CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Applications of machine learning to cryptanalysis, and increased privacy preservation through homomorphic encryption
Authors	Ebrahimi Moghaddam, Amirhossein
Publication date	2026-06-01
Original Citation	Ebrahimi Moghaddam, A. 2026. Applications of machine learning to cryptanalysis, and increased privacy preservation through homomorphic encryption. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2026, Amirhossein Ebrahimi Moghaddam. - https://creativecommons.org/licenses/by-nc-nd/4.0/
Download date	2026-11-04 21:11:19
Item downloaded from	https://hdl.handle.net/10468/19280



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Applications of machine learning to cryptanalysis, and increased privacy preservation through homomorphic encryption

Amirhossein Ebrahimi Moghaddam

MSc
119222673

Thesis submitted for the degree of Doctor of Philosophy

University College Cork
College of Science, Engineering and Food Science
School of Computer Science & Information Technology
ADVANCE CRT

2026

Head of School: Professor Utz Roedig
Supervisor: Dr. Paolo Palmieri
Co-supervisor: Prof. Utz Roedig

Contents

List of Figures . . . iv
List of Tables . . . v
Acknowledgements . . . viii
Abstract . . . ix

1 Introduction **1**
1.1 General Overview . . . 1
1.2 Motivation and Background . . . 2
1.2.1 Choice of Primitives and Use Cases . . . 3
1.2.2 Research Questions . . . 4
1.3 Contributions . . . 5
1.4 Thesis Structure . . . 6

2 Preliminaries and State of the Art **7**
2.1 Introduction to Cryptography . . . 7
2.1.1 Symmetric Cryptography . . . 7
2.1.1.1 Block ciphers . . . 8
2.2 Introduction to Cryptanalysis . . . 9
2.2.1 Principles of Cryptanalysis . . . 9
2.2.2 Differential Cryptanalysis . . . 10
2.2.3 Rotational-XOR Cryptanalysis . . . 12
2.3 Machine Learning in Cryptanalysis . . . 17
2.3.1 Overview of Machine Learning . . . 18
2.3.1.1 Fully Connected Layers (Dense) . . . 20
2.3.1.2 Convolutional Neural Networks (CNNs) . . . 21
2.3.1.3 Recurrent Neural Network (RNN) Layers . . . 22
2.3.1.4 Transformers and the Attention Mechanism . . . 23
2.3.2 Early Work on Machine Learning in Cryptanalysis . . . 25
2.3.3 Machine-Learning-Assisted Differential Cryptanalysis . . . 28
2.3.4 Recent Advances and Current Trends . . . 31
2.3.5 Deep-Learning-Based Differential Distinguishers . . . 32
2.3.6 Evolutionary Algorithms for Parameter Search . . . 35
2.4 Homomorphic Encryption and Privacy in Machine Learning . . . 36
2.4.1 Introduction to Homomorphic Encryption (HE) . . . 36
2.4.1.1 Early Concepts and Breakthroughs . . . 36
2.4.2 Cheon-Kim-Kim-Song (CKKS) Scheme . . . 39
2.4.2.1 Encoding and Decoding in CKKS . . . 40
2.4.2.2 Encryption and Decryption in CKKS Scheme . . . 41
2.4.2.3 Relinearization in CKKS . . . 42
2.4.2.4 The Rescaling Process in CKKS . . . 44
2.4.3 Privacy-Preserving ML Models . . . 44
2.4.3.1 Privacy-Preserving for Natural Language Processing (NLP) . . . 45
2.4.4 Recent Advances and Future Directions . . . 46
2.4.4.1 HE based Techniques . . . 46

2.4.4.2 Hybrid Approaches 49
2.4.5 Challenges and Limitations 50

3 Reducing the Time Complexity of Machine-Learning-Assisted Differential Attacks via Bit Selection 53
3.1 Introduction 54
3.1.0.1 Contribution 55
3.1.1 Outline 55
3.2 Preliminaries 56
3.2.1 The SPECK cipher 56
3.2.2 Key Averaging Algorithm 56
3.3 Partial-Differential Machine-Learning Distinguisher (PD-ML) . . 57
3.3.1 Methodology 58
3.3.2 Results and Discussion 62
3.4 Measuring Bit Effectiveness for ML-Based Distinguishers 63
3.4.1 Methodology 63
3.4.2 Results and Discussion 65
3.4.3 Reducing the Time Complexity of *Key Averaging* 71
3.5 Conclusion 72

4 Deep Learning-Based Rotational-XOR Distinguishers for AND-RX Block Ciphers 75
4.1 Introduction 76
4.1.1 Our Contribution 77
4.1.2 Outline 78
4.2 Preliminaries 78
4.2.1 AND-RX Ciphers 79
4.2.2 Weak Key Model Attacks 81
4.3 Identification of Optimal RX Distinguishers in Cryptanalysis with Evolutionary Algorithm 81
4.3.1 AI Tools and Model Development 82
4.3.2 Training DL-based RX distinguishers 84
4.3.2.1 Training phase 84
4.3.2.2 Machine Interpretation 85
4.3.3 Evolutionary Optimisation of Deep Learning RX Differential Distinguishers 87
4.4 Results and Discussion 90
4.4.1 Simeck Cipher 91
4.4.2 Simon Cipher 93
4.5 Impact of the Diffusion Layer and Optimal Rotation Parameters 95
4.6 Conclusion 97

5 Privacy-Preserving Sentiment Analysis Using Homomorphic Encryption and Attention Mechanisms 99
5.1 Introduction 100
5.1.1 Motivation and Contribution 100
5.1.2 Outline 102

5.2 Efficient Activation Functions 102
5.3 Methodology 103
5.3.1 Proposed Method 103
5.3.1.1 Model Architecture 104
5.3.2 Training and Inference 107
5.3.2.1 Training 107
5.3.2.2 Inference Phase 108
5.3.2.3 Dataset Description 109
5.4 Implementation 110
5.4.1 Hardware and Software Setup 110
5.4.1.1 Training 110
5.4.1.2 Inference 111
5.4.2 Impact of Padding Number on the Implementation . . . 112
5.4.3 Rationale for Selecting a Linear Activation Function . . . 113
5.4.4 Justification for Server-Side Computation Despite Simplified Activation Function 114
5.5 Results 115
5.6 Conclusion 118

6 Conclusions and Future Work **120**
6.1 Overview and Summary of Contributions 120
6.2 Limitations 122
6.3 Future Work 123

List of Figures

2.1 The process of symmetric encryption and decryption. 8
2.2 Overview of Fully Connected Layer 20
2.3 Overview of ResNet Layer 22
2.4 Overview of RNN Layer 24
2.5 High Level View of CKKS 40

3.1 Accuracy of ML-based distinguishers according to number of bits in LSB→MSB direction for 6 rounds of SPECK32/64 61
3.2 Accuracy of ML-based distinguishers according to number of bits in MSB→LSB direction for 6 rounds of SPECK32/64 61
3.3 Normalized Scores for 4 different combination sets. The bits with vertical lines are the ones that are chosen as the most effective bits for training a PD-ML-based distinguisher for six rounds of SPECK32/64. 66
3.4 The Training Process of PD-ML distinguishers. Rank values are as per Table 3.1. Plotted bars are approximation of the values (due to graphical constraints). 68

4.1 Simon cipher for $m = 4$ 80
4.2 Simeck cipher 80
4.3 Scatter plot of Bias Score vs Accuracy of RX Distinguisher (Colored by Rounds) 87

5.1 Simplified Schematic Diagram of the Client-Server Privacy-Preserving Neural Network 105
5.2 High-level architecture of the privacy-preserving machine learning system, delineating the client and server components. . . . 107
5.3 Distribution of comment lengths in the dataset 113

List of Tables

3.1 The average score (s_i) and standard deviation (SD_i) of each bit of δ in the experiments. 67
3.2 Comparison of different machine learning based differential distinguishers. The training time refers to the networks running on the Google Colab platform [Col] 70
3.3 Evaluation of the performance of ML-based distinguishers in executing the key averaging algorithm. 72

4.1 Comparison of related-key DL-based distinguishers for Simeck. RX: Rotational-XOR cryptanalysis, RD: Related-key Differential cryptanalysis. The Combined Accuracy Score [GLN22] for m pairs is $\frac{1}{1+\prod_{i=1}^{m}\frac{1-p_i}{p_i}}$ 77
4.2 Comparison of the RX distinguishers for different versions of Simeck 77
4.3 Summary of the optimal RX input differences (δ), key differences (δ_{key}), rotational offsets (γ), the number of rounds, and distinguisher accuracy for different versions of Simeck block ciphers, where *optimal* means best found under the evolutionary-search objective used in this chapter. 92
4.4 Summary of the optimal RX input differences (δ), key differences (δ_k), rotational offsets (γ), the number of rounds, and distinguisher accuracy for different versions of Simon block ciphers, where *optimal* means best found under the evolutionary-search objective used in this chapter. 94
4.5 Optimal rotation sets for AND-RX ciphers with non-linear key schedule and $n = 32$, where *optimal* means best found under the evolutionary-search objective used in this chapter 96

5.1 Comparison of our work with previous approaches 116

Declaration

This is to certify that the work I am submitting is my own and has not been submitted for another degree, either at University College Cork or elsewhere. All external references and sources are clearly acknowledged and identified within the contents. I have read and understood the regulations of University College Cork concerning plagiarism and intellectual property.

