CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Deep Learning-Assisted Differential Cryptanalysis on Round-reduced Block Ciphers: What are the advantages?

Master's thesis in Computer Science and Engineering

Lucas Flink

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Deep Learning-Assisted Differential Cryptanalysis on Round-reduced Block Ciphers: What are the advantages?

Lucas Flink

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Deep Learning-Assisted Differential Cryptanalysis on Round-reduced Block Ciphers: What are the advantages?

Lucas Flink

Supervisor: Rhouma Rhouma, Computer Science and Engineering
Advisor: Jonas Karlsson, Inter IKEA
Examiner: Elena Pagnin, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Deep Learning-Assisted Differential Cryptanalysis on Round-reduced Block Ciphers: What are the advantages?

Lucas Flink
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates and evaluates the application of deep learning techniques in differential cryptanalysis of lightweight block ciphers. The main two focuses are on comparison between a hybrid attack pipeline and a classical attack pipeline, and on transfer learning compared to training a model from scratch. The study considers both SPN ciphers and the SIMON32/64 cipher, which is part of the ISO standard for RFID systems. Neural distinguishers based on convolutional neural networks are trained to classify pairs of ciphertexts, as either random noise or ciphertexts produced by the cipher. For the SPN cipher it is integrated into a key recovery attack pipeline. For SIMON32/64 it is compared to the outcome of transfer learning of a pre-trained network. The results show that a hybrid approach is comparable to a classical approach in terms of key recovery attack. The transfer learning enables faster convergence, but does not reach the same accuracy in classification compared to training the model from scratch. These findings contribute to the understanding of the role of machine learning in cryptanalysis, and how it can be further studied to potentially be more useful in the future, and what the security impacts might be for real-world use in especially supply chains using RFID technology.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

I want to express my deepest thank you to both my supervisor Rhouma Rhouma and my examiner Elena Pagnin for supporting me throughout the thesis from day one to the very end. I also would like to give a big thank you to my advisor Jonas Karlsson at Inter IKEA for supporting me throughout the whole time of writing my thesis.

Lucas Flink, Gothenburg, 2026-04-21

Contents

List of Figures

List of Tables

1

Introduction

This thesis is divided into several steps to guide the reader through the process of the work. Chapter 1 gives an overview of the background to why this thesis is relevant, along with the goals this thesis aims to achieve. In Chapter 2, the reader is introduced to the necessary and relevant theory to get an understanding of the methods used. These methods are described in Chapter 3 where each experiment is described step by step and the reason why they were performed. Chapter 4 shows graphs, tables, and data from all the experiments. Finally, Chapter 5 will discuss and conclude the thesis from the results produced.

1.1 Background

In modern cryptography, block ciphers are fundamental components that are used to ensure data confidentiality in many applications. However, as computational power increases and as machine learning (ML) and deep learning (DL) techniques advance, cryptanalysis must also adapt to ensure further effectiveness. Traditional approaches to break these block ciphers, including differential, linear, and related-key attacks, require large amount of both computational resources and data, which limits their practicality on modern day ciphers.

Research conducted in recent years has demonstrated that ML/DL can significantly change the approach for cryptanalysis [1][2][3][4]. Some models have been trained to distinguish between ciphertexts and random data, score subkey candidates, and learn patterns between plaintext and ciphertext, could potentially make the attacks stronger and increase the number of rounds successfully attacked. Combining classical techniques with ML/DL components, so-called *hybrid attacks*, have been shown to outperform classical methods used on reduced-rounds variants of ciphers such as SPECK, SIMON, and DES.

This project will explore how DL can assist differential cryptanalysis performed on a round-reduced SPN-cipher (Substitution-Permutation Network). The overall goal is to design, implement, and evaluate a pipeline with a hybrid attack model where a DL model comes in to act as a distinguisher and assist with key-recovery. With systematic comparison with a classical method as a baseline model, the project aims to give deeper insight into when and why DL can improve cryptanalysis.

1.2 Goal

The goal of this thesis is to evaluate the effectiveness achieved when including a DL model in a hybrid pipeline for differential cryptanalysis on two variants round-reduced SPN cipher, where one has identical rounds all the way through, and one has a different round structure in the last round. Round-reduced SIMON32/64 will also be studied. The following goals will be pursued:

- Implementing a non-DL attack pipeline of differential cryptanalysis on a round-reduced SPN-cipher
- Develop and train a DL model to act as a distinguisher and key scorer, replacing these parts in the non-DL pipeline
- Compare the baseline attacks against the DL-assisted attacks in terms of success rate
- Compare transfer learning and training from scratch on SIMON32/64 from an already constructed and pre-trained model

With these goals in mind, this thesis will be summarized into to three main research questions to be answered in the end. These are:

1. How does a hybrid attack pipeline compare to purely classical attacks in both classification and key recovery tasks?
2. How does the hybrid pipeline scale with respect to the number of rounds, i.e if at all, up to how many rounds can it maintain advantage over classical methods?
3. How effective is transfer learning across different ciphers compared to training a network from scratch?

1.3 Previous work

Cryptanalysis, which is more described in Chapter 2 and 2.2, plays an important role in the development of cryptographic systems. By using it to analyze and attempting to break ciphers, researchers can find weaknesses and evaluate the continued security of these ciphers. As modern cryptography is widely used in a wide range of applications, including communication systems, finance, and supply chains, the understanding of the current limits of the cryptographic systems is necessary.

Previous work shows the promise and limitations of ML-assisted cryptanalysis. Kim et al. [1] propose that ML-based information theoretic metrics should be used in auditing a cryptosystem with a distinguisher, proven to identify faulty cryptosystems at low computing cost. Hou et al. [3] show that improvement can be made when using ML-aided linear cryptanalysis, by applying it to DES. Comparing their work to previous ones, they show that both success rate and complexity improved compared to classical methods. Wu et al. [2] explore the use of differential cryptanalysis

on round-reduced SIMON32/64 by using ML, which also shows improvement in distinguisher performances and key-recovery rates.

The work done in this thesis is mainly based on the work done by Gohr [4]. In his paper *Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning* he explores how well a neural network can act as a distinguisher, and later combining that with round key recovery attacks. The model he developed is based on SPECK, a block cipher developed by NSA, and the goal is to approach other block ciphers using the same model.

2

Theory

In cryptology there are two different branches, each serving its own purpose: Cryptography and Cryptanalysis.

Cryptography is the field used to design methods to protect information. At its core, it treats a message called the plaintext, as a mathematical object that can be transformed and altered according to various functions, formally known as through a cipher. This alteration of a message through a cipher is called encryption. When the plaintext has been processed through encryption, and thus being encrypted, the resulting message is called a ciphertext. The goal of the process is that the ciphertext should not reveal any information about the original message to parties for whom the message is not intended. Successfully being able to hide the original information is called confidentiality within cryptography. With the correct key, the user can reverse the encryption process in a process called decryption, which results in the retrieval of the original message. The key itself is a secret text that only trusted parties should have access to.

According to the International Organization for Standardization (ISO) [5], there are two more important factors beyond confidentiality: integrity and availability. Integrity is the process of making sure that the encrypted data has not been altered in any way. Availability means that the encryption and decryption is always accessible for the user to access and use.

The ciphers used for the experiments in this thesis belong to symmetric-key cryptography. In symmetric encryption, both encryption and decryption use the exact same key. The counterpart of this is called asymmetric encryption, where encryption and decryption use different keys. This is not studied in this work. Symmetric ciphers are used due to their efficiency [6], and examples of these ciphers include SPN ciphers and Feistel networks, SIMON for example. These are explained later in this chapter.

Cryptanalysis, on the other hand, is the study of analyzing and breaking cryptographic systems. The goal is to identify weaknesses in the cipher to allow exploitation and potential recovery of the information that is intended to be kept a secret. The exploitation can also include the ability to distinguish between random data or if the data come from the cipher. There exist different varieties of cryptanalysis techniques, including differential cryptanalysis and linear cryptanalysis. In this thesis the focus is on differential cryptanalysis. This technique takes a pair of messages

and looks at the difference between them, i.e. which bits are the same and which are not. The difference is then studied in how it propagates throughout the cipher. This is explained in more detail in Chapter 2.2.

2.1 Block ciphers

A block cipher is a symmetric encryption algorithm that processes fixed-length groups of plaintexts. These are called blocks, and defines a deterministic function that maps a plaintext block to a ciphertext block using a secret key of the same length. This function, or encryption, is reversible. The reversing is called decryption and it recovers the original message in doing so.

Formally, for a key k, the encryption is defined as E_k, decryption as D_k, such that

$$D_k(E_k(m)) = m$$

for all plaintexts m [7]. If the plaintext would be longer than the fixed-length in the cipher, the plaintext is split into multiple blocks and processed individually.

Block ciphers are typically created as iterative functions consisting of multiple rounds. For each round, the message is transformed according to the functions each round contains. Repeating the amount of rounds iteratively increases the complexity of the cipher. The goal of the repeating of rounds is to make the cipher ideal, and that the ciphertexts produced looks like random permutation. This would make each ciphertext appear with random probability, or $\frac{1}{2^b}$ where b is the number of bits in the block. Each round introduces a round key which acts as the randomness in the cipher. The round key is usually derived from a master key along with a fixed set of rules. Using the same master key every time would then result in the same round keys every time.

The designing of block ciphers takes two principles into account, confusion and diffusion, introduced by Shannon originally in 1945 [8]. Confusion is the obscuring of the correlation between the plaintext and the ciphertext by applying different keys throughout the cipher. Diffusion is the spreading of the influence of each input bit across the output.

2.1.1 SPN ciphers

Substitution-Permutation Networks (SPNs) are a class of symmetric block ciphers where each round consists of substitution, permutation, and key addition. SPN-based ciphers are widely used in modern cryptography, and simplified versions are commonly used in cryptanalysis studies and literature, such as in Heys tutorial on cryptanalysis [9].

Let x^{r-1} denote the input to round r. A generic SPN round can be described as

$$\begin{aligned} u^r &= x^{r-1} \oplus k^r \\ v^r &= S(u^r) \\ x^r &= P(v^r) \end{aligned}$$

where k^r is the round key, S the substitution function, and P the permutation layer. The substitution layer is implemented by using substitution boxes, or S-boxes. An S-box contains a mapping

$$S : \{0,1\}^m \rightarrow \{0,1\}^m$$

where m is the bit size of the S-box. This function transforms the input block into a corresponding output block according to a lookup table. An example of this table can be seen in Table 3.1. In many SPN designs, and also the ones used in this thesis, divides the plaintext into equal-sized subblocks, and applies the same S-box independently on each of the subblocks.

After substitution, the internal state is processed by a permutation layer. The permutation is a function

$$P : \{0,1\}^n \rightarrow \{0,1\}^n$$

where n is the bit size of the permutation. It rearranges the bit positions according to a set rule, which for example can be according to a table (example Table 3.2), or as a bit-shift which shifts all bits to the right or left.

This round structure makes an SPN cipher suitable for studying how different inputs propagate bit-wise through each round.

2.1.2 SIMON cipher

The National Security Agency (NSA) released the SIMON cipher family to the public in June 2013 [10]. The cipher is a lightweight block cipher, meaning that it is computationally cheap and suitable for small devices with limited computational power [5]. SIMON uses arithmetic functions like XOR and shifting of bits, combined with a key scheduler for each round of encryption. The main purpose of developing this cipher was the increasing need of securing IoT devices which usually have limited hardware and computational capacity. This cipher is designed to be light and quick on limited hardware. The cipher became an ISO standard in 2018 (ISO/29167-21), together with Speck (ISO/29167-22) which was developed at the same time, and is now a standard part of the RFID air interface standard widely used in supply chain and warehousing.

The cipher comes in different variants where the size of input, size of key, and number of rounds differs. The naming convention for the variants is SIMON XX/YY, where XX represents the input size in bits, and YY represents the key size in bits. In Table 2.1 the different versions available is shown.

