Project Report: Related-Key Neural Distinguisher for Simon 32/64

(github repository - link)

1. Introduction

This project implements a Deep Learning-based cryptanalysis attack on the **Simon 32/64** block cipher. Based on the research paper *"A Multi-Differential Approach to Enhance Related-Key Neural Distinguishers"* by Yuan and Wang, I developed a neural distinguisher to identify non-random patterns in the ciphertext.

The core objectives of this project were to:

1. Implement the Simon 32/64 cipher in Python from scratch.
2. Construct a Residual Network (ResNet) enhanced with Squeeze-and-Excitation (SE) attention blocks.
3. Compare **Single-Differential** vs. **Multi-Differential** training strategies to experimentally determine the maximum number of rounds that can be distinguished from random noise using raw ciphertext.

2. Implementation

2.1 Cipher Implementation (simon.py)

I implemented the Simon 32/64 cipher using numpy to support efficient batch encryption. This allowed for the rapid generation of millions of training samples. The implementation supports a variable number of rounds, which is crucial for testing the security margin of the distinguisher.

Key Features:

- **Block Size:** 32 bits (two 16-bit words)
- **Key Size:** 64 bits (four 16-bit words)
- **Operations:** Bitwise XOR, AND, and Circular Shifts (ROL/ROR).

```python
import numpy as np
class Simon3264:
    def __init__(self):
        # Simon 32/64 Constants
        self.n = 16  # Word size
        self.m = 4   # Number of key words
        self.rounds = 32

        # Z0 sequence for Simon 32/64 key schedule
        self.z0 = 0b11111010001001010110000111001101111101000100101011000011100110

    def _rol(self, x, k):
        """Circular rotate left x by k bits."""
        return ((x << k) & 0xFFFF) | ((x >> (16 - k)) & 0xFFFF)

    def _ror(self, x, k):
        """Circular rotate right x by k bits."""
        return ((x >> k) & 0xFFFF) | ((x << (16 - k)) & 0xFFFF)
```

2.2 Data Generation Strategy

To train the neural network, I implemented two distinct data generators. Each generator produces positive samples (pairs satisfying a differential characteristic) and negative samples (random pairs).

1. **Single-Differential Generator:** Uses a single high-probability differential characteristic (Input Diff: 0x40) derived from the literature.
2. **Multi-Differential Generator:** Mixes two characteristics (Input Diff: 0x40 and 0x8010) to potentially provide richer features to the network.

```python
def generate_batch(self, batch_size=10000, rounds=12):
    """
    Generates a balanced batch of positive and negative samples.
    Returns:
        X: Ciphertext pairs (converted to binary/features)
        Y: Labels (1 for Related-Key, 0 for Random)
    """
    half_batch = batch_size // 2
    # --- 1. Generate Positive Samples (Label 1) ---
    # Random master keys K
    keys_pos = np.frombuffer(os.urandom(2 * 4 * half_batch), dtype=np.uint16).reshape(half_batch, 4)
    # Calculate related keys K' = K ^ diff_key
    # We need to broadcast the key difference
    diff_k_array = np.array(self.diff_key, dtype=np.uint16)
    keys_pos_prime = keys_pos ^ diff_k_array
    # Random plaintexts P (Left, Right)
    pt_pos = np.frombuffer(os.urandom(2 * 2 * half_batch), dtype=np.uint16).reshape(half_batch, 2)
    # Calculate related plaintexts P' = P ^ diff_in
    # diff_in is [Right, Left] because our encrypt takes [R, L] format in the previous code?
    # Let's verify standard: usually (Left, Right).
    # Paper says delta_p = (0x0, 0x40). Usually denotes (Left_Diff, Right_Diff).
    # My Simon3264.encrypt expects [Right, Left] order in the numpy array based on previous code logic.
    # Let's assume standard (L, R) for input params and convert.
    diff_p_array = np.array([self.diff_in_R, self.diff_in_L], dtype=np.uint16)
    pt_pos_prime = pt_pos ^ diff_p_array
    # Expand keys
    round_keys_pos = self.cipher.expand_key(keys_pos)
    round_keys_pos_prime = self.cipher.expand_key(keys_pos_prime)
    # Encrypt
    ct_pos = self.cipher.encrypt(pt_pos, round_keys_pos, rounds=rounds)
```

2.3 Neural Network Architecture (model.py)

The discriminator is a deep neural network built in **PyTorch**. It treats the cryptanalysis problem as a binary classification task, outputting a probability score where 0 represents Random and 1 represents Related-Key.