Digital signature of the candidate: Amirhossein Ebrahimi Moghaddam

To my family, for their unwavering support and encouragement.
To my friends, for their understanding and patience during this long journey.
To my supervisors, for their guidance and mentorship.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Paolo Palmieri and Prof. Utz Roedig, for their guidance and support throughout this work.

Special thanks to my family, especially my wife, for her unwavering support and belief in me.

I am grateful to my colleagues and collaborators, Francesco Regazzoni, David Gerault, and Buvana Ganesh, for their encouragement and insightful discussions.

This research was supported by Science Foundation Ireland under Grant No. 18/CRT/6222.

Amirhossein Ebrahimi Moghaddam
Cork,

Abstract

This thesis examines how machine learning and cryptography interact in practical security settings where computation takes place between parties with different objectives. Its central theme is the pursuit of practical usefulness under computational constraint. In cryptanalysis, machine learning is used to analyse and exploit statistical structure in symmetric block ciphers. In privacy-preserving inference, cryptographic techniques are used to ensure that machine-learning computations can be performed without revealing sensitive user data. These two directions represent complementary roles of the machine learning–cryptography interface: machine learning can be used to evaluate and stress-test cryptographic constructions, while cryptography can be used to enable secure deployment of machine-learning systems.

The first part of the thesis focuses on symmetric cryptography and uses machine learning as a tool for cryptanalysis and security evaluation. It studies machine-learning-assisted differential distinguishers with particular emphasis on efficiency as a condition for practical cryptanalytic use. A Partial Differential machine-learning distinguisher is introduced, trained on a selected subset of output-difference bits rather than the full block state, together with an empirical method for estimating bit effectiveness. Evaluated on reduced-round SPECK32/64, the results show that compact feature sets can retain useful distinguishing capability while reducing computational cost, demonstrating that a small loss in predictive performance can be acceptable when the distinguisher remains effective and substantially cheaper to train and apply. The thesis then extends the same machine-learning-assisted perspective to rotational-XOR (RX) analysis of AND-RX block ciphers. Focusing on the Simon and Simeck families, it develops and evaluates deep-learning-based RX distinguishers alongside RX-characteristic analysis, providing evidence on how design choices such as rotation parameters and key schedules influence RX-difference propagation and distinguishability. In this setting, efficiency matters because the distinguisher acts not only as a predictor but also as a practical search and evaluation tool over a large parameter space.

The second part of the thesis addresses the complementary direction, where cryptography supports privacy-preserving machine learning. It proposes a sentiment-inference pipeline based on the public-key CKKS homomorphic encryption scheme, using a client–server split with a client-side attention mechanism and an encrypted prediction component designed to operate under

ciphertext-level constraints. Evaluated on the IMDb dataset, the method demonstrates a practical trade-off between predictive performance and the overhead introduced by encrypted computation. Here, efficiency is not a secondary optimisation target but a deployment requirement, since encrypted inference is only useful when its computational cost remains manageable.

Chapter 1

Introduction

1.1 General Overview

Cryptography underpins confidentiality, integrity, and authenticity in modern digital systems. In practice, security mechanisms combine different classes of cryptographic primitives. Symmetric primitives (for example, block ciphers) provide efficient bulk encryption and are widely deployed in constrained environments, while public-key primitives enable functionality such as key establishment and digital signatures.

Cryptanalysis studies the security of cryptographic constructions under explicit attack models. Typical goals include distinguishing a cipher from an ideal primitive, identifying statistical weaknesses, or quantifying a security margin against a class of attacks. Classical techniques such as differential cryptanalysis and linear cryptanalysis remain central tools in this area [BS91, Mat93]. As cryptographic designs and deployment settings have evolved, cryptanalysis has increasingly combined mathematical reasoning with automation and large-scale computation.

Machine learning (ML) has become relevant to this landscape in two complementary ways. First, it can serve as a data-driven tool in cryptanalysis by learning statistical structure from samples and producing effective distinguishers, including neural distinguishers for reduced-round block ciphers [Goh19, BGPT21, BBDH21, GHHP24, SSL+24]. Second, the widespread deployment of ML systems has intensified privacy concerns, particularly when inference is outsourced to an untrusted or semi-trusted environment such as a cloud platform. Addressing such concerns motivates privacy-preserving inference methods, including homomorphic encryption [Gen09, CKKS17, WWL+25, ZYH+25].

1.2 Motivation and Background

The research in this thesis is motivated by practical constraints that arise in both cryptanalysis and privacy-preserving computation.

A recurring theme across the thesis is efficiency, but efficiency is not treated here as an abstract optimisation target. Instead, it is relevant because each contribution operates under a concrete practical constraint. In machine-learning-assisted cryptanalysis, a method that is too expensive to train or too costly to integrate into downstream attacks has limited practical value even if its raw accuracy is strong. In RX analysis, the cost of evaluating distinguishers and searching parameter spaces affects how broadly cipher variants, key classes, and rotation settings can be explored. In homomorphic-encryption-based inference, efficiency is even more fundamental: multiplicative depth, ciphertext growth, latency, and memory usage can determine whether deployment is feasible at all. For this reason, the thesis does not argue that lower cost is always preferable by itself. Rather, it studies when a reduction in cost is justified because the remaining level of accuracy or distinguishing advantage is still sufficient for the underlying task.

On the symmetric-cryptanalysis side, machine-learning-assisted distinguishers can be effective, but their cost can be prohibitive when training data, compute, or time is limited. This is especially relevant for lightweight symmetric primitives that are deployed in resource-constrained settings and must be evaluated using realistic budgets, including candidates studied in standardisation efforts [BÇMT18]. Reducing the computational burden of ML-assisted differential distinguishers is therefore important for making these techniques usable as part of practical security evaluation rather than only as proof-of-concept attacks.

In this setting, the relevant trade-off is between raw classifier accuracy and cryptanalytic practicality. A distinguisher does not need to maximise benchmark accuracy if it still preserves enough non-random signal to support procedures such as key averaging or key ranking at substantially lower cost. The central question is therefore not whether some accuracy is lost, but whether the loss is small enough that the distinguisher remains useful while the reduction in computational burden makes the overall attack pipeline more realistic.

Rotational-XOR (RX) cryptanalysis provides a complementary perspective for analysing AND-RX designs such as Simon and Simeck [BSS+15, YZS+15]. RX analysis has a substantial prior literature [KN10, AL16], but parameter choices

and modelling assumptions can strongly influence results in practice. Extending deep-learning-based distinguishers to RX settings supports a more systematic evaluation of these ciphers and can provide additional insight into how design choices (for example, rotation constants and key schedules) affect RX-difference propagation.

Here, efficiency matters in a different way. The objective is not only to train an accurate distinguisher, but also to use that distinguisher as part of a broader search and evaluation framework over RX parameters and cipher variants. Even a moderate distinguishing advantage can be valuable if it enables wider exploration of the design space, more systematic comparison across settings, or evaluation beyond highly restricted weak-key cases. The trade-off of interest is therefore between distinguisher strength and the cost of obtaining enough evidence to support meaningful cryptanalytic or design-level conclusions.

On the privacy-preserving side, many ML applications process sensitive inputs. When inference is outsourced, the service provider may be able to learn private information from client data or intermediate representations. Homomorphic encryption addresses this problem by enabling computation on encrypted data. In particular, CKKS is a public-key (asymmetric) homomorphic encryption scheme that supports approximate arithmetic and is widely used for encrypted inference [CKKS17]. However, encrypted inference is expensive and requires architectural choices that control multiplicative depth, ciphertext growth, and latency. Designing usable privacy-preserving pipelines under these constraints is therefore an important engineering and security problem [PT20, PT21].

In this context, the efficiency–accuracy trade-off is not optional: it is a deployment constraint. A more expressive model may achieve higher predictive performance, but if encrypted evaluation becomes too deep, too slow, or too memory-intensive, then the model may be impractical for real use. Accordingly, the aim in this part of the thesis is not to maximise plaintext-style benchmark performance, but to identify a point at which predictive quality remains acceptable while encrypted inference becomes materially more feasible.

1.2.1 Choice of Primitives and Use Cases

The specific primitives and datasets studied in this thesis were selected to align with the practical focus of the contributions.

For Chapter 3, we evaluate on SPECK32/64 because it is a widely used benchmark in the literature on neural differential distinguishers and ML-assisted

differential cryptanalysis [Goh19, BGPT21, BBDH21, GHHP24]. SPECK is a lightweight ARX block cipher and is publicly specified alongside Simon [BSS+15], making it a standard testbed for comparing distinguishers under controlled settings.