Block size (bits)	Key size (bits)	Rounds
32	64	32
48	72	36
	96	36
64	96	42
	128	44
96	96	52
	144	54
128	128	68
	192	69
	256	72

Table 2.1: SIMON block and key size configurations

The structure of SIMON consists of r identical rounds, where the data is split into two equally big parts L and R (Left and Right). In each round, certain operations takes place on each of these halves. These operations are: AND &, bit-wise left rotation S^i where i indicates how many bits to shift, and XOR $\oplus$. Let L_r denote the left half of the round input text, R_r denote the right half of the round input text, and k_r denote the round key. As shown in Figure 2.1, the two outputs of a single round can be calculated as:

$$R_{r+1} = L_r$$
$$L_{r+1} = R_r \oplus (S^8(L_r) \& S^1(L_r)) \oplus S^2(L_r) \oplus k_r$$

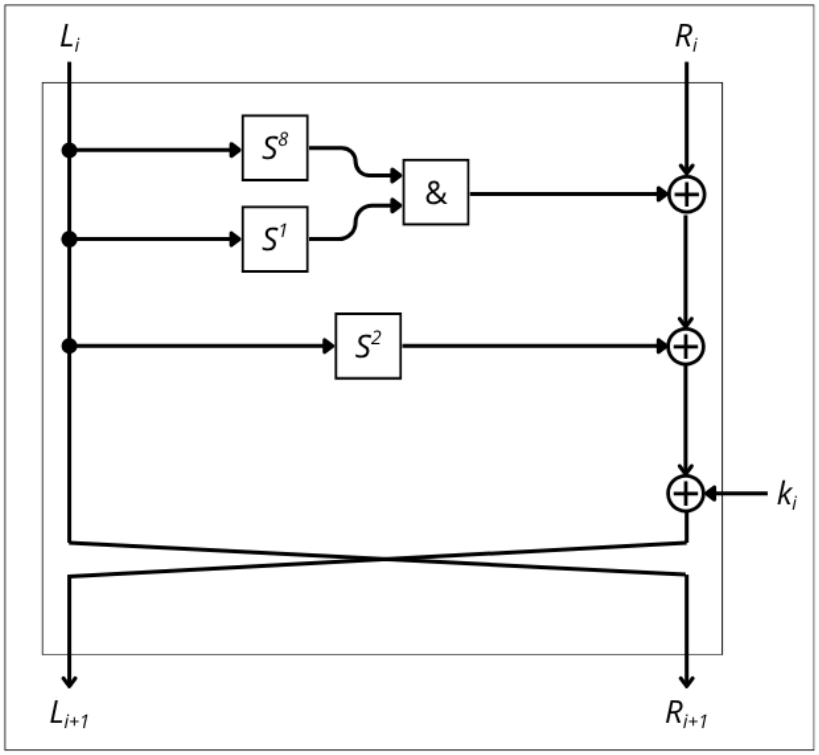

Figure 2.1: A visual on an encryption round in SIMON cipher.

2.2 Differential Cryptanalysis

Differential cryptanalysis, first introduced by Biham and Shamir [11], is one of the classical cryptanalysis techniques on block ciphers. Classical differential cryptanalysis is performed by finding differences between pairs of input plaintext and pairs of output ciphertext that propagates through the block cipher, and that these differences deviates from random chance. The differences are thereafter exploited to recover subkeys or to distinguish between random data and actual cipher output. This type of cryptanalysis often requires a large number of pairs of plaintext and ciphertext, and large computational power.

The differential cryptanalysis works as follows: Let $E_K(P)$ denote the encryption of a plaintext P using the secret key K. The attacker chooses pairs of plaintexts (P, P') such that:

$$P' = P \oplus \Delta P$$

where ΔP is a chosen input difference and $\oplus$ denotes bitwise XOR. Both plaintexts are then encrypted using the same key K to produce ciphertexts:

$$C = E_K(P), \quad C' = E_K(P')$$

The corresponding ciphertext difference is given by:

$$\Delta C = C \oplus C' = E_K(P) \oplus E_K(P')$$

If the output difference ΔC occurs with a probability p significantly greater than 2^{-n} (where n is the block size), then the pair $(\Delta P, \Delta C)$ forms a *differential characteristic*. These characteristics allow the attacker to distinguish the cipher from a random permutation and to recover parts of the key by analyzing how these differences propagate through the rounds of the chosen cipher.

2.3 Machine learning components

A machine learning model consists of various layers put together to perform a classification task. The components and their settings can vary. Here follows the different components introduced in this thesis, and the theory behind them.

Convolutional layer

A convolutional layer processes the data by applying a small filter that slides across the data to extract local patterns it finds. They can come in different dimensions, and the ones used in this thesis are one dimensional. The filter slides across the one dimensional input word and extracts patterns and correlations it finds between neighboring bits. For cryptanalysis, this kind of layer is useful due to that many of the cipher operations use structured dependencies. These operations can include XOR's, rotations, permutations, and substitutions. By capturing these and propagating them further in the network, the model itself can approximate global behavior even though the layers themselves capture local behavior.

ResNet, residual block, and residual tower

A Residual Network, or ResNet, is a network containing residual blocks. When these blocks gets chained together, they form a residual tower. The number of residual blocks is called the depth of the tower. A residual block contains l amount of convolutional layers, often combined with a normalization and activation after each. What makes it a residual block is the skip connection that is added. The skip connection is added between the input and the output of the block, which allows the network to preserve useful already learned information from previous parts of the network. Figure 2.2 shows how the skip connection is added. Instead of the block having the output $y = F(x)$ as it would be without the connection, the resulting output becomes $y = F(x) + x$. Chaining them together creates a residual tower, and can be any numbers of blocks, called the depth. The important advantage of this is helping with the vanishing gradient problem, which is where gradients become too small in deeper networks and training becomes more difficult. The flow of gradients flowing through both the convolutional layers and the skip connection therefore creates more reliable learning. ResNet was developed in 2015 for image recognition **resnet**, but has proven beneficial in cryptanalysis also due to its ability to find patterns across many bits.

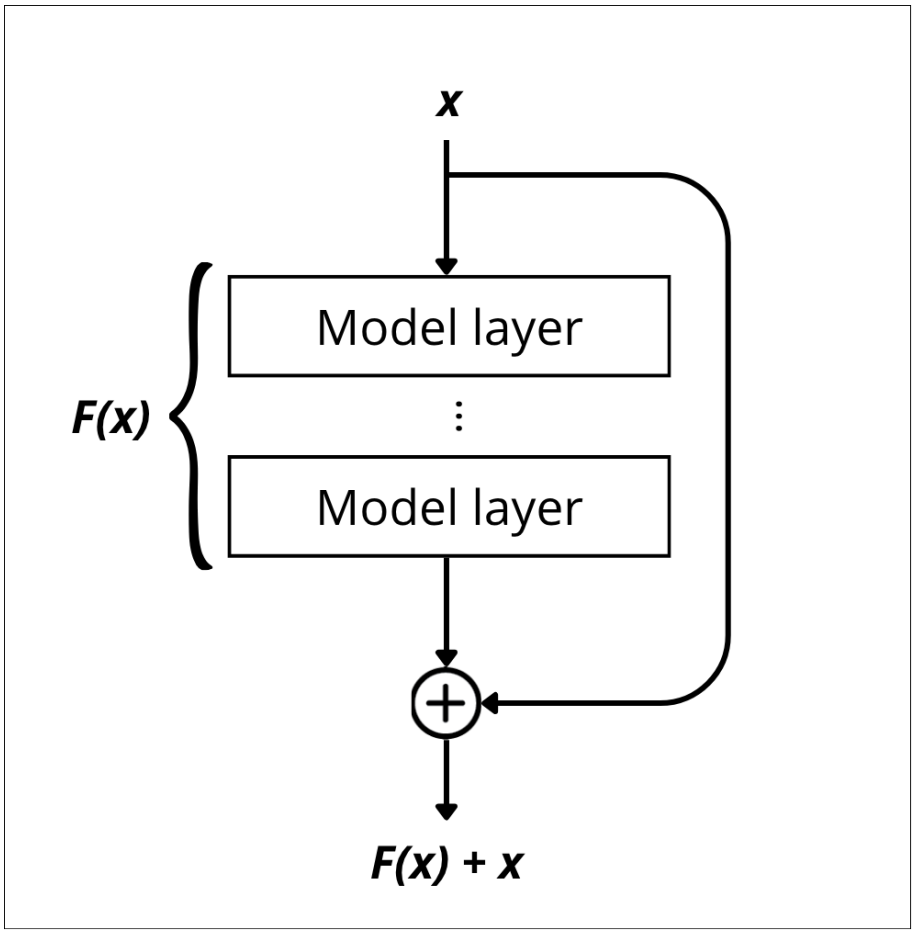

Figure 2.2: A typical residual block with a skip connection.

Optimizer, loss function, and activation

Some of the defining properties of how a neural network learns and makes predictions are the chosen optimizer, loss function, and activation function. The loss function measures how well the model guesses correctly according to the labels, which provides a guiding signal in the learning. The loss function used in this thesis is called *Mean Square Error (MSE)*, which measures the average squared difference between the predicted value and the actual label. The optimizer uses the signal from the loss function to change and optimize the parameters in the model iteratively during training. The one used here is called *Adam* [12] and is a gradient-based optimizer which adapts each parameter individually whilst keeping track of estimates of both the mean and variance of gradients. Lastly the activation function is what makes the prediction in the output layer. For this model, the *Sigmoid* function is used $\sigma(x) = \frac{1}{1+e^{-x}}$, and the result can be interpreted as the probability that the given input belongs to a particular class.

Hyperparameters

When setting up a ML model, there are multiple parameters that can be changed and adjusted for better performance. The ones used in this thesis are explained here:

- Depth: The number of residual blocks in the residual tower
- Epochs: The number of iterations the ML model trains and try to improve on its previous iteration
- Cyclic learning rate: A range of learning rate showing how small or large changes the model does to its internal weights between each epoch
- Regularization parameter: This controls the strength of the penalty added to the loss function to prevent overfitting the model
- Dropout: A percentage of nodes which are turned off in each epoch, at random, which can help with finding patterns and reduce overfitting
- Dense layer units: the amount of neurons in the dense layers in the end of the model, which controls the decision of classifying the input
- Input shape: The shape of the data required for the model to be able to process it

3

Method

This chapter describes the methods used in this thesis, and how they explore and evaluate machine learning-assisted differential cryptanalysis through an experimental pipeline. It consists of four main stages. First, plaintext pairs are generated with a specified input difference along with pairs of random relationship. Second, the pairs are encrypted using one of the specified ciphers. Third, the resulting ciphertexts are used in training the neural distinguisher. Fourth and last, the trained distinguisher is evaluated both as a classifier and, when applicable, as a component in the key recovery attack.

Each of the stages is designed to be modular, which allows controlled changes and comparisons between different configurations. A visual overview of the pipeline is shown in Figure 3.1.

3.1 Data generation

A lot of data samples are needed to be able to train a neural network, and for that there is a need of easy data generation and data management. To be able to reproduce and compare the different experiments performed later, a way of storing the data was needed instead of generating it from scratch for every experiment. A HDF5 database was chosen as the means of storing the data due to its ability to easily label the data, quick read and write times, and its compatibility with NumPy functions. Depending on which cipher is going to be used, the exact attributes needed can differ. The example that follows takes into account what is needed for the SPN cipher that will be used, which is described later in this chapter. The attributes needed for each entry in this case are: *Plaintext 1 (p1), Plaintext 2 (p2), Master key, Label, Ciphertext 1 (c1), and Ciphertext 2 (c2).* The number of entries and the shape of the data in each dataset can be modified as needed. An example of a HDF5 database creation can be seen in Algorithm 1.

