```

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

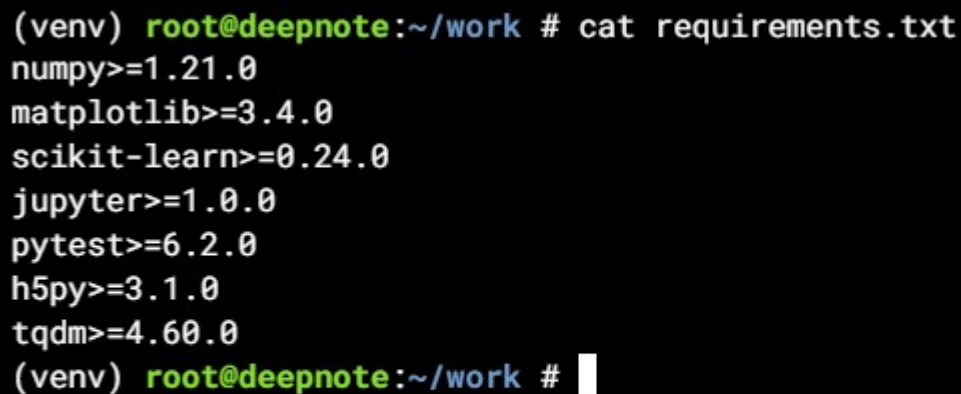
- `git clone <repository-url>`
- `cd iot-wearable-security`

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.

```
(venv) root@deepnote:~/work # pip install -r requirements.txt
Requirement already satisfied: numpy>=1.21.0 in /root/venv/lib/python3.10/site-packages (from -r requirements.txt (line 1)) (1.25.2)
Requirement already satisfied: matplotlib>=3.4.0 in /root/venv/lib/python3.10/site-packages (from -r requirements.txt (line 2)) (3.6.3)
Requirement already satisfied: scikit-learn>=0.24.0 in /root/venv/lib/python3.10/site-packages (from -r requirements.txt (line 3)) (1.1.3)
Collecting jupyter>=1.0.0
  Downloading jupyter-1.1.1-py2.py3-none-any.whl (2.7 kB)
Collecting pytest>=6.2.0
  Downloading pytest-8.4.1-py3-none-any.whl (365 kB)
    365.5/365.5 kB 40.3 MB/s eta 0:00:00
Requirement already satisfied: h5py>=3.1.0 in /root/venv/lib/python3.10/site-packages (from -r requirements.txt (line 6)) (3.14.0)
Requirement already satisfied: tqdm>=4.60.0 in /root/venv/lib/python3.10/site-packages (from -r requirements.txt (line 7)) (4.67.1)
Requirement already satisfied: pillow>=6.2.0 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (11.3.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (1.4.8)
Requirement already satisfied: contourpy>=1.0.1 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (0.12.1)
Requirement already satisfied: packaging>=20.0 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (25.0)
Requirement already satisfied: fonttools>=4.22.0 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (4.59.0)
Requirement already satisfied: pyparsing>=2.2.1 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /root/venv/lib/python3.10/site-packages (from matplotlib>=3.4.0->-r requirements.txt (line 2)) (2.9.0.post0)
Requirement already satisfied: joblib>=1.0.0 in /root/venv/lib/python3.10/site-packages (from scikit-learn>=0.24.0->-r requirements.txt (line 3)) (1.5.1)
Requirement already satisfied: scipy>=1.3.2 in /root/venv/lib/python3.10/site-packages (from scikit-learn>=0.24.0->-r requirements.txt (line 3)) (1.9.3)
```

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.

```
[notice] A new release of pip is available: 23.0.1 -> 25.2
[notice] To update, run: pip install --upgrade pip
(venv) root@deepnote:~/work # python run_analysis.py
Generating dataset...
Running dataset generation...
Generated 1000 traces

Running analysis...
```

```
Training attack model for bit 2...
Saved confusion matrix to results/analysis/confusion_matrix_bit_2.png

Analyzing bit position 15...
Training attack model for bit 15...
Saved confusion matrix to results/analysis/confusion_matrix_bit_15.png

Analyzing bit position 31...
Training attack model for bit 31...
Saved confusion matrix to results/analysis/confusion_matrix_bit_31.png

Benchmarking performance...
Saved performance comparison to results/analysis/performance_comparison.png

Analysis complete! Report saved to results/analysis/analysis_report.json

Summary:
- Dataset: 1000 traces
- Average attack accuracy: 56.67%
- Maximum correlation: 0.158
- Masking overhead: 2.66x

Analysis complete! Check the 'results' directory for outputs.
(venv) root@deepnote:~/work #
```

Figure 7: Traces generation and attack 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/>.