For Chapter 4, we focus on rotational-XOR (RX) analysis for Simon and Simeck because RX is a natural difference model for rotation-based lightweight designs and has become an active line of research for Simon-like primitives. Many ARX and AND-RX constructions use fixed rotations to achieve diffusion; these rotations can also induce rotational structure that is not well captured by standard XOR-differential analysis alone. RX cryptanalysis targets such structure by analysing how an input pair related by a rotation and XOR mask propagates through the cipher, and it has been shown to yield distinguishers and related-key insights for Simon-like designs [KN10, LLA+22, CZX+23]. Recent work has also explored deep-learning-based RX distinguishers and optimisation of RX parameters, which motivates the methodology developed in this thesis [EGP23, LCX+25]. Simon and Simeck are widely studied AND-RX families with simple round functions and well-documented design choices, making them suitable testbeds for evaluating RX distinguishers and for analysing how rotation parameters and key schedules influence RX-difference propagation [BSS+15, YZS+15].

For Chapter 5, we consider privacy-preserving sentiment inference under homomorphic encryption. We use CKKS because it supports approximate arithmetic suited to neural-network inference and has become a common choice in recent encrypted-inference pipelines [CKKS17, WWL+25, ZYH+25, FAL25]. Sentiment analysis provides a concrete inference task with clear evaluation metrics, and the IMDb dataset is a standard benchmark in this setting [MDP+11].

1.2.2 Research Questions

Thesis-level question. How can the interaction between machine learning and cryptography be leveraged to support practical security and privacy goals under realistic computational and deployment constraints, and how should efficiency be balanced against accuracy or distinguishing strength in each setting?

This thesis addresses the thesis-level question through three related subquestions:

1. **(Chapter 3).** How can the computational cost of ML-assisted differential distinguishers be reduced while retaining useful distinguishing capability

in practice?

2. **(Chapter 4).** How can deep-learning-based distinguishers be adapted to RX analysis of AND-RX block ciphers (such as Simon and Simeck), and what does this reveal about design choices that influence RX-difference propagation?
3. **(Chapter 5).** What architectural and cryptographic design choices enable homomorphic-encryption-based sentiment inference with acceptable predictive performance under HE constraints (for example, multiplicative depth and ciphertext-level computation costs)?

1.3 Contributions

The main contributions of this thesis are summarised as follows:

- **Cost reduction for ML-assisted differential cryptanalysis.** We introduce a Partial Differential machine-learning distinguisher trained on a selected subset of output-difference bits and propose an empirical scoring method to estimate bit effectiveness. The approach is evaluated on reduced-round SPECK32/64 with the aim of reducing training and downstream attack cost while preserving enough distinguishing capability for practical cryptanalytic use.
- **Deep-learning-based RX analysis of AND-RX ciphers.** We study deep-learning-based distinguishers for RX cryptanalysis and RX-characteristic analysis for the Simon and Simeck families. The results include RX characteristics and distinguishers for multiple variants together with analysis of factors that influence RX propagation, with emphasis on making RX evaluation practical across parameter settings rather than treating classifier accuracy as the only objective.
- **Privacy-preserving sentiment inference using public-key homomorphic encryption.** We propose a privacy-preserving sentiment-analysis pipeline based on the CKKS homomorphic encryption scheme. The approach combines a client–server split with an attention mechanism and an encrypted prediction component designed to operate under ciphertext-level constraints, prioritising a usable balance between predictive performance and encrypted-inference cost.

1.4 Thesis Structure

Chapter 2 reviews background and related work needed for the remainder of the thesis, including differential and RX cryptanalysis, machine-learning-based distinguishers, and homomorphic encryption with emphasis on CKKS.

Chapters 3 and 4 focus on machine learning as a tool for cryptanalysis and security evaluation of symmetric primitives. Chapter 3 develops methods to reduce the cost of ML-assisted differential distinguishers and evaluates them on SPECK32/64. Chapter 4 investigates deep-learning-based RX distinguishers and RX-characteristic analysis for Simon-like AND-RX designs.

Chapter 5 focuses on cryptography as an enabler for privacy-preserving machine-learning inference. It presents a CKKS-based sentiment-inference pipeline and evaluates design choices required for computation over ciphertexts.

Chapter 6 concludes the thesis, summarises limitations, and outlines directions for future work.

Across Chapters 3–5, the unifying principle is therefore not efficiency in the abstract, but practical usefulness under constraint: each chapter studies a setting in which reduced cost is valuable only when the remaining level of accuracy, distinguishing strength, or predictive performance is still sufficient for the intended purpose.

Chapter 2

Preliminaries and State of the Art

This chapter reviews the background required for the contributions in Chapters 3–5. It focuses on (i) symmetric cryptography and the threat models relevant to differential and rotational-XOR cryptanalysis, (ii) how machine learning is used to construct statistical distinguishers and to support cryptanalytic workflows, and (iii) homomorphic encryption—with emphasis on the CKKS scheme—as a foundation for privacy-preserving inference.

The aim is not to be encyclopedic. Instead, the chapter introduces the concepts and notation used later in the thesis and summarises the prior work most directly related to the three contributions.

2.1 Introduction to Cryptography

Cryptography provides mechanisms to protect confidentiality, integrity, and authenticity in digital systems. This thesis primarily studies symmetric primitives because the attacks developed in later chapters target round-reduced block ciphers and related constructions. Public-key cryptography (used in practice for key exchange and signatures) is not required for the contributions in Chapters 3–5 and is therefore omitted here beyond this brief context.

2.1.1 Symmetric Cryptography

Symmetric (or secret-key) cryptography uses the same key for encryption and decryption. Because its operations are relatively lightweight, it is typically used for bulk data encryption once parties have established a shared key. In this thesis, symmetric cryptography is central: Chapters 3 and 4 study differential

and rotational-XOR cryptanalysis against lightweight block ciphers. Figure 2.1 illustrates the high-level process.

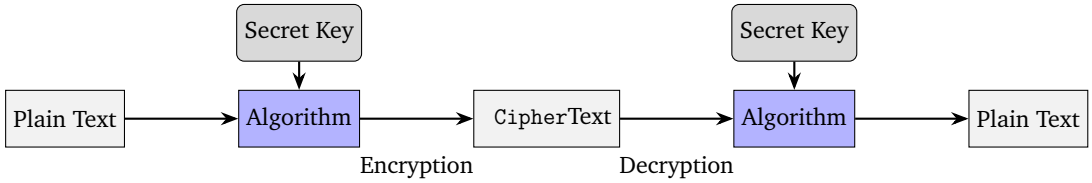


Figure 2.1: The process of symmetric encryption and decryption.

2.1.1.1 Block ciphers

Block ciphers operate as iterative algorithms, where a non-linear function is applied repeatedly. Each iteration, known as a round, uses a different subkey derived from the main key. The design of block ciphers must carefully balance the number of rounds to achieve an optimal trade-off between security and computational efficiency. A sufficient number of rounds is crucial to thwart cryptographic attacks, while too many rounds can reduce performance unnecessarily.

These algorithms encrypt data in fixed-length groups of bits, called blocks, using a specific secret key. Each block is processed through the algorithm independently. Some well-known block ciphers include AES (Advanced Encryption Standard) [DR99], DES (Data Encryption Standard) [S+99], and IDEA (International Data Encryption Algorithm) [Bas11]. AES is the most widely used symmetric key algorithm and is considered the industry standard, often used in government and financial institutions for secure data encryption.

The need for secure communication in devices with constrained resources has led to the development of lightweight cryptography. These cryptographic algorithms are designed to be efficient in terms of power, memory, and computational requirements, making them ideal for use in small-scale or embedded systems such as RFID tags, sensors, smart cards, and IoT devices.

The importance of lightweight cryptography stems from the vast and growing number of devices that require security but do not have the capability to run traditional cryptographic algorithms due to their size, power, or computational limitations. These devices often operate within a constrained environment where power consumption and processing capabilities are at a premium. As such, lightweight cryptographic algorithms provide a balance between security and performance, ensuring data protection without overtaxing the limited

resources of these devices. The main focus of Chapter 3 and Chapter 4 are on these ciphers. Some examples of these ciphers are: PRESENT [BKL+07], SIMON and SPECK [BSS+15], and CLEFIA [SSA+07].

Other symmetric primitives such as stream ciphers, message authentication codes, and authenticated encryption are widely deployed, but they are not used in the contributions of this thesis. They are therefore omitted from the remainder of this chapter to keep the preliminaries focused.

2.2 Introduction to Cryptanalysis

Cryptanalysis studies the security of cryptographic constructions under explicit threat models. Typical goals include distinguishing a real cipher from an ideal primitive, recovering secret key material, or quantifying a cipher's security margin against specific classes of attacks. This chapter focuses on differential cryptanalysis and rotational-XOR (RX) cryptanalysis because they are directly relevant to the methods developed in Chapters 3 and 4.

2.2.1 Principles of Cryptanalysis

Understanding the foundational concepts of symmetric cryptography is essential for understanding different cryptanalysis techniques. Symmetric cryptography, at its core, involves several key elements:

Plaintext (P): The original, unencrypted data that is to be secured. It represents the input to the encryption algorithm in a cryptographic system.

Ciphertext (C): The result of encrypting plaintext using a cryptographic algorithm. It is the scrambled, unreadable output, denoted as $C = E_k(P)$, where E_k represents the encryption process under a key k.

Keys (K): In symmetric cryptography, the key is a shared secret used for both encryption and decryption. The security of the cryptographic system hinges on the secrecy and strength of this key.

Encryption (E): The process of converting plaintext into ciphertext using a cryptographic algorithm and a key, expressed as $C = E_k(P)$.