The next step after creating the database is to populate it with data. The generation for all plaintexts was done using the Python module *random*, which generates random numbers to a set bit length. The generation parameters are the number of samples N, the number of rounds R, the word size as the number of bits WS, and the desired plaintext XOR difference Δ as hexadecimal representation. The program generates one plaintext of length WS at random and XOR's that plaintext with the specified delta, which generates the second plaintext of that pair. This pair

Algorithm 1: Dataset Creation

Input: Sample size N, Number of rounds R

Create/overwrite file `dataset.h5` in write mode
    Create datasets:
        `p1`, shape (N)
        `p2`, shape (N)
        `keys`, shape $(N, R+1)$
        `label`, shape (N)
        `c1`, shape (N)
        `c2`, shape (N)
Close file

is labeled 1, signaling that this is a plaintext pair fulfilling the property of a *real* pair. Although the plaintext generation is random, it is beneficial to be guaranteed to include a set of extraordinary pairs that fulfill certain patterns, as this might help the neural distinguisher training in identifying patterns from these. Example of these are when `p1` = all 0's, all 1's, alternating between 0's and 1's, only 1 bit active, and so on. The full list of extraordinary `p1's` used can be seen in Table A.1. For random pairs, two plaintexts were generated at random with no connection to each other. This pair is labeled 0, signaling that this is a completely random pair of plaintexts. An array of R random WS keys was also generated to be used in the SPN cipher, where each element of the array is considered a specific round key. A visual representation of this generation can be seen in *Step 1* in Figure 3.1, and the algorithm for this step is shown in Algorithm 2. The first for-loop fills the database with the extraordinary plaintexts. The second for-loop fills the remaining true pairs up to half of the sample size. The third for-loop fills the database with half of the sample size with false pairs. Generating half of each kind of labeled pair ensures a balanced dataset which the model can use for training.

3.2 Cipher implementations

As mentioned in Chapter 1, two different ciphers are used in this thesis: a small SPN cipher and SIMON32/64. They were both chosen because they belong to the block cipher family. The SPN cipher provides a simple and light-weight environment in which the differential behavior and key recovery attacks can be analyzed. SIMON32/64 was chosen because of the similarity to SPECK32/64 for which Gohr [4] made his work. By studying both ciphers, it became possible to observe how well the ML distinguisher adapted to different structures and data formats, when originally intented for another cipher.

3.2.1 SPN cipher variants

The first cipher used in this work is the substitution-permutation network (SPN) which Heys introduces in his paper *A tutorial on linear and differential cryptanalysis*[9]. The structure of the cipher is inspired by design choices introduced by Horst

Algorithm 2: Data generation

Input: Sample size N, Number of rounds R, Word size WS, Input difference Δ, Extraordinary plaintexts [$ep1$]

Initialization: `p1[], p2[], label[], key[]`
for ep1 in [$ep1$]:
    `p1` = ep1
    `p2` = $ep1 \oplus \Delta$
    `label` = `1`
    `key` = randomize $R + 1$ keys of WS bits
    Append all values to its array
for _ in range $N/2 - len([ep1])$:
    `p1` = random WS bits
    `p2` = `p1` $\oplus \Delta$
    `label` = `1`
    `key` = randomize $R + 1$ keys of WS bits
    Append all values to its array
for _ in range $N/2$:
    `p1` = random WS bits
    `p2` = random WS bits
    `label` = `0`
    `key` = randomize $R + 1$ keys of WS bits
    Append all values to its array
Open file `dataset.h5` in access mode
    Save all values in database
Close file

Feistel in 1973 [13], and shows important components of modern block ciphers, although in a simplified setting in Heys paper where he says that such a simplified construction still provide valuable insight into the security properties of more complex ciphers.

The substitution layer in this cipher splits the 16-bit text into four 4-bit nibbles, each of which is going through a separate S-box. For this cipher, the same S-box table is used (Table 3.1). The permutation layer takes the resulting 16-bit text and permutes it bit by bit into a new 16-bit text. This layer uses the same table for permutation (Table 3.2) much like the substitution layer.

A key design aspect in this thesis is that two different variants of the SPN cipher are considered. The first variant follows Heys design completely. The final round in this design differs slightly compared to the previous round, and consists of round key addition, a substitution layer, and another round key addition. The output after this step is the final ciphertext of the cipher. A visual overview of the cipher is shown in Figure 3.2, where *round 4* indicates the last round structure. The second variant is a modified version, in which all rounds are kept identical, including the last round. These rounds follow the same structure as mentioned above, with XOR'ing the key, then substituion with an S-box, followed by permutation, and lastly another XOR

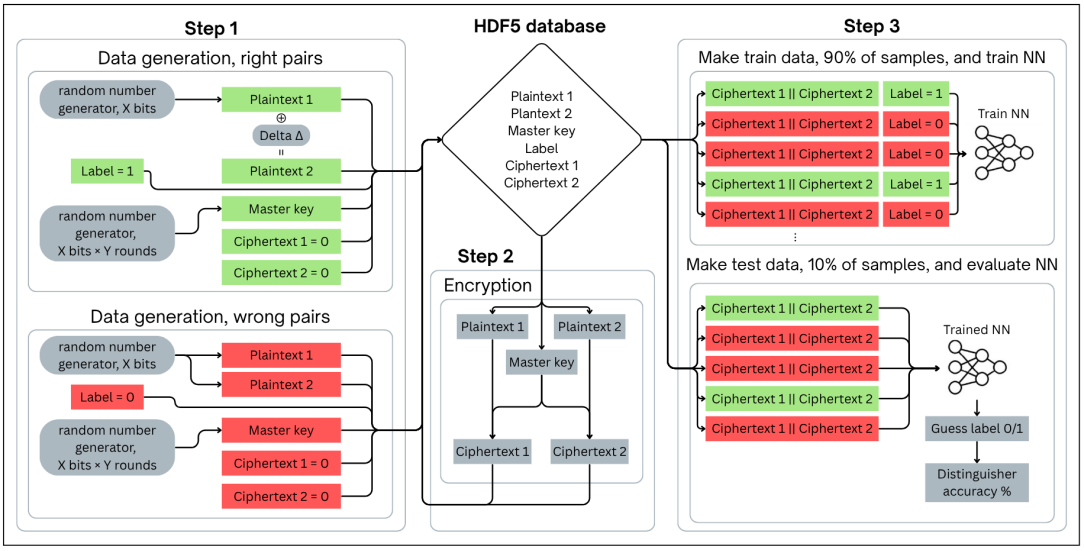

Figure 3.1: The flow from data generation to trained distinguisher.

with the last round key. This distinction between the two variants were made to analyze the behavior when the structure differs depending on having a differing last round or when the same structure is kept all through the cipher.

To select a suitable plaintext XOR differenc for the experience, a Difference Distribution Table (DDT) must be created. It is created by looking at each possible XOR difference for the size of the S-box input, and what the resulting XOR difference will be at the output. In Table 3.3, the inputs x and x^* resulting in XOR difference `1011` is shown in the two left columns. By using the S-box table showing in Table 3.1, the resulting outputs y and y^* can be derived, and thereafter their XOR difference showing in the rightmost column. By counting the occurrence of each output difference, a bias can be found. In this case, `0010` appears in 8 out of 16 times, showing a strong bias towards it. By creating similar tables for each possible input difference and counting the output difference occurrences, the DDT can be derived showing in Table 3.4. The DDT can then be used to calculate probabilities of specific values propagating throughout each round. For the first round, input difference `1011` shows the strongest bias, and therefore it was chosen as a starting point in the experiments.

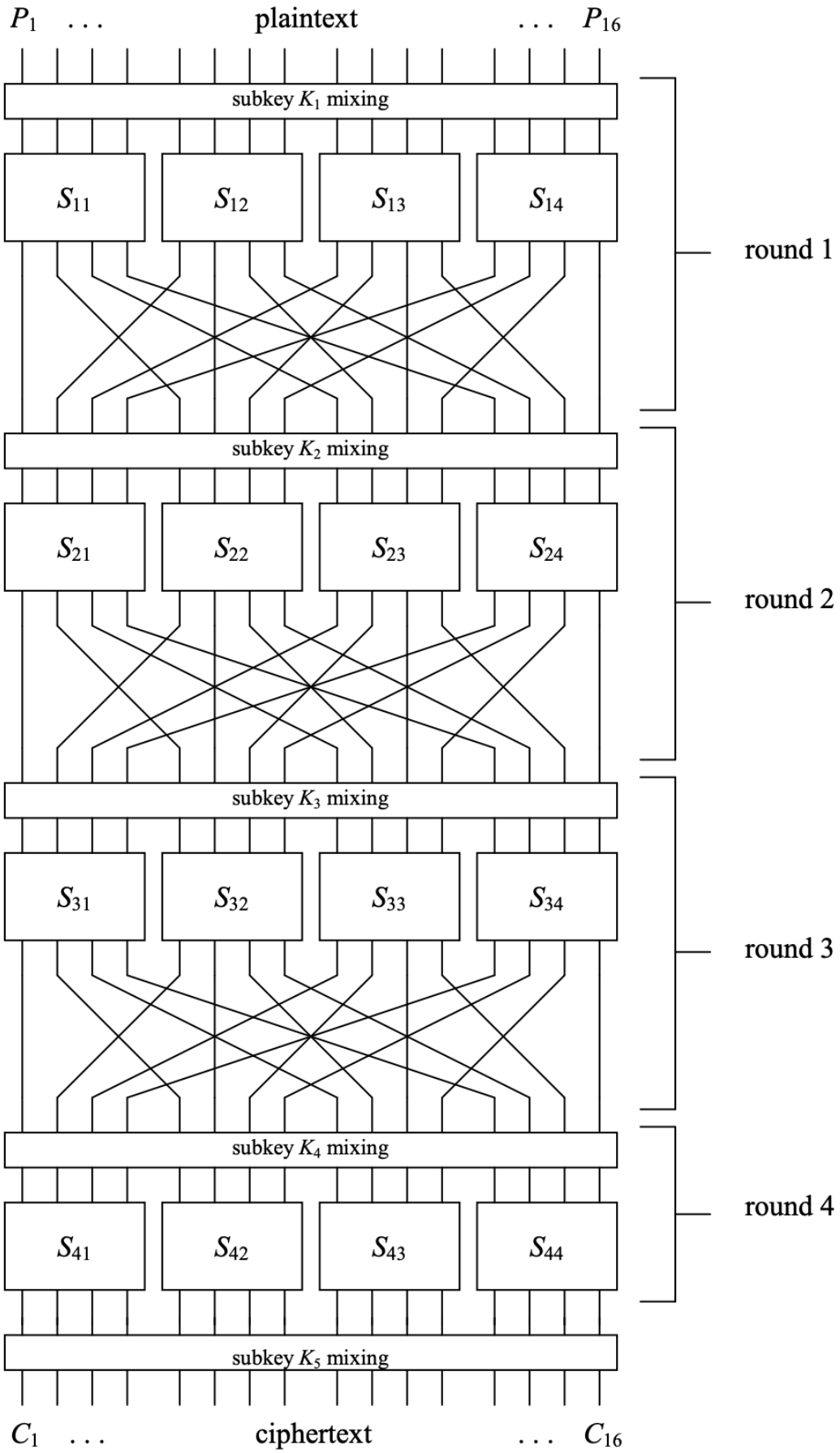

Figure 3.2: A visual on the SPN cipher used for encrypting and decrypting. Reprinted from *A tutorial on linear and differential cryptanalysis* by Howard. M. Heys [9]

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Table 3.1: S-box table (in hexadecimal) from the cipher used by Heyes

input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
output	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

Table 3.2: Permutation table (in bit index) from the cipher used by Heyes

$\mathbf{x}$	$\mathbf{x}^*$	$\mathbf{x} \oplus \mathbf{x}^*$	$\mathbf{y}$	$\mathbf{y}^*$	$\mathbf{y} \oplus \mathbf{y}^*$
0000	1011	1011	1110	1100	0010
0001	1010	1011	0100	0110	0010
0010	1001	1011	1101	1010	0111
0011	1000	1011	0001	0011	0010
0100	1111	1011	0010	0111	0101
0101	1110	1011	1111	0000	1111
0110	1101	1011	1011	1001	0010
0111	1100	1011	1000	0101	1101
1000	0011	1011	0011	0001	0010
1001	0010	1011	1010	1101	0111
1010	0001	1011	0110	0100	0010
1011	0000	1011	1100	1110	0010
1100	0111	1011	0101	1000	1101
1101	0110	1011	1001	1011	0010
1110	0101	1011	0000	1111	1111
1111	0100	1011	0111	0010	0101

Table 3.3: The three left columns are showing all different combinations of input plaintexts corresponding to XOR difference 1011. The three columns to the right show the corresponding output values after passing through an S-box, and the XOR difference between those two outputs.