Architecture Highlights:

- **Input:** Concatenated ciphertext pair (C_L, C_R, C'_L, C'_R) converted to a 64-bit binary vector.
- **Backbone:** 1D Convolutional layers with Residual connections to prevent vanishing gradients during training.
- **Attention Mechanism: Squeeze-and-Excitation (SE)** blocks that re-calibrate channel weights, enabling the model to focus on specific bit-level correlations that are statistically significant.

```python
if __name__ == "__main__":
    # Check for GPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # Instantiate Model
    net = SimonDistinguisher(input_size=64).to(device)

    # Create Dummy Input (Batch=10, 64 bits)
    dummy_input = torch.randn(10, 64).to(device)

    # Forward Pass
    output = net(dummy_input)

    print(f"Input Shape: {dummy_input.shape}")
    print(f"Output Shape: {output.shape}")
    print(f"Sample Output: {output[0].item()}")
    print("SUCCESS: Model architecture is valid.")
```

```
Using device: cpu
Input Shape: torch.Size([10, 64])
Output Shape: torch.Size([10, 1])
Sample Output: 0.526356041431427
SUCCESS: Model architecture is valid.
```

3. Experiments & Results

I conducted a series of experiments to determine the "distinguishing limit" of the model using raw ciphertext.

Experiment 1: 6 Rounds (Baseline Test)

- **Method:** Single-Differential
- **Goal:** Verify the pipeline implementation. Accuracy should be significantly high.
- **Result:** The model achieved **83.46%** accuracy. This confirmed that the cipher implementation, data generation, and neural network were all functioning correctly.

```
Training on device: cpu
Starting training for Simon 32/64 (6 Rounds)...
Epoch [1/10] Loss: 0.2175 | Acc: 0.6569 (65.69%)
Epoch [2/10] Loss: 0.1535 | Acc: 0.7736 (77.36%)
Epoch [3/10] Loss: 0.1411 | Acc: 0.7873 (78.73%)
Epoch [4/10] Loss: 0.1369 | Acc: 0.7899 (78.99%)
Epoch [5/10] Loss: 0.1335 | Acc: 0.7959 (79.59%)
--- Validation Accuracy (Round 6): 0.7980 ---
Epoch [6/10] Loss: 0.1320 | Acc: 0.7992 (79.92%)
Epoch [7/10] Loss: 0.1303 | Acc: 0.8015 (80.15%)
Epoch [8/10] Loss: 0.1277 | Acc: 0.8072 (80.72%)
Epoch [9/10] Loss: 0.1272 | Acc: 0.8097 (80.97%)
Epoch [10/10] Loss: 0.1244 | Acc: 0.8152 (81.52%)
--- Validation Accuracy (Round 6): 0.8346 ---
Training Complete.
Model saved to simon_distinguisher.pth
```

Experiment 2: 7 Rounds (Single-Differential)

- **Method:** Single-Differential (Diff = 0x40)
- **Goal:** Test the limit of the primary differential.
- **Result:** The model achieved **60.50%** accuracy.
- **Analysis:** This proves the cipher is "broken" at 7 rounds, as the result is statistically distinguishable from random guessing (50%).

```
Training on device: cpu
Starting training for Simon 32/64 (7 Rounds)...
Epoch [1/10] Loss: 0.2514 | Acc: 0.5181 (51.81%)
Epoch [2/10] Loss: 0.2439 | Acc: 0.5734 (57.34%)
Epoch [3/10] Loss: 0.2409 | Acc: 0.5998 (59.98%)
Epoch [4/10] Loss: 0.2402 | Acc: 0.6032 (60.32%)
Epoch [5/10] Loss: 0.2398 | Acc: 0.6030 (60.30%)
--- Validation Accuracy (Round 7): 0.6071 ---
Epoch [6/10] Loss: 0.2393 | Acc: 0.6054 (60.54%)
Epoch [7/10] Loss: 0.2391 | Acc: 0.6053 (60.53%)
Epoch [8/10] Loss: 0.2389 | Acc: 0.6035 (60.35%)
Epoch [9/10] Loss: 0.2390 | Acc: 0.6005 (60.05%)
Epoch [10/10] Loss: 0.2379 | Acc: 0.6006 (60.06%)
--- Validation Accuracy (Round 7): 0.6049 ---
Training Complete.
Model saved to simon_distinguisher.pth
```