Decryption (D): The reverse process of encryption, converting ciphertext back into plaintext using the same key, expressed as $P = D_k(C)$.

Chosen Plaintext Attack (CPA): An attack model where the cryptanalyst has the capability to choose arbitrary plaintexts and obtain the corresponding

ciphertexts. The goal is to uncover key information or weaknesses in the encryption algorithm.

Chosen Ciphertext Attack (CCA): In this attack model, the cryptanalyst can choose ciphertexts and obtain their decrypted plaintexts, aiming to exploit vulnerabilities in the decryption process.

Related Key Attack (RKA): This attack model involves the analysis of the relationships between keys. In a related key attack, the cryptanalyst observes the operation of a cryptographic algorithm under several different keys where these keys are related in a known way. The cryptanalyst's goal is to uncover the secret key or exploit these key relationships to breach the cryptographic system's security.

Single Key Attack: Unlike a related key attack, a single key attack involves analyzing the cryptographic system's behavior under one specific key. The objective is to exploit weaknesses in the algorithm's structure or implementation when the same key is used repeatedly, often without knowledge of key relationships used in related key attacks.

Based on these concepts, the general goal of cryptanalysis in symmetric cryptography is that for a given cryptographic algorithm, represented by a function f, and a ciphertext $C = f(P, k)$, the objective is to deduce some information, or leakage (L), from the plaintext $L(P)$ and/or the key $L(k)$. This is often approached probabilistically, assessing the likelihood of different P and k pairs producing the known C, i.e., finding P and k such that the probability of $f(P, k) = C$ is maximized.

It is important to note the probabilistic nature of cryptanalysis. It means that cryptanalysis often adopts a probabilistic approach, reflecting the complexity and robustness of modern cryptographic algorithms. This approach involves analyzing the probabilities of various outcomes under different scenarios, rather than relying on deterministic methods.

While there are numerous other cryptanalytic attacks, such as linear [Mat93, Bih95], integral [KW02], and division attacks [Tod15], this thesis specifically focuses on two techniques: Differential Cryptanalysis and Rotational-XOR (RX) cryptanalysis.

2.2.2 Differential Cryptanalysis

Differential cryptanalysis [BS91] is a sophisticated technique employed in cryptanalysis to uncover potential vulnerabilities in block ciphers. Classified as a cho-

sen plaintext attack, it allows the attacker to select arbitrary plaintexts, encrypt them, and analyze the resulting ciphertexts to deduce information about the encryption key.

The cornerstone of differential cryptanalysis lies in the analysis of plaintext pairs, denoted as P_1 and P_2, and their corresponding ciphertexts C_1 and C_2. The primary goal is to discover specific and predictable differences between these pairs. The input difference is defined as

$$\Delta P = P_1 \oplus P_2,$$

and the corresponding output difference is

$$\Delta C = C_1 \oplus C_2.$$

In these expressions, $\oplus$ signifies the XOR (exclusive OR) operation.

For a cipher exhibiting random behavior, the probability of observing a specific output difference ΔC for a given input difference ΔP should ideally be $\frac{1}{2^n}$, where n is the size of P in bits. However, certain ciphers demonstrate non-random behavior, where specific output differences occur more frequently than $\frac{1}{2^n}$. Differential cryptanalysis seeks to exploit these statistical anomalies by identifying pairs of differences, $(\Delta P, \Delta C)$, that occur with higher than expected probabilities. Such pairs are referred to as differentials.

The methodology of differential cryptanalysis includes several key steps:

1. Choosing a collection of plaintext pairs that conform to a predetermined input difference ΔP.
2. Encrypting these pairs and observing the output differences in the resulting ciphertexts.
3. Utilizing these observed output differences to infer details about the encryption key, potentially leading to its recovery.

Differential cryptanalysis is particularly potent against block ciphers where the input-output difference relationship can be predicted with greater accuracy than mere chance. Through meticulous analysis of these relationships, insights into the key's structure or the key itself can be derived.

The effectiveness of a differential attack is quantified by the probability p_D, which indicates the likelihood that a chosen input difference ΔP in plaintexts will result in a specific output difference ΔC in ciphertexts. This probability can

be expressed as:

$$p_D^r = \Pr((\Delta = P_1 \oplus P_2) \rightarrow (\Delta^r = C_1^r \oplus C_2^r))$$

where P_1, P_2 represent the plaintext pairs, and C_1^r, C_2^r are the resulting ciphertexts after r round encrypting P_1, P_2.

To recover the key in differential cryptanalysis, assuming we have a differential characteristic for $r-1$ rounds, p_D^{r-1} one follows these steps:

1. Select pairs of plaintexts P_1, P_2 that exhibit the desired input difference ΔP.
2. Encrypt these pairs to obtain the ciphertexts C_1^r, C_2^r.
3. For each hypothetical value of the final round subkey K^r, partially decrypt C_1^r and C_2^r using K^r and calculate the difference after this partial decryption:
$$\Delta^{r-1} = C_1^{r-1} \oplus C_2^{r-1}$$
4. Assess this difference against the expected differential characteristic. Track the frequency of each subkey hypothesis leading to the expected difference.
5. The subkey hypothesis that most consistently results in the expected difference is likely to be the correct subkey for the final round.

This structured approach allows cryptanalysts to methodically probe and potentially unravel the key structure of block ciphers, showcasing the depth and utility of differential cryptanalysis in the field of cryptography. Algorithm 1 shows the differential cryptanalysis steps

Differential cryptanalysis has paved the way for various specialized cryptanalytic techniques, such as Impossible Differential [BBS99], Truncated Differential [Knu95], and Higher-Order Differential Cryptanalysis [Lai94], each extending the foundational principles to exploit different aspects of cryptographic vulnerabilities. These adaptations highlight the technique's robustness and its critical role in advancing the field of cryptanalysis.

2.2.3 Rotational-XOR Cryptanalysis

Rotational cryptanalysis is a technique used to analyze the security of symmetric algorithms. Khovratovich and Nikolić introduced and formalized this approach for ARX structures, the schemes that only have modular addition, AND opera-

Algorithm 1 Differential Attack Algorithm

Input: Collection of plaintext pairs (P_1, P_2) with known input difference ΔP, encryption function E
Output: Estimated key K^r for the final round
Select pairs of plaintexts P_1, P_2 that exhibit the desired input difference ΔP.
Encrypt these pairs to obtain the ciphertexts C_1^r, C_2^r.
Initialize a frequency count array for each hypothetical value of the final round subkey K^r.
for each pair of ciphertexts (C_1, C_2) **do**
    for each hypothetical value of the final round subkey K^r **do**
        Partially decrypt C_1^r and C_2^r using K^r to obtain C_1^{r-1} and C_2^{r-1}.
        Calculate the difference after this partial decryption: $\Delta^{r-1} = C_1^{r-1} \oplus C_2^{r-1}$.
        if Δ^{r-1} matches the expected differential characteristic **then**
            Increment the frequency count for K^r.
        end if
    end for
end for
$K^r \leftarrow$ the hypothesis with the highest frequency count.
return K^r

tion and XOR operation in their algorithm, in their work cited as [KN10], and subsequently applied it to scrutinize other ciphers like Skein [KNR14]. In this technique, the attacker focuses on rotational pairs of plaintext and ciphertext, where the input values are related through a fixed rotation. The attacker then looks for statistical biases or patterns in the ciphertexts of these pairs that can be exploited to recover the secret key.

However, rotational cryptanalysis can be less effective in the presence of constants, as these fixed values can disrupt the rotational properties of the pairs. This is because the rotation operation alone does not account for the XOR operations that involve these constants, which can be present in many cryptographic algorithms. When constants are involved, the rotational relations between the input and output values might be obscured, making it harder to analyze the cipher using traditional rotational cryptanalysis techniques.

To address the limitations of traditional rotational cryptanalysis, Rotational-XOR cryptanalysis has been introduced as an extension. Ashur and Liu [AL16] have developed this technique to account for the XOR operations with constants that are commonly present in cryptographic algorithms. Rotational-XOR pairs and Rotational-XOR differences are defined to provide a more comprehensive framework for analyzing cryptographic primitives that involve both rotation

and XOR operations with constants.