Input Diff.	Output Difference															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

Table 3.4: Difference Distribution Table

3.2.2 SIMON 32/64

The second cipher used in this thesis is SIMON32/64. This variant takes a 32-bit input together with a 64-bit key, and produces a 32-bit ciphertext. As mentioned in Chapter 2.1.2, SIMON does not use substitution tables or bit permutations as the SPN cipher does. Instead, it is based on bitwise operations such as XOR, AND, and rotations.

SIMON32/64 was chosen for two main reasons. Firstly, it is a lightweight block cipher that has been studied in previous works [2]. Secondly, the structure is similar to SPECK32/64 which is the cipher used by Gohr in his work [4]. Therefore it can be used in experiments on both transfer learning, and training Gohr's model from scratch.

The Python implementation used for SIMON32/64 was taken from a public repository with MIT license [14]. The number of rounds was changed to be modular instead of fixed, so a round-reduced variant could be used in the different experiments.

3.3 Neural distinguisher model

The model proposed by Gohr in his paper *Improving Attacks on Round-Reduced Speck32/64 using Deep Learning* [4] is based on a residual convolution neural network. The architecture of the model is designed to act as a distinguisher on ciphertext pairs, usually from a round-reduced block cipher. It combines bit-sliced convolutions with a residual tower and fully connected prediction head.

Input representation: Each input to the model consists of two concatenated ciphertexts that make up the ciphertext pair on which the model is trained. Gohr is using SPECK 32/64 as cipher in his paper, and since it operates on 32 bit plaintexts/ciphertexts, the total input to the model becomes 64 bit. The model reshapes these into 4×16 bit words in the first step. This representation preserves the left/right word structure of SPECK 32/64, which allows the model to learn differential patterns on these separated words. Since SIMON 32/64 keeps the same word structure as SPECK 32/64, no modifications were needed for the architecture. However, for the SPN the input representation was changed to 2×16 because the cipher operates on the whole 16 bit word at a time.

Bit-sliced convolution: After the input reshaping, the data is processed by a bit-sliced convolutional layer consisting of 32 output channels. Having a representation like this makes each bit position be treated as a feature dimension. This treatment helps the model to learn potential correlations between bit positions in the ciphertext pairs. This feature is suitable for cryptanalysis on these kinds of ciphers since differences propagate on bit-level throughout.

Residual tower: The main part of the model consists of a residual tower. The tower consists of a sequence of residual blocks, and the number of blocks is set freely by the user. Each residual block contains two convolutional layers, each with kernel size 3. Each layer is followed by batch normalization and rectified linear unit (ReLU) activation. Each residual block contains a skip connection between its input and output. This connection enables more stable training of deeper networks and helps mitigate vanishing gradient issues. The more blocks the tower contains, the more complex patterns could potentially be captured by the model.

Prediction head: The output of the residual tower is first flattened and then input to two densely fully connected layers with 64 units each, where each layer is also followed by batch normalization and ReLU activation. The final layer is a single neuron output with sigmoid activation. This produces a probability and estimation to whether or not the input ciphertext pair came from the cipher or if it was random noise.

3.4 Key recovery attack

Once a distinguisher for x rounds has been trained to distinguish real ciphertext pairs from random noise with accuracy significantly above random chance (0.5), it can be used to perform a key recovery attack on $x + 1$ rounds. This would take a

more classical approach as described by Heys [9], but replace the manually statistical method with an ML distinguisher.

3.4.1 Attack overview

Let $E_k^{(r)}(P_i)$ denote encryption under key k for r rounds of plaintext P_i.

Let $D_{k'}^{-1}(C_i) = C_i^{-1}$ denote decryption of the last round using key guess k'.

The goal is to recover subkey k_r used at round r, which is the last round, by using a distinguisher trained on $r-1$ rounds.

A set of N plaintext pairs that fulfills the property $P_{0,i}, P_{1,i} = P_{0,i} \oplus \Delta$, where Δ is a specified difference, is encrypted under a fixed master key producing the ciphertext pairs:

$$(C_{0,i}, C_{1,i}) = (E_k^r(P_{0,i}), E_k^r(P_{1,i}))$$

For each subkey candidate k' from a set of key guesses $[K']$, decryption of the last round is performed:

$$(C_{0,i}^{-1}, C_{1,i}^{-1}) = D_{k'}^{-1}(C_{0,i}, C_{1,i})$$

This creates $N \times [K']$ of partially decrypted ciphertext pairs. If $k' = k_r$, the partially decrypted pair should resemble the same properties as the ciphertexts from encrypting $r-1$ rounds, and behave as random noise otherwise.

3.4.2 Scoring the key guesses

The trained distinguisher $X(C_0, C_1)$ outputs a value $S \in [0, 1]$ for any given ciphertext pair, that represents the probability that the given ciphertext pair originates from the r-round cipher, where 0 = random noise and 1 = ciphertexts from the cipher. For each of the subkeys k', the aggregated score is denoted as:

$$S(k') = \sum_{i=0}^{N-1} log \frac{X(C_{0,i}^{-1}, C_{1,i}^{-1})}{1 - X(C_{0,i}^{-1}, C_{1,i}^{-1})}$$

By using Log-Likelihood Ratio, the correct confident guesses are amplified and the wrong confident guesses are penalized. Over many samples, it helps to create a more clear separation between the right and wrong guesses.

3.4.3 Pseudo-code

Algorithm 3 shows the procedure of the key recovery attack used.

Algorithm 3: Key recovery using trained distinguisher

Input: Ciphertext pairs $(C_{i,0}, C_{i,1})_{i=1}^{N}$, trained distinguisher X, key space K of round key candidates k', scoring function $s(ciphertext\_pair)$

Output: Top 10 guesses of subkey k^*

foreach $k' \in K$ **do**
  $key\_score = 0$
  for $i = 1$ **to** N **do**
    $(C_{i,0}^{-1}, C_{i,1}^{-1}) = \text{DecryptLastRound}(C_{i,0}, C_{i,1}, k')$
    $s = log\frac{X(C_{i,0}^{-1}, C_{i,1}^{-1})}{1-X(C_{i,0}^{-1}, C_{i,1}^{-1})}$
    $key\_score = key\_score + s$
  end
  $final\_scores[k'] = key\_score$
end
$[k^*] = TopTen(final\_scores)$
return k^*

3.5 Experimental design

This thesis explores several experimental designs aimed at evaluating different aspects and approaches of cryptanalysis combined with neural network models. This section will explain each of the experiments and how they were performed.

3.5.1 The classic distinguisher

To be able to identify differential characteristics using classical differential cryptanalysis, a statistical baseline must be established. This baseline should correspond to an ideal cipher and modeled as random permutation, where all output differences occur with equal probability. According to Heys [9], for a specific input difference ΔP, the expected number of occurrences for a specific output difference ΔC for N plaintext pairs is:

$$\mathbb{E}[X] = \frac{N}{2^b}$$

where X is the number of observed occurrences of a specific ciphertext difference, and b is the number of bits in the block size of the cipher. A ciphertext difference ΔC is considered a differential characteristic if the output difference occurs significantly more often than the baseline, i.e: $X \gg \mathbb{E}[X]$ indicates a non-random differential characteristic. Two main methods can be used to evaluate whether X fulfills this property.

The first method considers a single output difference and models X as a binomial random variable under the null hypothesis of a random permutation:

$$X \sim Binomial(N, p_0), \text{where } p_0 = 2^{-b}$$

For sufficiently large N, the binomial distribution can be approximated by a normal

distribution, which allows the use of a z-score:

$$z = \frac{X - Np_0}{\sqrt{Np_0(1-p_0)}}$$

This z-score shows how many standard deviations the observed count deviates from the expected value. A large z indicated that the output difference ΔC occurs more frequently than random permutation, classifying it as a differential characteristic.

The second method considers the full distribution of output differences using chi-squared statistics (χ^2). This statistics is given by:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed count of input difference i, and E_i is the expected number of occurrences $\frac{N}{2^b}$ as mentioned above. This method is computationally more expensive, but takes the whole distribution into account at the same time. The computed χ^2 value is then compared to a pre-defined threshold. If the value is larger than the threshold, the distribution is classified as output of the real cipher. Otherwise, it is classified as random noise.

In this work, both methods are used in the experiments. The single output difference method is used for the key recovery attack, where each guess is queried one by one, making this method suitable. The whole distribution method is used as a comparison to the ML distinguisher in the ability to classify encrypted ciphertext pairs as real pairs or random permutation.

3.5.2 Training Gohr's model on SPN

This approach adapts the model proposed by Gohr, described in Chapter 3.3. In its original form, the model takes 64-bit inputs, which corresponds to the pairs of ciphertexts from the cipher SPECK32/64 that is used in Gohr's paper. The structure of SPECK32/64 consists of two 16-bit words, which leads to the input representation of 4×16 in the model.

In contrast to this, the SPN cipher used in this work operates on a single 16-bit block, making the pairs consist of 32 bits in total. Because there is no left/right structure, the input structure is modified to a 2×16 matrix, where each row represents one ciphertext in the pair. The columns represent the bit positions.

Altering the input layer is the primary modification done to the architecture to fit the new cipher. The other components of the model are initially kept according to the original design, such as the residual block or the prediction head. However, their parameters are later adjusted during the training phase to better match the change in input size and the cipher's behavior.

3.5.3 Performing the key recovery attack

After training distinguishers on both variants of the SPN cipher, a key recovery attack was performed on one additional round beyond the number of training rounds.

The attack follows the method described in Chapter 3.4. The purpose of this experiment is to evaluate whether the trained distinguishers performs sufficiently to identify the correct last round-key among all guesses. The output of the experiment is therefore a ranked list of key guesses.

This experiment is particularly important to this thesis, since it directly shows how well a ML distinguisher can be combined with a key recovery attack. It evaluates more than only accuracy, and if integrating it with another technique will work. By comparing the results between the two SPN cipher variants, the experiment also provides insight into how the different round structures affect the effectiveness of the combination of ML distinguisher and key recovery attack.

3.5.4 Training Gohr's model on SIMON 32/64

For this approach, no modification of the input representation was required. The model is designed for ciphertext pairs of SPECK32/64, where each ciphertext consists of two 16-bit words. Since SIMON32/64 follows the same structure, keeping the input representation as 4×16 is reasonable.

The similarity between the two cipher structures allows the model to be applied to SIMON32/64 without any further modifications either. The training could therefore be performed using the original architecture, with no changes done to the hyperparameters for a strict comparison between the two classes of ciphertext data.

3.5.5 Fine-tuning a pre-trained model

The final experiment in this thesis investigates transfer learning. Instead of training the model from scratch, the pre-trained model provided by Gohr in his repository is loaded together with its weights and then fine-tuned on SIMON32/64 ciphertexts, instead of the SPECK32/64 ciphertexts it is trained on to begin with.

Once the pre-trained model was loaded, all the layers in it were frozen meaning that they stay exactly as they are during training. Then, iteratively, layer by layer were unfrozen and re-trained on the new ciphertext data. This was performed in three steps. First, the output layer was unfrozen and re-trained, then the output layer and the last dense layer, and lastly the output layer together with both dense layers. This step-by-step approach allowed evaluation in between each training of how much modification is needed before any useful transfer learning was observed.

This method was only performed on ciphertexts produced by SIMON32/64 since they directly match the structure of the ones from SPECK32/64.

4

Results

4.1 Classical distinguisher

Table 4.1 and 4.2 shows the classification results of the χ^2 distinguisher, when fed all the distributions of one to seven rounds of encryption, for both versions of the SPN cipher.

Rounds	Cipher or random
1	Cipher
2	Cipher
3	Cipher
4	Cipher
5	Cipher
6	Cipher
7	Random

Table 4.1: Classification of the chi-squared distinguisher on Heys version of the SPN cipher.

Rounds	Cipher or random
1	Cipher
2	Cipher
3	Cipher
4	Cipher
5	Cipher
6	Cipher
7	Random

Table 4.2: Classification of the chi-squared distinguisher on the modified version of the SPN cipher.