Experiment 3: 7 Rounds (Multi-Differential Comparison)

- **Method:** Multi-Differential (Diff A = 0x40, Diff B = 0x8010)
- **Goal:** Test if adding a second differential improves accuracy beyond 60.5%.
- **Result:** The model achieved **55.34%** accuracy.
- **Analysis:** The accuracy **decreased** compared to the single-differential approach. This indicates **Signal Dilution**: the second differential (0x8010) was likely too weak at 7 rounds, acting as noise that diluted the strong signal from the primary differential.

```python
# --- EXECUTE ---
if __name__ == "__main__":
    diff_trained_model = train_model()

    # Save the model
    torch.save(diff_trained_model.state_dict(), "simon_distinguisher.pth")
    print("Model saved to simon_distinguisher.pth")
```

```
Training on device: cpu
Starting training for Simon 32/64 (7 Rounds)...
Epoch [1/10] Loss: 0.2530 | Acc: 0.4996 (49.96%)
Epoch [2/10] Loss: 0.2509 | Acc: 0.5055 (50.55%)
Epoch [3/10] Loss: 0.2501 | Acc: 0.5122 (51.22%)
Epoch [4/10] Loss: 0.2493 | Acc: 0.5252 (52.52%)
Epoch [5/10] Loss: 0.2485 | Acc: 0.5387 (53.87%)
--- Validation Accuracy (Round 7): 0.5512 ---
Epoch [6/10] Loss: 0.2485 | Acc: 0.5410 (54.10%)
Epoch [7/10] Loss: 0.2479 | Acc: 0.5484 (54.84%)
Epoch [8/10] Loss: 0.2479 | Acc: 0.5489 (54.89%)
Epoch [9/10] Loss: 0.2478 | Acc: 0.5483 (54.83%)
Epoch [10/10] Loss: 0.2475 | Acc: 0.5515 (55.15%)
--- Validation Accuracy (Round 7): 0.5534 ---
Training Complete.
Model saved to simon_distinguisher.pth
```

Experiment 4: 8 Rounds (The Limit)

- **Method:** Single-Differential
- **Goal:** Find the "Avalanche Wall" where the cipher becomes secure against this specific attack.
- **Result:** The model converged to **50.95%** accuracy (Random Guessing).

```
Training on device: cpu
Starting training for Simon 32/64 (8 Rounds)...
Epoch [1/10] Loss: 0.2535 | Acc: 0.4987 (49.87%)
Epoch [2/10] Loss: 0.2514 | Acc: 0.5006 (50.06%)
Epoch [3/10] Loss: 0.2510 | Acc: 0.5006 (50.06%)
Epoch [4/10] Loss: 0.2509 | Acc: 0.4988 (49.88%)
Epoch [5/10] Loss: 0.2508 | Acc: 0.5011 (50.11%)
--- Validation Accuracy (Round 8): 0.5071 ---
Epoch [6/10] Loss: 0.2506 | Acc: 0.5009 (50.09%)
Epoch [7/10] Loss: 0.2504 | Acc: 0.5030 (50.30%)
Epoch [8/10] Loss: 0.2505 | Acc: 0.5011 (50.11%)
Epoch [9/10] Loss: 0.2503 | Acc: 0.5052 (50.52%)
Epoch [10/10] Loss: 0.2502 | Acc: 0.5046 (50.46%)
--- Validation Accuracy (Round 8): 0.5095 ---
Training Complete.
Model saved to simon_distinguisher.pth
```

4. Conclusion

This project successfully demonstrated the application of Deep Learning to cryptanalysis. The key findings are:

1. **Successful Distinguisher:** I built a neural distinguisher capable of breaking **7 rounds** of Simon 32/64 with **60.5% accuracy**, surpassing standard statistical randomness tests.
2. **Signal Dilution:** My experiments proved that simply adding more differentials (the Multi-Differential approach) does not automatically improve performance. If the added differentials are weaker than the primary one, they dilute the training signal, as seen in the drop from 60.5% to 55.3%.
3. **Security Margin:** Using raw ciphertext inputs, the "8-round wall" remains secure against this specific ResNet architecture without implementing partial decryption techniques.

5. References

1. **Yuan, X. & Wang, Q.** (2025). *A Multi-Differential Approach to Enhance Related-Key Neural Distinguishers*. The Computer Journal.
2. **Beaulieu, R., et al.** (2013). *The SIMON and SPECK lightweight block ciphers*.