Before exploring the specifics of Rotational-XOR (RX) cryptanalysis in greater depth, it is necessary to first define several key terms and concepts. The following are definitions for RX pairs, RX difference, and RX cryptanalysis, respectively

Definition 2.2.1 (Rotational XOR Pair [AL16]). *An RX-pair is a rotational pair with rotational offset γ under translations δ_1 and δ_2, defined as the pair $P_1 \oplus \delta_1, (P_1 \lll \gamma) \oplus \delta_2$. However, for the sake of simplicity, a slightly different notation is used, where an RX-pair is represented by P_1 and $P_2 = (P_1 \lll \gamma) \oplus \delta$, or alternatively as $P_1, (P_1 \lll \gamma) \oplus \delta$, where $\delta = \delta_1 \oplus \delta_2$.*

Definition 2.2.2 (Rotational XOR Difference [AL16]). *An RX-difference of P_1 and P_2, denoted by $\Delta_\gamma(P_1, P_2)$, is formed by the rotational XOR of P_1 with a constant δ such that $P_2 = (P_1 \lll \gamma) \oplus \delta$, where $0 < \gamma < n$ and $\delta \in F_n^2$ is a constant. In another word*

$$\Delta_\gamma(P_1, P_2) = P_2 \oplus (P_1 \lll \gamma)$$

Definition 2.2.3 (Rotational XOR Cryptanalysis [AL16]). *Rotational XOR Cryptanalysis is a cryptanalytic technique that extends traditional rotational cryptanalysis to handle XOR operations with non-rotational-invariant constants between input and output pairs of a cryptographic primitive. This method aims to estimate the transition probability with respect to non-linear operations in block ciphers (like modular addition or $\wedge$ operation) of two input RX-differences to an output RX-difference. The technique introduces the concept of a (δ, γ)-Rotational-XOR-difference (or RX-difference), which represents a rotational pair with rotation γ under translation δ, i.e., $(P, (P \lll \gamma) \oplus \delta)$. The method seeks to analyze the propagation of RX-differences through the cryptographic primitive.*

The transmission of RX-differences through linear operations is known to be deterministic; however, this is not the case for nonlinear operations. Prior research conducted by Ashur et al. [AL16] and Lu et al. [LLA+22] investigated the transmission of RX-differences through modular addition and AND ($\wedge$) operations, respectively.

Computing the probability of p_{RX}, where:

$$p_{RX} = \Pr(\Delta_\gamma(C_1, C_2) = C_2 \oplus (C_1 \lll \gamma) \,|\, \Delta_\gamma(P_1, P_2) = P_2 \oplus (P_1 \lll \gamma)),$$

is not as straightforward as computing p_D. The following theorem helps to estimate the propagation of RX pairs through non-linear elements of block ciphers.

In cryptographic systems utilizing ARX (Addition-Rotation-XOR) designs, assessing the security against rotational-XOR attacks necessitates understanding the propagation of RX differences through modular addition ($\boxplus$). The theoretical framework for this assessment can be quantified by a formula calculating the probability that an RX pair maintains its rotational relationship post-modular addition. The following formula is for $\gamma = 1$ which is proved in [AL16]. In this theorem, the hamming weight of x is denoted by $|x|$, and $|$ is the bitwise OR. A left shift by 1 is denoted by SHL, and $(I \oplus \text{SHL})(x) = x \oplus \text{SHL}(x)$. Lastly, $1_{x \preceq y}$ is the characteristic function which evaluates to 1 when $x_i \leq y_i$ for all i and 0 for other conditions.

Theorem 2.2.4. *Suppose that x, y are n-bits operands, and $z = x \oplus y$. Let $\Gamma_x = \Delta_1(x, x'), \Gamma_y = \Delta_1(y, y')$ and $\Gamma_z = \Delta_1(z, z')$ be constants, which are the RX-differences. Then,*

$$\begin{aligned}
&\Pr\Big(((x \oplus \Gamma_x) \lll 1) \boxplus ((y \oplus \Gamma_y) \lll 1) = (z \oplus \Gamma_z) \lll 1\Big) \\
&\quad = 1_{(I \oplus SHL(\delta_x \oplus \delta_y \oplus \delta_z)) \oplus 1 \preceq (SHL((\delta_x \oplus \delta_z) | (\delta_y \oplus \delta_z))} \cdot 2^{-(|SHL((\delta_x \oplus \delta_z) | (\delta_y \oplus \delta_z))|} \cdot 2^{-3} \\
&\quad + 1_{(I \oplus SHL(\delta_x \oplus \delta_y \oplus \delta_z)) \preceq (SHL((\delta_x \oplus \delta_z) | (\delta_y \oplus \delta_z))} \cdot 2^{-(|SHL((\delta_x \oplus \delta_z) | (\delta_y \oplus \delta_z))|} \cdot 2^{-1.415}
\end{aligned}$$

where

$$\delta_x = L(\Gamma_x), \quad \delta_y = L(\Gamma_y), \quad \delta_z = L(\Gamma_z).$$

and $L(x)$ is the most significant bit of x.

On the other hand, the propagation of RX differences through the AND operation in cryptographic algorithms can be formally analyzed through a mathematical theorem, providing a closed algebraic formula for its expected probability. Specifically, this addresses how an RX difference with translation δ_1 passes through the vectorial AND operation, which is pivotal for AND-RX ciphers analysis.

Theorem 2.2.5 (Propagation of RX Differential through AND)**.** *Given a function $f(x, y) = S_a(x) \wedge S_b(y)$, where $S_a(x) = x \lll a$ and $S_b(y) = y \lll b$ represent the cyclic left rotation of a binary string x by a and b bits respectively, and $\wedge$ denotes the bitwise AND operation, let $\Delta_\gamma(x, y) = y \oplus (x \lll \gamma)$ define the RX difference*

between two binary strings x and y with a rotational offset γ. The probability $p_{RX}(\alpha \to \beta)$ that an RX difference $\alpha = \Delta_\gamma(x, y)_1$ at the input of f leads to an RX difference $\beta = \Delta_\gamma(x, y)_2$ at the output of f can be expressed as:

$$p_{RX}(\alpha \to \beta) = \begin{cases} 2^{-n+1} & \text{if } \alpha = 1^n \text{ and } |\beta| \text{ is even,} \\ 2^{-w} & \text{if } \alpha \neq 1^n \text{ and } \beta \wedge (\overline{S_a(\alpha) \vee S_b(\alpha)}) = 0 \quad \text{and} \\ & (\beta \oplus S_{a-b}(\beta)) \wedge (\overline{S_a(\alpha)} \wedge S_{2a-b}(\alpha) \wedge S_b(\alpha)) = 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $w = |\overline{(S_a(\alpha) \vee S_b(\alpha)} \oplus (\overline{S_a(\alpha)} \wedge S_{2a-b}(\alpha) \wedge S_b(\alpha))|$ denotes the Hamming weight of the expression within the parentheses, and n is the block size in bits. The conditions for β are derived from the bitwise operations on the RX differences affected by S_a and S_b.

It is important to note that, the RX cryptanalysis is a related-key attack, which means we need two different keys to encrypt P_1 and P_2. The strategic use of different keys is crucial for exploiting the vulnerabilities of ARX (Addition, Rotation, XOR) and And-RX (logical AND, Rotation, XOR) ciphers because by defining a specific condition on keys, we can exploit the symmetry structure that exists in these structures. The mathematical relationship between the original key K and a related key K' is foundational to this analysis, expressed as:

$$K' = (K \lll \gamma) \oplus \delta \tag{2.1}$$

where δ is a constant. This relationship is central to probing the cipher's behavior under various key scenarios.

The process of key recovery in RX cryptanalysis involves the following mathematical steps:

1. Choose pairs of plaintexts (P_1, P_2) such that $P_2 = (P_1 \lll \gamma) \oplus \delta$, where γ signifies the bit rotation.
2. Encrypt the chosen plaintext pairs under the related keys r rounds to obtain ciphertext pairs (C_1, C_2), where:

$$C_1^r = E_K(P_1)$$
$$C_2^r = E_{K'}(P_2)$$

3. For each candidate keys (K_i, K'_j), partially decrypt C_1 and C_2 by one round:

$$D_{K_i}(C_1^r) = C_1^{r-1}$$
$$D_{K'_j}(C_2^r) = C_2^{r-1}$$

   And then compute $\Delta_\gamma = (C_1^{r-1} \lll \gamma) \oplus C_2^{r-1}$ for each pair

4. Determine the correctness of each key hypothesis based on its ability to account for the observed encryption characteristics under related key conditions. The correct key is identified through:

This structured approach, leveraging the rotational relationship between keys and mathematical rigor in the analysis, enhances the understanding and effectiveness of RX cryptanalysis in identifying vulnerabilities within block cipher designs. The Algorithm 2 represents the RX key recovery.

Algorithm 2 Rotational XOR Key Recovery

Input: Collection of plaintext pairs (P_1, P_2) with known relation $P_2 = (P_1 \lll \gamma) \oplus \delta$, encryption function E, bit rotation γ, and difference δ
Output: Estimated keys K, K' for the encryption rounds
Choose plaintext pairs (P_1, P_2) following the input relation
Encrypt P_1 and P_2 under related keys to obtain ciphertext pairs (C_1, C_2)
Initialize a frequency count array for each key hypothesis (K_i, K'_j)
for each pair of ciphertexts (C_1, C_2) **do**
    for each candidate keys (K_i, K'_j) **do**
        Partially decrypt C_1 and C_2 by one round to get (C_1^{r-1}, C_2^{r-1})
        Compute $\Delta_\gamma = (C_1^{r-1} \lll \gamma) \oplus C_2^{r-1}$
        if Δ_γ matches the expected characteristic **then**
            Increment the frequency count for (K_i, K'_j)
        end if
    end for
end for
$(K, K') \leftarrow$ key hypotheses with the highest frequency count
return (K, K')

2.3 Machine Learning in Cryptanalysis

Machine learning (ML) provides data-driven methods for modelling statistical structure and making predictions. In cryptanalysis, ML is typically used to build distinguishers, to rank key hypotheses, or to approximate intermediate properties that are hard to model analytically. Since Gohr's 2019 work on

SPECK32/64, ML-based differential distinguishers have become an active line of research and have inspired many variations and improvements.