Figures 4.1 to 4.8 shows the distribution of the occurrences of each output XOR difference, when counting the whole dataset. The first four shows the distributions for different rounds on Heys variant of the SPN cipher, and the last four shows the distribution for the modified SPN variant.

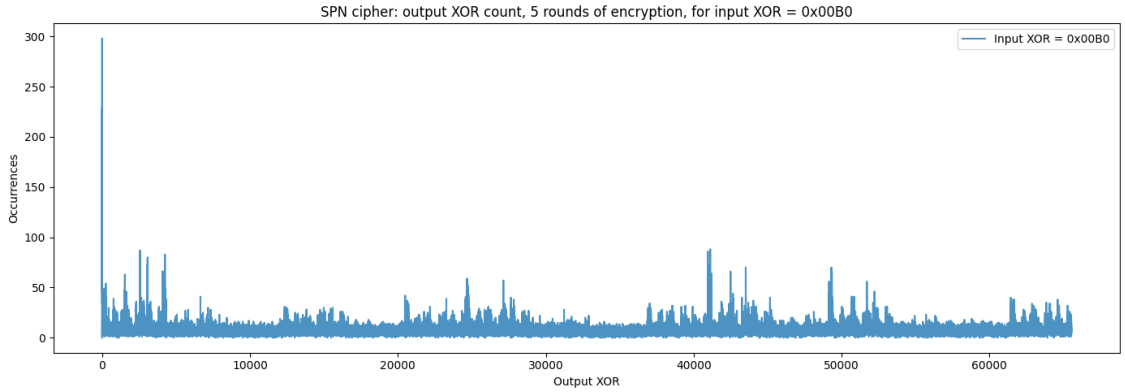

Figure 4.1: Number of occurrences of XOR outputs for target input difference. Heys SPN cipher, 5 rounds.

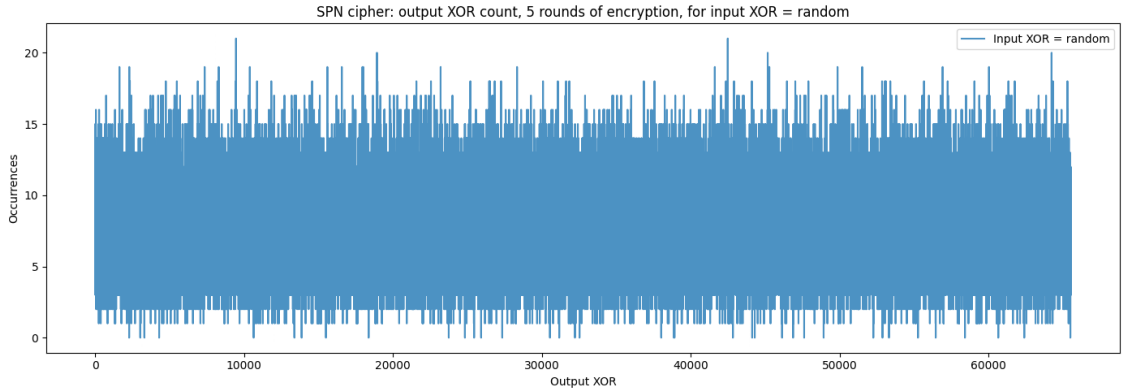

Figure 4.2: Number of occurrences of XOR outputs for input pairs with random XOR difference. Heys SPN cipher, 5 rounds.

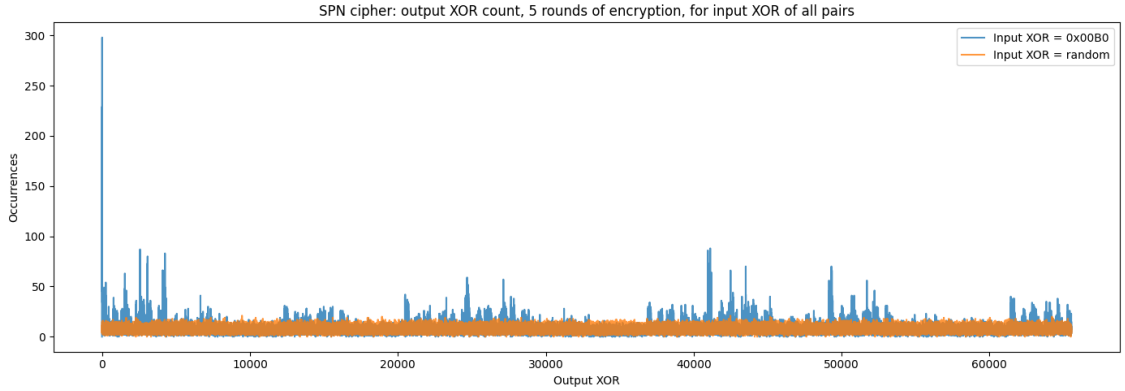

Figure 4.3: Number of occurrences of XOR outputs for all pairs, both with target input difference and with random differences. Heys SPN cipher, 5 rounds.

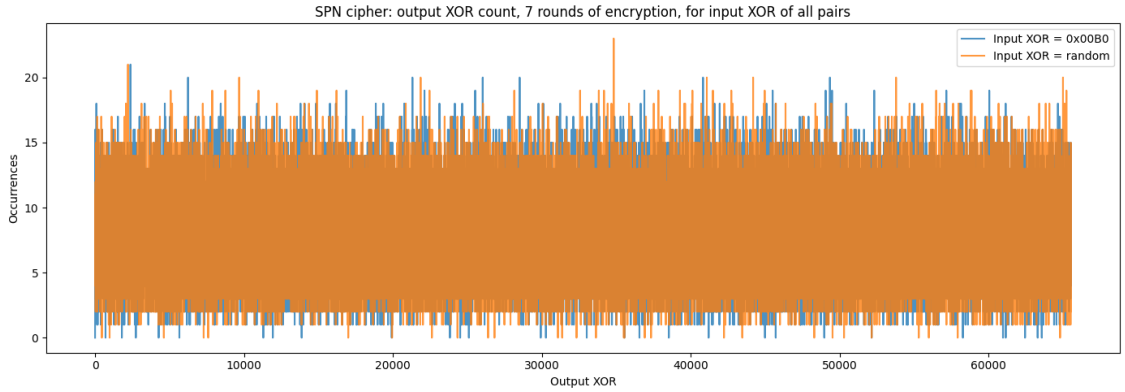

Figure 4.4: Number of occurrences of XOR outputs for all pairs, both with target input difference and with random differences. Heys SPN cipher, 7 rounds.

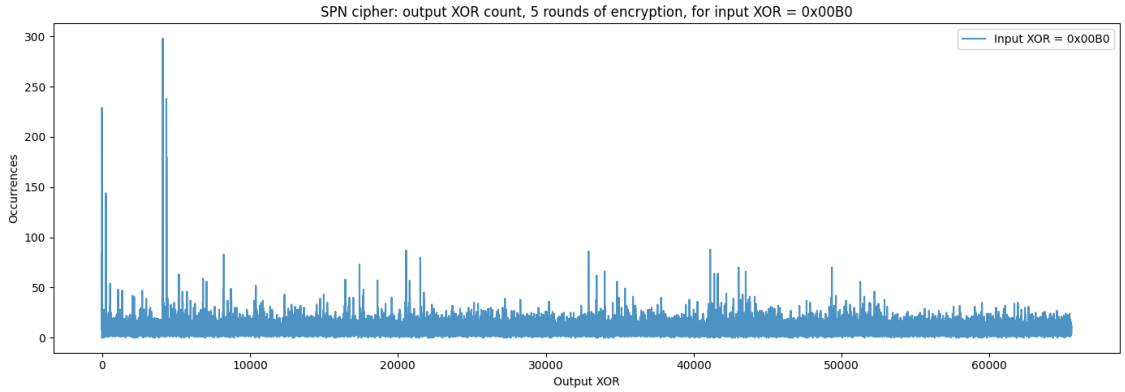

Figure 4.5: Number of occurrences of XOR outputs for target input difference. Modified SPN cipher, 5 rounds.

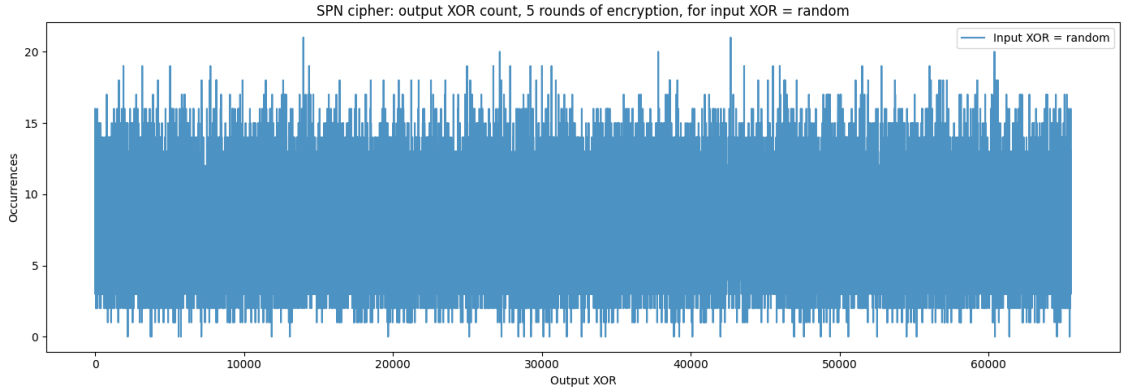

Figure 4.6: Number of occurrences of XOR outputs for input pairs with random XOR difference. Modified SPN cipher, 5 rounds.

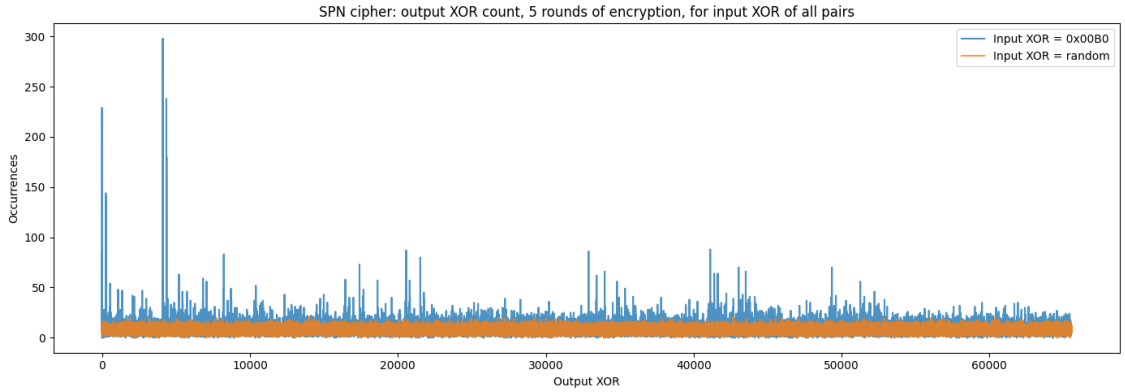

Figure 4.7: Number of occurrences of XOR outputs for all pairs, both with target input difference and with random differences. Modified SPN cipher, 5 rounds.

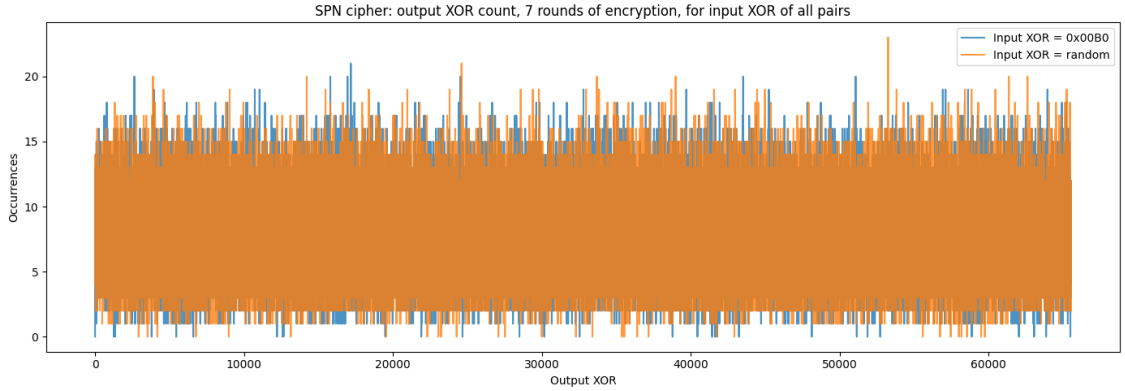

Figure 4.8: Number of occurrences of XOR outputs for all pairs, both with target input difference and with random differences. Modified SPN cipher, 7 rounds.