This section reviews the ML concepts needed later in the thesis, summarises early and recent work on ML-assisted cryptanalysis, and highlights the role of differential ML attacks in evaluating symmetric ciphers.

2.3.1 Overview of Machine Learning

Machine Learning (ML) represents a pivotal area within artificial intelligence (AI) that focuses on enabling algorithms to learn from and make predictions or decisions based on data. At its core, ML encompasses a variety of techniques and models, ranging from simple linear regression to complex ensemble methods, all designed to understand patterns and make informed decisions. The essence of ML lies in its ability to adapt to new data independently, improving its analysis or predictions over time without being explicitly programmed to do so.

From a mathematical standpoint, ML can be viewed through the lens of optimization and function approximation. At its core, an ML model seeks to approximate the true but unknown function f^* that maps an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. This approximation is denoted by $\hat{f}$ and is constructed based on observed data. The goal of machine learning is to minimize the discrepancy between the predicted outputs $\hat{f}(X)$ and the actual outputs Y across the distribution of possible inputs, typically measured by a loss function $\mathcal{L}$:

$$\min_{\hat{f} \in \mathcal{F}} \mathbb{E}_{(X,Y)\sim\mathcal{D}}[\mathcal{L}(Y, \hat{f}(X))]$$

where $\mathcal{F}$ is the family of functions that the learning algorithm can choose from, and $\mathcal{D}$ represents the data distribution. The expectation $\mathbb{E}_{(X,Y)\sim\mathcal{D}}$ mathematically formalizes the average performance of the model $\hat{f}$ across the entire data distribution $\mathcal{D}$, emphasizing that the objective of ML is to perform well on average, rather than just on the observed data.

In the context of ML, there are two primary learning paradigms: supervised learning and unsupervised learning.

Supervised Learning involves training an ML model on a labeled dataset, where each example is a pair consisting of an input vector and the corresponding target output. In supervised learning, the function approximation $\hat{f}$ directly maps inputs X to outputs Y, and the model is trained to minimize the

loss function $\mathcal{L}(Y, \hat{f}(X))$, which quantifies the difference between the actual and predicted outputs. Some examples of supervised learning are: Regression models [DS98, CG16], Support Vector Machines (SVMs) [CV95], decision trees [Loh11], and random forest [Bre01].

Unsupervised Learning, on the other hand, deals with learning patterns from untagged data. The goal is to model the underlying structure or distribution in the data in order to learn more about the data itself. Here, $\hat{f}$ might represent a function that identifies clusters within the data or models the probability density of inputs X. Some examples are: k-means Clustering [M$^+$67], Density-Based Clustering (DBSCAN) [EKS$^+$96], Principal Component Analysis (PCA) [Pea01], and Isolation Forest [LTZ08].

Deep learning (DL), a specialized subset of machine learning, distinguishes itself through the use of deep neural networks that mimic the human brain's structure and function [Ros58]. Unlike traditional machine learning algorithms that linearly map inputs to outputs, deep learning models utilize multiple layers of nonlinear processing units for feature extraction and transformation. Each layer of a deep neural network builds upon the previous one to refine and abstract the input data, enabling the model to learn complex patterns in large datasets. Mathematically, deep learning models can be represented as a composition of functions:

$$\hat{f}(x) = f_n(\dots f_2(f_1(x; \theta_1); \theta_2) \dots; \theta_n)$$

where x is the input, $\hat{f}(x)$ is the model's prediction, $f_1, \dots, f_n$ are the functions corresponding to each layer of the network, and $\theta_1, \dots, \theta_n$ are the parameters of these functions that the training process optimizes. This hierarchical representation of functions allows deep learning models to perform feature learning automatically—transforming raw data into a more abstract and composite representation—which is a significant departure from traditional machine learning models that require manual feature engineering. Deep learning's capability to handle unstructured data such as images, text, and audio has revolutionized fields such as computer vision, natural language processing, and audio recognition, making it a cornerstone of modern AI applications. Common DL layer types include fully connected (dense) layers [Ros58], convolutional layers (CNNs) [LBBH98], and recurrent layers (RNNs) [RHW86].

2.3.1.1 Fully Connected Layers (Dense)

A fully connected (FC) layer, often positioned towards the end of a deep neural network, plays a critical role in integrating learned features for final predictions or classifications. Mathematically, an FC layer performs a linear transformation followed by a nonlinear activation function on its inputs. If we denote the input to the FC layer as $\mathbf{x}$, which is a flattened vector of features from previous layers, the output $\mathbf{y}$ of the FC layer can be expressed as:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Here, $\mathbf{W}$ represents the weight matrix, $\mathbf{b}$ is the bias vector, and ϕ denotes the nonlinear activation function, such as the ReLU (Rectified Linear Unit) or sigmoid function. The operation $\mathbf{W}\mathbf{x}$ embodies the linear transformation, mapping the input vector $\mathbf{x}$ to a new space, and $\mathbf{b}$ allows for an adjustable offset. The activation function ϕ introduces non-linearity, enabling the network to learn complex patterns beyond linear separability.

The essence of the FC layer is to synthesize the features extracted by preceding layers into a form that is suitable for making the final prediction. By adjusting the weights $\mathbf{W}$ and biases $\mathbf{b}$ during the training process, the network learns to weigh the importance of different features for the task at hand, whether it be classification, regression, or another predictive modeling task. This mechanism allows deep neural networks to perform highly sophisticated tasks by learning hierarchical representations of the input data. Figure 2.2 represents a high-level view of Dense layers.

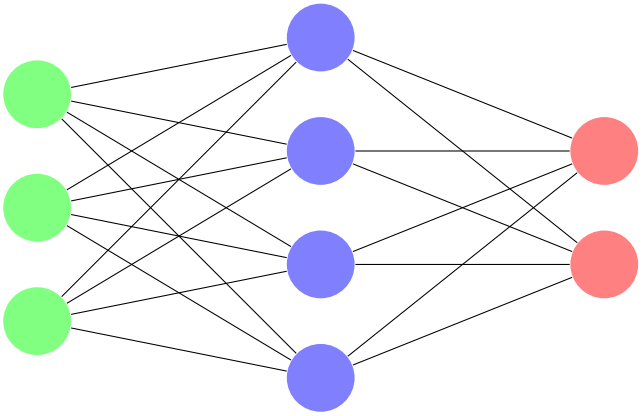


Figure 2.2: Overview of Fully Connected Layer

2.3.1.2 Convolutional Neural Networks (CNNs)

Convolutional layers form the backbone of Convolutional Neural Networks (CNNs), designed to automatically and adaptively learn spatial hierarchies of features from input images. The key operation in a convolutional layer is the convolution operation, which applies a filter or kernel to the input data to produce a feature map. Mathematically, this operation for a single filter can be expressed as:

$$\mathbf{S}(i,j) = (\mathbf{K} * \mathbf{X})(i,j) = \sum_m \sum_n \mathbf{X}(i-m, j-n) \cdot \mathbf{K}(m,n)$$

where $\mathbf{X}$ is the input image or feature map from the previous layer, $\mathbf{K}$ is the kernel or filter, $\mathbf{S}$ is the resulting feature map, and $*$ denotes the convolution operation. The indices i, j run over the spatial dimensions of the input space, while m, n iterate over the dimensions of the kernel, capturing the local spatial information by computing the dot product between the kernel and local regions of the input.

Each convolutional layer may apply multiple filters to the input, producing a set of feature maps that represent different aspects of the input, such as edges, textures, or more complex patterns in deeper layers. Following the convolution operation, a nonlinear activation function, typically the Rectified Linear Unit (ReLU) [NH10], is applied element-wise to introduce non-linearity, allowing for the learning of complex features:

$$\mathbf{A}(i,j) = \phi(\mathbf{S}(i,j))$$

where $\mathbf{A}$ is the activated feature map and ϕ represents the activation function.

Convolutional layers, through the convolution operation and subsequent activation, enable CNNs to build increasingly abstract representations of the input data, making them exceptionally powerful for tasks such as image recognition, segmentation, and classification. The learnable parameters in the filters adapt through training, allowing the network to focus on the most informative features for the task.

Despite their success, a significant challenge emerges as these networks grow deeper: training becomes increasingly difficult due to issues like the vanishing gradient problem, where gradients of the loss function become smaller as they are propagated back through the network, making it hard to update the weights of earlier layers effectively.

To mitigate these challenges and harness the potential of deeper networks, the concept of Residual Networks (ResNets) was introduced [HZRS16]. ResNets, illustrated by Figure 2.3, are built on the principle that instead of layers learning direct mappings from input to output, they should learn the residual mappings, essentially the difference between the actual output and the input. This approach is facilitated by introducing shortcut connections that skip one or more layers, directly adding the input of a block to its output before applying the non-linear activation function.

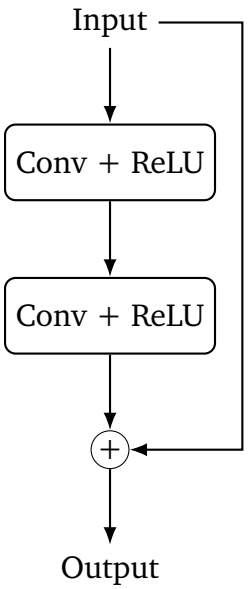


Figure 2.3: Overview of ResNet Layer

By adopting this architecture, ResNets have achieved remarkable performance across a variety of challenging tasks, demonstrating that deeper networks, when equipped with mechanisms to facilitate training, can provide significant improvements over shallower architectures. ResNets' innovation marks a pivotal advancement in deep learning, showcasing the importance of architectural choices in overcoming the practical challenges of training deep neural networks.

2.3.1.3 Recurrent Neural Network (RNN) Layers

Natural Language Processing (NLP) utilizes another class of deep learning models known as Recurrent Neural Networks (RNNs). Recurrent Neural Networks (RNNs) are designed to handle sequential data, making them ideal for tasks such as natural language processing, time series analysis, and any domain where data points are temporally dependent. The core concept of an RNN is its

ability to maintain a 'memory' of previous inputs by incorporating a loop within its network structure, allowing information to persist.

The operation of an RNN layer at a single time step t can be described by the following equations:

$$h_t = \phi(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

where:

- x_t is the input vector at time step t,
- h_t is the hidden state at time step t, representing the 'memory' of the network up to t,
- y_t is the output vector at time step t,
- W_{hh}, W_{xh}, and W_{hy} are the weight matrices for the hidden-to-hidden, input-to-hidden, and hidden-to-output connections, respectively,
- b_h and b_y are the bias vectors for the hidden layer and output, respectively,
- ϕ is a nonlinear activation function, typically sigmoid or tanh.

The hidden state h_t serves as a compact representation of all past information seen by the network up to time t, and it's updated at each step based on the current input x_t and the previous hidden state h_{t-1}. This recurrence relation enables RNNs to theoretically remember long-term dependencies, although, in practice, they are often challenged by issues such as vanishing and exploding gradients.

To address these issues, advanced variations of RNNs, such as Long Short-Term Memory (LSTM) [HS97] and Gated Recurrent Unit (GRU) [CVMG+14] networks, introduce gates and memory cells to better control the flow of information and preserve long-term dependencies more effectively.

RNNs exemplify the power of deep learning in modelling sequential data, leveraging their unique architecture to process inputs of variable lengths and maintain a form of stateful memory across time steps. Figure 2.4 shows a high-level overview of RNN layers.

2.3.1.4 Transformers and the Attention Mechanism

Transformers, introduced in the seminal paper by Vaswani et al. [VSP+17], represent a architecture in deep learning that primarily relies on the attention

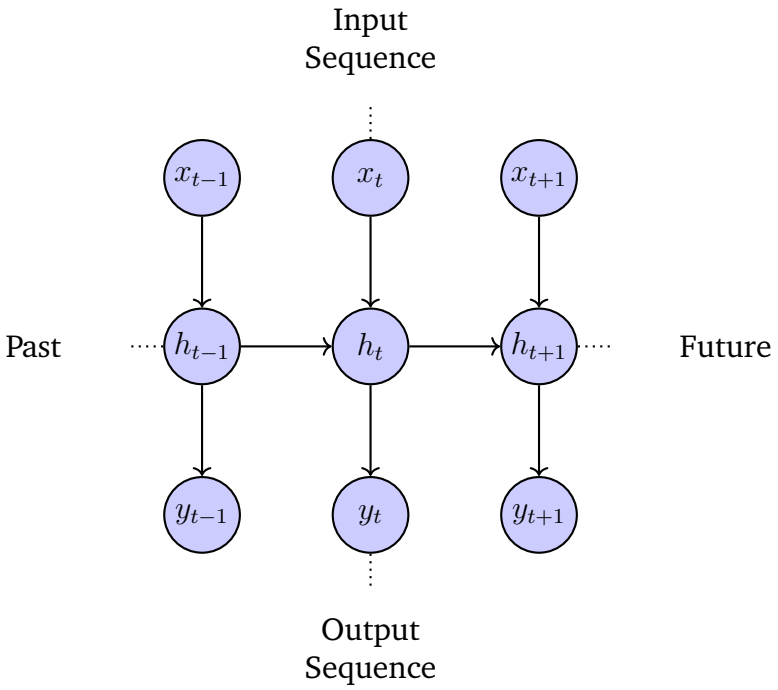


Figure 2.4: Overview of RNN Layer

mechanism, moving away from the recurrent layers of traditional sequence models. The core idea behind the attention mechanism is to weigh the significance of different parts of the input data differently, enabling the model to focus on relevant parts of the input when producing an output.

The mathematical formulation of the attention mechanism in transformers can be described as follows. Given an input sequence, the transformer model computes three sets of vectors for each input: queries (Q), keys (K), and values (V), through linear transformations of the input embeddings. The attention weights are then calculated using the softmax of the dot product of queries and keys, divided by the square root of the dimension of the keys to stabilize the gradients, as shown in the equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the keys.

This attention output is a weighted sum of the values, where the weight assigned to each value is determined by the compatibility of the corresponding key with the query. The softmax function ensures that the weights sum up to 1, allowing the model to distribute its focus over the input sequence. The transformer uses multi-head attention, which means this process is performed

in parallel with different linear transformations of Q, K, and V, allowing the model to capture different aspects of the input data in each "head" of attention.

Transformers leverage this attention mechanism throughout their architecture, both in the encoder, which processes the input sequence, and the decoder, which generates the output sequence. This allows transformers to model complex dependencies between input and output sequences without relying on the sequential processing inherent to RNNs, leading to significant improvements in tasks like translation, text summarization, and even areas beyond NLP, such as image processing.

By focusing on relevant parts of the input data and processing input sequences in parallel, transformers achieve remarkable efficiency and performance, setting new state-of-the-art benchmarks across a wide range of applications.

2.3.2 Early Work on Machine Learning in Cryptanalysis

In the beginning, combining machine learning (ML) with cryptanalysis was a brand-new idea. Researchers were curious whether the same ML tools that find patterns in large datasets could also identify hidden patterns in encrypted messages, a task that is time-consuming and complex when addressed through conventional cryptanalysis alone. These first steps were crucial because they showed that ML could offer a fresh perspective on cryptography and open up new possibilities. In the following paragraphs, we review several important works on this subject.

The seminal work by Rivest, in 1991, explores the complicated relationship between cryptography and machine learning (ML), highlighting the mutual contributions of these fields to each other's advancement [Riv91]. The paper explains how cryptographic problems have pushed advancements in ML, especially in understanding how learning algorithms work when faced with cryptographic challenges. On the other hand, ML has provided new insights and methods for cryptography, particularly in cryptanalysis, where the ability of learning algorithms to detect hidden patterns has been useful. Rivest also discusses the potential for further collaboration between these fields, suggesting that both can continue to benefit from sharing ideas. This study not only highlights the historical link between cryptography and ML but also points to the potential of this interdisciplinary approach for solving future problems in security and learning algorithms.

The exploration of evolutionary computing techniques, specifically Particle

Swarm Optimization (PSO) and Differential Evolution (DE), in the context of cryptanalysis, reveals an enhancement in decrypting the Data Encryption Standard (DES) when reduced to four and six rounds. Laskari et al. study [LMSV07] signifies an improvement in search efficiency of recovering additional subkey bits after differential attack, as evidenced by a marked reduction in the search tree size compared to traditional brute force approaches. In scenarios involving DES reduced to four rounds, the methodologies employed succeeded in identifying the missing key bits with a high accuracy rate, ranging from 93% to 100%, and requiring fewer computational resources. The average success rate exceeded 99%, underscoring the potential of these evolutionary algorithms to optimize the cryptanalytic process effectively. Furthermore, the application to six-round reduced DES also demonstrated the utility of these methods, albeit with varied effectiveness contingent upon the objective function's construction.

Klimov, Mityagin, and Shamir investigate the integration of machine learning with cryptanalysis, focusing on a novel key exchange protocol that utilizes mutual learning among neural networks, diverging from the traditional number theory-based cryptographic methods towards more resource-efficient solutions [KMS02]. The study investigates how two parties can synchronize to a common key, highlighting the difficulties for an attacker attempting to replicate this process. The paper then exposes the protocol's susceptibilities, introducing three potential breaches: a genetic attack leveraging evolutionary principles to refine neural networks that mimic the key exchange, a geometric attack exploiting the network's structural relationships to extract pivotal information, and a probabilistic attack employing statistical predictions to undermine the key exchange's security. Despite these security concerns, the research highlights the use of chaotic synchronization in neural networks for cryptography, urging further exploration of this concept. Although the analyzed protocol has vulnerabilities, the underlying idea inspires the creation of new, secure cryptographic methods.

Other studies have explored machine learning's (ML) ability to differentiate between encrypted messages and random noise. Chou [CLC12] scrutinizes the effectiveness of ML techniques across different encryption modes and datasets, revealing that the encryption mode significantly affects the success of ML applications. The paper highlights the difficulties encountered by ML methods when analyzing data encrypted using Cipher Block Chaining (CBC) mode with random initial vectors, as opposed to the more promising results obtained with Electronic Codebook (ECB) mode under specific conditions. These findings un-

derscore the current limitations of ML in breaking encrypted codes and call for enhanced ML algorithms that can adapt to the complexities of modern encryption techniques. This research prompts a deeper examination of how ML tools for cryptanalysis can be improved for the evolving landscape of cryptographic security.