4.2 Training Gohr's model on SPN

The hyperparameters used to achieve the best accuracy were the same for both Heys version and the modified version of the SPN cipher. The parameters used to achieve this is shown in Table 4.3

Parameter	Value
Depth	1
Number of epochs	50
Cyclic learning rate	(10, 0.0002, 0.0001)
Regularization parameter	0.0005
Dropout (after dense layer 1 and 2)	0.10
Dense layer units	32

Table 4.3: Hyperparameters for the best achieved accuracy in both Heys version of the SPN cipher, and the modified version with identical rounds.

The best accuracy achieved for rounds one to six using Heys version of the SPN cipher is shown in both Figure 4.9 and Table 4.4.

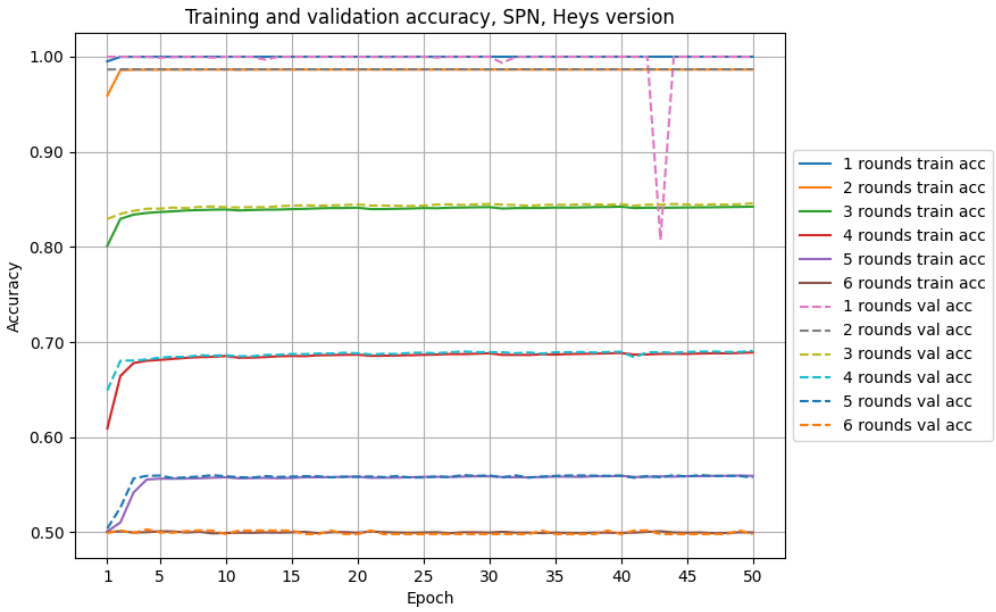

Figure 4.9: Training and validation accuracy for Heys version of SPN.

Rounds	**Training acc**	**Validation acc**
1	0.999	0.999
2	0.987	0.987
3	0.842	0.846
4	0.689	0.691
5	0.559	0.558
6	0.500	0.498

Table 4.4: Final training and validation accuracy for different numbers of rounds, on Heys version of the SPN cipher.

The best accuracy achieved for rounds one to six using the modified version of the SPN cipher with identical rounds is shown in both Figure 4.10 and Table 4.5.

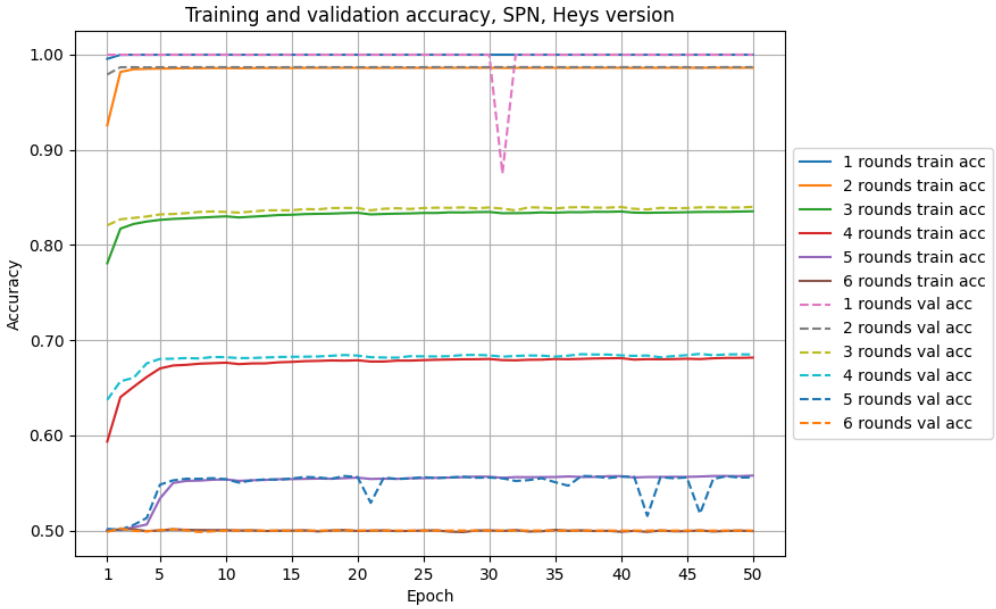

Figure 4.10: Training and validation accuracy for the modified version of SPN with identical rounds.

Rounds	Training acc	Validation acc
1	0.999	0.999
2	0.986	0.987
3	0.835	0.840
4	0.682	0.690
5	0.558	0.5586
6	0.500	0.500

Table 4.5: Final training and validation accuracy for different numbers of rounds, on the modified version of the SPN cipher with identical rounds.

4.3 Key recovery attack on SPN

This section splits the key recovery attack into four different sections depending on what variant of the SPN cipher was used, and if an ML or classical distinguisher were used.

4.3.1 Heys version of SPN with classical distinguisher

Table 4.6 and 4.7 shows the top 10 key guesses when using a classic distinguisher when querying a single ciphertext pair difference at a time, combined with Heys version of the SPN cipher.

Rank	Key	Score
1	0xe1be	58.0000
2	0x60b0	58.0000
3	0x60b6	58.0000
4	0x60b5	58.0000
5	0x60b4	58.0000
6	0x60b3	58.0000
7	0x60b2	58.0000
8	0x60b1	58.0000
9	0xb4be	58.0000
10	0x60b8	58.0000

Table 4.6: 2 rounds of encryption, using a 1 round classical distinguisher. Correct key: 0x5fbe.

Rank	Key	Score
1	0x4ae0	22.0000
2	0xeae0	22.0000
3	0x8ae0	22.0000
4	0x2ae0	22.0000
5	0xdaf6	22.0000
6	0xcad6	21.0000
7	0xcaf6	21.0000
8	0xcae0	21.0000
9	0xcae6	21.0000
10	0x0ae0	21.0000

Table 4.7: 3 rounds of encryption, using a 2 round classical distinguisher. Correct key: 0x3208.

4.3.2 Heys version of SPN with ML distinguisher

Tables 4.8 and 4.9 are showing the top 10 key guesses for keyrecovery attacks on 2 and 3 rounds of encryption respectively.

Rank	Key	Score
1	0x3ae6	-56.198
2	0xdae6	-56.211
3	0x8ae6	-56.276
4	0x4ae6	-56.277
5	0xeae6	-56.333
6	0x2ae6	-56.345
7	0xfae6	-56.348
8	0xaae6	-56.366
9	0x0ae6	-56.370
10	0x5ae6	-56.410

Table 4.8: 2 rounds of encryption, using a 1 round ML distinguisher. Correct key: 0x60e4

Rank	Key	Score
1	0xdc2c	-625.293
2	0xdc7c	-637.134
3	0xda2c	-637.820
4	0xdc0c	-643.418
5	0xdc6c	-648.569
6	0x8c7c	-648.973
7	0xec7c	-650.525
8	0x8c2c	-650.538
9	0x7c2c	-653.130
10	0xdc3c	-653.427

Table 4.9: 3 rounds of encryption, using a 2 round ML distinguisher. Correct key: 0x3492

4.3.3 Modified version of SPN with classical distinguisher

Tables 4.16 to 4.15 shows the top 10 key guesses for each round of key recovery attack, when using the classical distinguisher together with the modified version of SPN.

Rank	Key	Score
1	0x5fbe	100.0000
2	0x5736	85.0000
3	0x57b6	83.0000
4	0x5f3e	82.0000
5	0x57be	80.0000
6	0x5fb6	80.0000
7	0x573e	79.0000
8	0x5bfe	79.0000
9	0x5ebf	78.0000
10	0xdfb6	78.0000

Table 4.10: 2 rounds of encryption, using a 1 round classical distinguisher. Correct key: 0x5fbe.

Rank	Key	Score
1	0x3208	100.0000
2	0x3a00	74.0000
3	0x3288	72.0000
4	0x3a80	72.0000
5	0xb208	71.0000
6	0x3200	70.0000
7	0x3280	70.0000
8	0x3a88	70.0000
9	0x3a08	68.0000
10	0xba00	67.0000

Table 4.11: 3 rounds of encryption, using a 2 round classical distinguisher. Correct key: 0x3208.

Rank	Key	Score
1	0x3faf	76.0000
2	0x3f27	51.0000
3	0x37a7	50.0000
4	0xbfaf	50.0000
5	0x2faf	49.0000
6	0x7faf	48.0000
7	0x1faf	47.0000
8	0x372f	47.0000
9	0x3727	47.0000
10	0x3f2f	46.0000

Table 4.12: 4 rounds of encryption, using a 3 round classical distinguisher. Correct key: 0x3faf.

Rank	Key	Score
1	0x3d77	41.0000
2	0xbd77	39.0000
3	0x0d77	34.0000
4	0x2d77	34.0000
5	0x7d77	33.0000
6	0x1d77	33.0000
7	0x8d77	32.0000
8	0x3977	31.0000
9	0x3d37	31.0000
10	0x9d77	31.0000

Table 4.13: 5 rounds of encryption, using a 4 round classical distinguisher. Correct key: 0x3d77.

Rank	Key	Score
1	0x0488	13.0000
2	0x1d08	12.0000
3	0x2ee0	12.0000
4	0x48ac	12.0000
5	0x4554	11.0000
6	0xddf7	11.0000
7	0x3ca9	11.0000
8	0x15cc	11.0000
9	0xcca6	11.0000
10	0x27cd	11.0000

Table 4.14: 6 rounds of encryption, using a 5 round classical distinguisher. Correct key: 0x0488.

Rank	Key	Score
1	0x3130	3.0000
2	0x043c	3.0000
3	0xfeb8	3.0000
4	0x1130	3.0000
5	0x3112	3.0000
6	0x5656	3.0000
7	0x5476	3.0000
8	0x9a5a	2.0000
9	0x9a58	2.0000
10	0x3872	2.0000

Table 4.15: 7 rounds of encryption, using a 6 round classical distinguisher. Correct key: 0xda5c.

4.3.4 Modified version of SPN with ML distinguisher

Tables 4.16, 4.17, 4.18, and 4.19 are showing the top 10 key guesses for two to five rounds of encryption, using a ML distinguisher trained on one round less. Tables 4.20 and 4.21 are showing the guesses for six rounds of encryption, where the first used 100 samples of ciphertexts in the key recovery attack and the second used 1000 samples.

Rank	Key	Score
1	0x7b0f	692.373
2	0x7b87	554.083
3	0x738f	527.769
4	0x7387	524.300
5	0x7307	496.024
6	0xfb07	490.280
7	0x7b07	482.501
8	0x7b8f	479.893
9	0xfb8f	475.839
10	0x730f	475.607

Table 4.16: 2 rounds of encryption, using a 1 round ML distinguisher. Correct key: 0x7b0f.

Rank	Key	Score
1	0xf00a	492.703
2	0xf082	234.384
3	0xf80a	225.716
4	0x788a	214.024
5	0xf88a	198.335
6	0xf002	186.272
7	0xf882	184.470
8	0x7802	184.222
9	0x780a	183.122
10	0xf08a	173.959

Table 4.17: 3 rounds of encryption, using a 2 round ML distinguisher. Correct key: 0xf00a.

Rank	Key	Score
1	0xd426	263.910
2	0x5426	180.153
3	0xd4ae	155.753
4	0xd42e	145.063
5	0x5cae	125.578
6	0x542e	122.565
7	0x54ae	115.444
8	0xdcae	114.545
9	0xf426	112.777
10	0xdc2e	109.927

Table 4.18: 4 rounds of encryption, using a 3 round ML distinguisher. Correct key: 0xd426.

Rank	Key	Score
1	0xd83f	60.878
2	0x583f	35.623
3	0xf83f	30.022
4	0x983f	27.555
5	0x183f	24.954
6	0x083f	24.239
7	0xc83f	22.018
8	0xd82e	19.587
9	0x283f	18.079
10	0x483f	17.414

Table 4.19: 5 rounds of encryption, using a 4 round ML distinguisher. Correct key: 0xd83f.

Rank	Key	Score
1	0x4179	9.075
2	0x6375	8.695
3	0x0179	8.615
4	0x7370	8.389
5	0xb322	7.881
6	0xaafb	7.732
7	0x5179	7.703
8	0xe374	7.678
9	0xb371	7.571
10	0xe365	7.433

Table 4.20: 6 rounds of encryption, using a 5 round ML distinguisher. Correct key: 0x4068. 100 samples of ciphertexts.

Rank	Key	Score
1	0xffbd	65.569
2	0xffad	24.537
3	0xbfbd	14.283
4	0xfeac	8.344
5	0xefbd	7.286
6	0x1fbd	5.139
7	0xc2d7	3.801
8	0xefad	2.079
9	0xeeac	1.793
10	0x0fbd	0.964

Table 4.21: 6 rounds of encryption, using a 5 round ML distinguisher. Correct key: 0xffbd. 1000 samples of ciphertexts.

4.4 Training Gohr's model on SIMON32/64

The model architecture proposed by Gohr was used to train on SIMON32/64 ciphertexts. The same parameters he used for his pre-trained models were used in this training, to do a comparison on the performance. These parameters are shown in Table 4.22.

Parameter	Value
Depth	1
Number of epochs	50
Regularization parameter	0.0001
Cyclic learning rate	(10, 0.001, 0.0001
Input shape	4×16
Dense layer units	64

Table 4.22: ML model hyperparameters for SIMON32/64.

The model was trained from five to ten rounds of encryption, and the resulting accuracy and validation for each model are shown in Figure 4.11. The exact numbers shows in Table 4.23

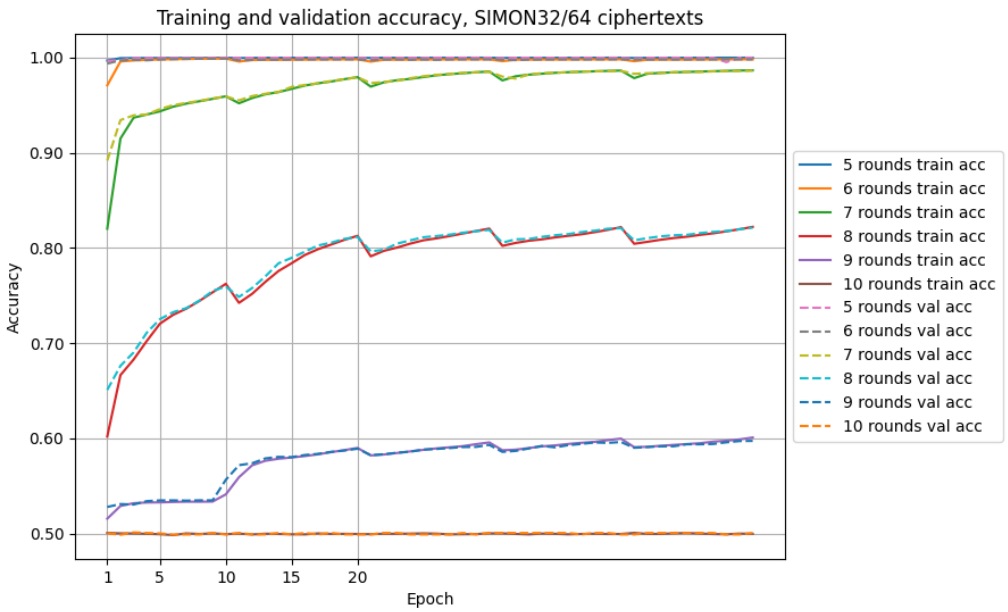

Figure 4.11: Training and validation accuracy for SIMON32/64 ciphertext pairs.

	Training acc	**Validation acc**
5 rounds	0.999	0.999
6 rounds	0.998	0.998
7 rounds	0.987	0.986
8 rounds	0.822	0.821
9 rounds	0.601	0.600
10 rounds	0.500	0.501

Table 4.23: Training and validation accuracy for different numbers of rounds of SIMON32/64 ciphertext pairs.

4.5 Fine-tuning a pre-trained model

Gohr provided pre-trained models from five to eight rounds, and they were all re-trained when unfreezing one, two, and three layers, respectively. The summarized results are showing in Figure 4.12, 4.13, and 4.14. Table 4.24 shows the exact numbers.

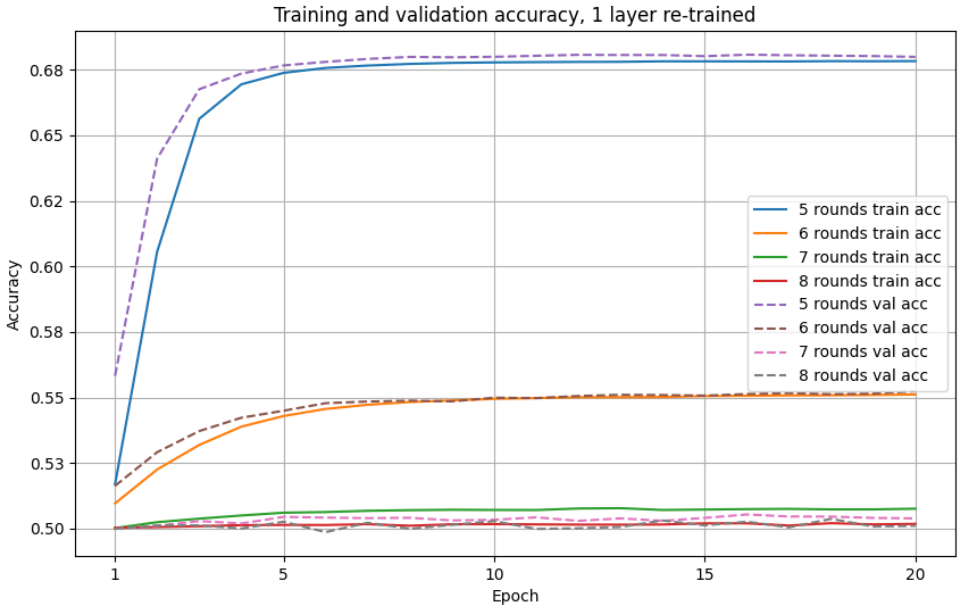

Figure 4.12: Training and validation accuracy for one unfrozen layer.

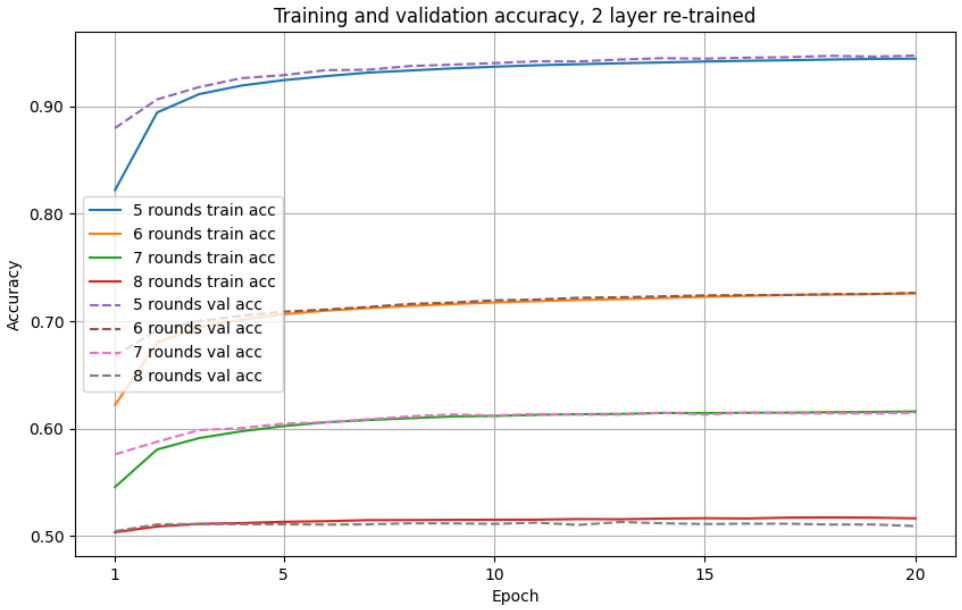

Figure 4.13: Training and validation accuracy for two unfrozen layers.

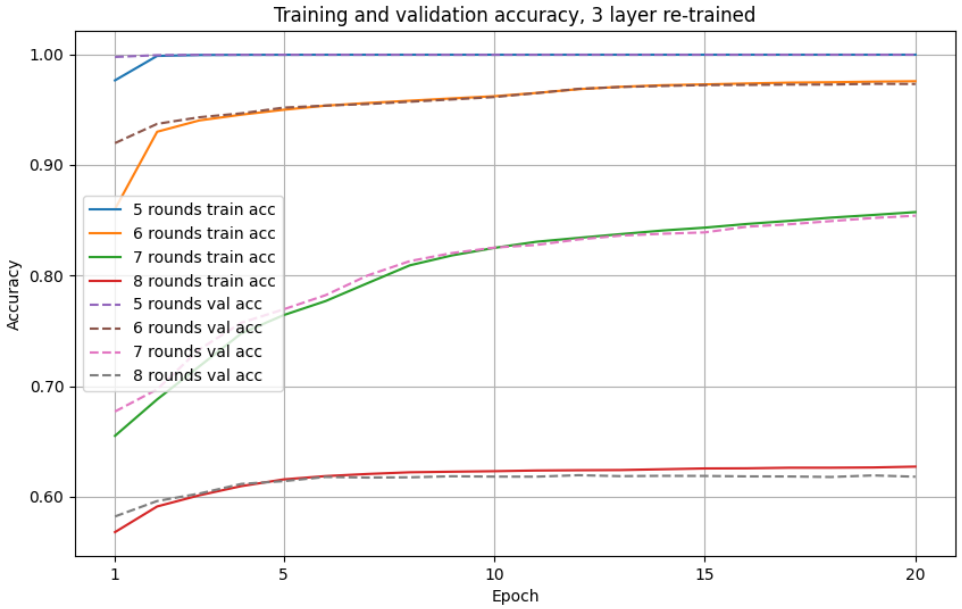

Figure 4.14: Training and validation accuracy for three unfrozen layers.

	5 rounds	**6 rounds**	**7 rounds**	**8 rounds**
1 unfrozen layer	0.678 / 0.680	0.551 / 0.552	0.508 / 0.504	0.502 / 0.501
2 unfrozen layers	0.944 / 0.945	0.726 / 0.726	0.616 / 0.615	0.517 / 0.510
3 unfrozen layers	0.999 / 0.999	0.976 / 0.973	0.857 / 0.854	0.627 / 0.618

Table 4.24: Training / validation accuracy for different numbers of unfrozen layers and cipher rounds.

5

Conclusion and Discussion

This chapters will discuss the results obtained throughout this work and connect them to the research questions. The findings will be discussed and analyzed in particular focus on the comparison between classical and machine learning-assisted cryptanalysis, the scalability of the approaches, and the role of transfer learning.

5.1 SPN

The experiments made on the SPN cipher variants provide a clear comparison between classical and machine-learning distinguishers. The classical χ^2 distinguisher showed strong performance in identifying non-random behavior in the ciphertext pair distributions up to six rounds. In contrast, the neural distinguisher showed good but decreasing accuracy, and reached random classification already at six rounds. This result indicates that for this type of relatively simple cipher, the classical statistical method are more effective at capturing the differential characteristics produced by the cipher. The machine learning model struggles to generalize and classify when the signal becomes weaker at increased rounds of encryption. This shows promise that the classical results remain more robust throughout and performs slightly better.

An important insight from the experiments made on SPN is that high accuracy does not automatically translate into good and effective key recovery. This was particularly shown in Heys version of the SPN cipher where both the classical and the ML-distinguisher were unable to recover the correct key, even at lower rounds. In comparison, the modified version with identical rounds allowed for successful key recovery up to six rounds. This result indicates and highlights the importance of the cipher structure in cryptanalysis. Even a small design difference, such as the last round being different, can affect the success rate quite significantally. It also shows that a distinguisher cannot be evaluated only by itself, but needs to be evaluated in the full pipeline.

To summarize, the results from the SPN cipher experiments shows that the classical methods are slightly outperforming the ML methods, and might still be preferable.

5.2 SIMON32/64

When evaluating the use of a pre-trained network to use in transfer learning, the main limitation was the requirement of the input data. For transfer learning to work, the architecture must be identical, and therefore the input shape of the data must be the same. As a result, the distinguisher can only be applied and trained on data that match this shape, which restricts the general application ability.

To achieve accuracy and performance that are somewhat promising, multiple layers needed to be unfrozen during fine-tuning. In this case, three layers had to be unfrozen out of five layers in total. Although this improved the performance, it reduces the benefit of transfer learning as the majority of the model was re-trained. However, one advantage was the faster convergence. Transfer learning reached its peak accuracy after 8-10 epochs, while the other trained distinguisher reached its peak after 30-40 epochs. Although this efficiency is showing, the final accuracy remains lower for each round in transfer learning compared to the other distinguisher trained from scratch.

When comparing the two approaches, training a model from scratch on the ciphertext data is favorable. More notably, the accuracy achieved in round r is almost identical to the one achieved in round $r-1$ in transfer learning. This indicates that the model trained from scratch generalizes to deeper rounds, which is the advantage one looks for.

5.3 Research questions

How does a hybrid attack pipeline compare to purely classical attacks in both classification and key recovery tasks?

The results show that the hybrid pipeline performs well, but not as well as the classical approach. For classification, the classical distinguisher was more effective on the SPN cipher, with the ability to classify correctly one round more than the ML distinguisher. For key recovery task, both approaches performed similarly. For the modified SPN cipher, both were able to recover the correct key for up to six rounds, while for the Heys variant neither were able to recover the key on any round. This suggests that the cipher structure can have much bigger effect on the outcome rather than the choice of classical vs ML approaches.

How does the hybrid pipeline scale with respect to the number of rounds, i.e if at all, up to how many rounds can it maintain advantage over classical methods?

The scalability of the hybrid pipeline is limited and shows close performance to the classical approach. As the number of rounds increases, both the hybrid and classical approach shows lower performance due to the diminishing strength of the differential signal. For the SPN cipher, both approaches could recover six rounds each, showing that the hybrid approach does not have an advantage over the classical approach. This suggests that a hybrid pipeline approach does not overcome the limitations

and features of the cipher's design, and that the weaknesses of the cipher can be exploited with both approaches almost just as good.

How effective is transfer learning across different ciphers compared to training a network from scratch?

The experiment on transfer learning showed clear limitations. While it reduced the training time needed and allowed for quicker convergence in terms of number of epochs, it showed under-performance in accuracy on each round when compared. In terms of fine-tuning the pre-trained network on SIMON32/64, good performance was showing after unfreezing multiple layers of the pre-trained model. At that point, the majority of the layers were unfrozen which removes most of the benefits of transfer learning. This indicates that patterns learned from one cipher, in this case SPECK32/64, does not generalize well to another, in this case SIMON32/64 even when they have similar structures.

The results suggests that the ML distinguishers learn cipher-specific structures and features that are hard to generalize. Transfer learning shows limited effectiveness in this setting and cannot replace the method of training a model from scratch.

5.4 Improvements and future work

Although the discoveries and results in this thesis indicate that machine learning does not have a clear advantage over classical approaches, the area remains relatively unexplored and offers several directions for future work.

One potential direction is to extend the experiments to a wider range of ciphers. These can be both more lightweight block ciphers, but also more complex ones. This could potentially provide a broader understanding of when and how machine learning-based approaches are most effective, and when classical approaches performs better.

Another direction is to explore other neural model architectures and training strategies. The models used in this work are based on existing designs, but there might be potential improvement in modifying the architecture more, or develop completely new ones.

Another area of interest would be the integration and combination of multiple distinguishers within a single attack pipeline. For example, combining a four round distinguisher with a key recovery attack on the fifth round, followed by another four round distinguisher. Being able to classify a nine round encryption in this way would show a big improvement compared to the results showed in this thesis.

Finally, a cross-cipher learning approach could be explored. By combining different ciphers and their structures, a more generalized distinguisher might be performing well regardless of the cipher it gets evaluated on. For example, by cross-training on the two variants of the SPN cipher might give the advantage needed to successfully perform a key recovery attack on Heys version.

5.5 Contributions to real world use

The findings in this thesis have practical contributions to the security of lightweight ciphers like SIMON, which is particularly used in RFID technology. As mentioned in Chapter 2, the SIMON cipher family is part of the ISO standard for RFID security, which is commonly used in systems for tracking goods, inventory management, and authenticating products. These systems usually operate on strict use of computational power, memory, and energy consumption, which requires the use of lightweight ciphers. However, these constraints may reduce the security margin which in turn makes it important to continuously evaluate the robustness and reliability of the chosen ciphers.

This thesis contributes to that evaluation by analyzing how the development of machine learning techniques affect and perform on round-reduced versions of the same cipher. The results show that while machine learning can detect non-random patterns in certain cases, it does not yet show promise beyond classical methods and approaches. This provides some reassurance regarding the current state and security measures for RFID technology.

From a supply chain perspective, the results indicates that as machine learning evolves and new methods and attacks emerges, there is a need to continue the assessment of the ciphers in place. Machine learning might lower the barrier on performing more advanced cryptanalysis, potentially weakening the security measures put in place throughout the supply chain.

By connecting theoretical cryptanalysis with real-world practical applications in supply chain systems, this thesis contributes to a better understanding of how emerging and developing techniques may impact the future security of RFID-based systems.

Bibliography

[1] B. D. Kim, V. A. Vasudevan, R. G. L. D'Oliveira, A. Cohen, T. Stahlbuhk, and M. Médard, *Cryptanalysis via machine learning based information theoretic metrics*, 2025. arXiv: 2501.15076 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2501.15076.

[2] Z. Wu, K. Qiao, Z. Wang, J. Cheng, and L. Zhu, "Mixture differential cryptanalysis on round-reduced simon32/64 using machine learning," *Mathematics*, vol. 12, no. 9, 2024. DOI: https://doi.org/10.3390/math12091401.

[3] Z. Hou, J. Ren, and S. Chen, "Improved machine learning-aided linear cryptanalysis: Application to des," *Cybersecurity*, vol. 8, no. 22, 2025. DOI: https://doi.org/10.1186/s42400-024-00327-4.

[4] A. Gohr, "Improving attacks on round-reduced speck32/64 using deep learning," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., Cham: Springer International Publishing, 2019, pp. 150–179, ISBN: 978-3-030-26951-7. DOI: https://doi.org/10.1007/978-3-030-26951-7_6.

[5] I. O. for Standardization, "What is cryptography?," [Online]. Available: https://www.iso.org/information-security/what-is-cryptography.

[6] J. Raigoza and K. Jituri, "Evaluating performance of symmetric encryption algorithms," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1378–1379. DOI: 10.1109/CSCI.2016.0258.

[7] H. Cheng and Q. Ding, "Overview of the block cipher," in *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, 2012, pp. 1628–1631. DOI: 10.1109/IMCCC.2012.379.

[8] C. E. Shannon, "Communication theory of secrecy systems - the material in this paper appeared originally in a confidential report "a mathematical theory of cryptography" dated sept. 1, 1945, which has now been declassified.," in *Claude E. Shannon: Collected Papers*. 1993, pp. 84–143. DOI: 10.1109/9780470544242.ch2.

[9] H. M. Heys, "A tutorial on linear and differential cryptanalysis," *Cryptologia*, vol. 26, no. 3, pp. 189–221, 2002. DOI: 10.1080/0161-110291890885.

[10] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, *SIMON and SPECK: Block ciphers for the internet of things*, Cryptology ePrint Archive, Paper 2015/585, 2015. [Online]. Available: https://eprint.iacr.org/2015/585.

[11] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *Journal of Cryptology*, vol. 4, 1991. DOI: https://doi.org/10.1007/BF00630563.

[12] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," 2014. DOI: https://doi.org/10.48550/arXiv.1412.6980. arXiv: 1412.6980.

[13] H. Feistel, "Cryptography and computer privacy," *Scientific American*, vol. 228, no. 5, pp. 15–23, 1973, ISSN: 00368733, 19467087. Accessed: Mar. 27, 2026. [Online]. Available: http://www.jstor.org/stable/24923044.

[14] C. McCoy, *Simon & speck block ciphers in python 3.x/2.x*, 2015-2016. [Online]. Available: https://github.com/inmcm/Simon_Speck_Ciphers/tree/master/Python/simonspeckciphers.

A

Appendix 1

Binary value of the extraordinary p1's generated			
Only 1 bit active			
0000 0000 0000 0001	0000 0000 0000 0010	0000 0000 0000 0100	0000 0000 0000 1000
0000 0000 0001 0000	0000 0000 0010 0000	0000 0000 0100 0000	0000 0000 1000 0000
0000 0001 0000 0000	0000 0010 0000 0000	0000 0100 0000 0000	0000 1000 0000 0000
0001 0000 0000 0000	0010 0000 0000 0000	0100 0000 0000 0000	1000 0000 0000 0000
Every nibble the same			
0000 0000 0000 0000	0001 0001 0001 0001	0010 0010 0010 0010	0011 0011 0011 0011
0100 0100 0100 0100	0101 0101 0101 0101	0110 0110 0110 0110	0111 0111 0111 0111
1000 1000 1000 1000	1001 1001 1001 1001	1010 1010 1010 1010	1011 1011 1011 1011
1100 1100 1100 1100	1101 1101 1101 1101	1110 1110 1110 1110	1111 1111 1111 1111
One nibble all ones			
0000 0000 0000 1111	0000 0000 1111 0000	0000 1111 0000 0000	1111 0000 0000 0000
Two nibbles all ones			
1111 1111 0000 0000	0000 0000 1111 1111	1111 0000 1111 0000	0000 1111 0000 1111
1111 0000 0000 1111	0000 1111 1111 0000		
One nibble all zeros			
1111 1111 1111 0000	1111 1111 0000 1111	1111 0000 1111 1111	0000 1111 1111 1111
All except one bit active			
1111 1111 1111 1110	1111 1111 1111 1101	1111 1111 1111 1011	1111 1111 1111 0111
1111 1111 1110 1111	1111 1111 1101 1111	1111 1111 1011 1111	1111 1111 0111 1111
1111 1110 1111 1111	1111 1101 1111 1111	1111 1011 1111 1111	1111 0111 1111 1111
1110 1111 1111 1111	1101 1111 1111 1111	1011 1111 1111 1111	0111 1111 1111 1111

Table A.1: A list of extraordinary pairs used for plaintext generation.