Gomez et al. introduce CipherGAN, leveraging a CycleGAN-inspired architecture for unsupervised cryptanalysis of substitution ciphers, notably shift and Vigenère ciphers [GHZ+18]. It showcases CipherGAN's ability to decrypt text with high accuracy, extending to larger vocabularies than prior attempts. The innovation lies in adapting CycleGAN for discrete data, ensuring stable training and addressing the challenge of uninformative discrimination commonly encountered in discrete GAN applications. The paper [Gre17] explores the application of Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) units, to decrypt polyalphabetic ciphers, including the Enigma cipher. The authors demonstrate that RNNs can learn the decryption functions from plaintext to ciphertext mappings, effectively capturing the algorithmic essence of these ciphers without being explicitly programmed to do so. This research is not aimed at breaking these ciphers but rather showcases the ability of RNNs to learn and represent complex algorithmic processes. The paper further investigates the internal representations learned by the RNN, showing that the model's memory usage and neuron activations provide insights into the cipher's structure. The approach taken is a sequence-to-sequence translation task, where the model is trained with examples of ciphertext and the corresponding plaintext. The results indicate that RNNs can serve as a tool for cryptanalysis, advancing our understanding of how machine learning models can interpret and replicate algorithmic sequences.

These early explorations into the realm of machine learning for cryptography, and more specifically in cryptanalysis, paved the way for a seminal contribution by Aron Gohr in 2019 [Goh19]. His paper not only showcased the potential of ML to outperform classical cryptanalytic attacks but also established a foundational approach that has significantly influenced subsequent research in the field. This milestone underscores the transformative impact of ML on cryptanalysis, serving as a critical juncture that will be explored in depth in the following section.

2.3.3 Machine-Learning-Assisted Differential Cryptanalysis

The paper [Goh19] by Aron Gohr presents a novel approach in the field of cryptanalysis, specifically targeting the Speck32/64 cipher through the lens of machine learning. Gohr's work is pioneering in several ways, demonstrating for the first time that deep learning can be effectively applied to produce highly competitive cryptographic distinguishers against round-reduced versions of a modern block cipher. The research offers a comprehensive analysis, starting with a detailed calculation of the predicted difference distribution of Speck32/64 under certain assumptions for up to eight rounds. It then introduces deep residual neural networks as powerful tools for developing cryptographic distinguishers, which outperform traditional methods in identifying key characteristics of ciphered texts.

In Gohr's attack, an ML-based distinguisher is trained with the aid of ciphertext pairs. These pairs are generated in two ways: in the first group, they are real ciphertexts (C_0, C_1) of a block cipher for a specific plaintext pair (P_0, P_1), where $P_0 \oplus P_1 = \Delta$; in the second group they are selected randomly (noted by $\in_r$) where $C_0, C_1 \in_r \{0,1\}^n \times \{0,1\}^n$. Then, the accuracy of this distinguisher is evaluated, and if this accuracy is more than 50%, a distinguisher is obtained. It is important to note that the power and efficacy of this ML-based differential attack compare positively with the All-in-One attack [AL13] for SPECK32/64. The All-in-One attack is a powerful differential analysis that considers a set of all output differences for a fixed given input difference Δ instead of one specific differential trail $\Delta \rightarrow \delta$.

$$All - in - one = \{\delta | \delta = E(P_0) \oplus E(P_0 \oplus \Delta)\}$$

The steps of finding an ML-based differential distinguisher for r round of SPECK32/64 cipher are as follows. Considering the feistel structure of SPECK32/64, every plaintext can be represented like $P = (L, R)$ that $L \in \{0,1\}^{16}$ and $R \in \{0,1\}^{16}$ are left and right part of plaintext, respectively. Furthermore, the 32-bit values for plaintexts, ciphertexts, and differences are represented in hexadecimal form, for e.g., $6659 = 0x1a03$.

1. First, 10^7 plaintext pairs $(P0, P1)$ are randomly generated in a way that $\Delta = (L_0 \oplus L_1, R_0 \oplus R_1) = (0x0040, 0x0000)$. Meanwhile, 10^7 labels $Y \in_r \{0,1\}^1$ are randomly generated and allocated to the pairs.
2. if $Y = 0$ the $P1$ is randomly changed to $P_1 \in_r \{0,1\}^{32}$ then all these pairs are encrypted with r rounds of SPECK32/64, and all the ciphertext pairs

$(C0, C1)$ are stored with their corresponding labels in a dataset.

3. An AI machine is trained with the help of these ciphertext pairs. In this training phase, zero label $Y = 0$ means it is a datum from a random permutation, while $Y = 1$ demonstrates a ciphertext for a fixed input difference $\Delta = (0x0040, 0x0000)$.
4. In testing stage, steps (1) and (2) are repeated for another 10^6 pairs, and the accuracy of the machine is measured. If accuracy is more than 50%, then the machine is a differential distinguisher.

Algorithm 3 DL-based Differential Distinguisher for r rounds of Speck32/64

Input: r (number of rounds), AI machine, (C_0, C_1)
Output: Trained AI machine, differential distinguisher status
Generate 10^7 plaintext pairs (P_0, P_1) with $\Delta = (L_0 \oplus L_1, R_0 \oplus R_1) = (0\text{x}0040, 0\text{x}0000)$
Randomly allocate 10^7 labels $Y \in_r \{0, 1\}$ to the pairs
for each pair (P_0, P_1) with label Y **do**
    if $Y = 0$ **then**
        $P_1 \leftarrow P_1 \in_r \{0, 1\}^{32}$
    end if
    Encrypt the pairs with r rounds of Speck32/64 to get ciphertext pairs (C_0, C_1)
    Store (C_0, C_1) with corresponding labels in a dataset
end for
Train DL-distinguisher using the dataset and their corresponding labels
Repeat steps 3-11 for another 10^6 pairs for testing
Measure the accuracy of the DL-based distinguisher
if accuracy $> 50\%$ **then**
    The machine is a DL-based differential distinguisher
end if

Following Gohr's seminal work, several papers have investigated the use of ML in cryptanalysis [YK21, BB22]. Some of these works changed certain steps of the original attack, as mentioned above. For instance, in [BBDH21], there are two major changes. Firstly, it is shown that using a multilayer perceptron (MLP) for training the ML-distinguisher can achieve better results than the CNN network as used in [Goh19]; and secondly, the training dataset containing $C0 \oplus C1$ instead of $(C0, C1)$ is more useful for increasing the accuracy of ML-distinguisher.

The final goal of having a distinguisher is to attack the cipher. By having a r-round ML-based distinguisher, a trivial attack on $r + 1$ rounds of SPECK32/64 cipher can be implemented as follows.

1. For a fixed input difference $\Delta = (0x0040, 0000)$, n pairs of $(P0, P1)$ is formed and their corresponding ciphertext $(C0, C1)^{r+1}$ after $r + 1$ rounds is obtained by asking from an oracle.
2. For all possible subkeys in round $r + 1$, (k^{r+1}), ciphertexts are partially decrypted for one round and $(C0, C1)^r$ saved in a dataset.
3. given each $(C0, C1)^r$ to r round ML-based distinguisher in the test phase, a score is attained for every ciphertext pair.
4. Average all the scores to have a final score for each subkey k^{r+1}.
5. Rank the subkeys based on their score. The subkey with the highest score has the highest probability of being the correct subkey.

Algorithm 4 Attack on $r + 1$ Rounds of Speck32/64 Using ML-based Distinguisher

Input: r (number of rounds for the distinguisher), ML-based distinguisher, Oracle for $r + 1$ rounds encryption, Number of pairs n
Output: Estimated subkey k_{r+1}
For fixed input difference $\Delta = (0\text{x}0040, 0\text{x}0000)$, generate n pairs (P_0, P_1)
Obtain $(C_0, C_1)^{r+1}$ for each pair by querying the oracle for $r + 1$ rounds of encryption
for each possible subkey k^{r+1} **do**
    Partially decrypt $(C_0, C_1)^{r+1}$ for one round using k^{r+1} to get $(C_0, C_1)^r$
    Store $(C_0, C_1)^r$ in a dataset
end for
Initialize an array to store scores for each k^{r+1}
for each $(C_0, C_1)^r$ in the dataset **do**
    Pass $(C_0, C_1)^r$ to the r-round ML-based distinguisher
    Obtain a score for the ciphertext pair
    Add the score to the corresponding k^{r+1}'s total score
end for
Average the scores for each k^{r+1} based on the number of pairs
Rank the subkeys k^{r+1} based on their average score
return The subkey k^{r+1} with the highest average score as the estimated subkey

In deep learning DL-based distinguishers, determining the optimal input difference can significantly improve their performance. The input difference to the best differential characteristic is not a good choice for neural distinguishers. This is because differential characteristics maximize the probability for a given output difference, but this probability is too low for the corresponding output difference to appear often on a single pair.

The method employed by the author for identifying the optimal input differ-

ence (Δ') for deep learning-based cryptographic distinguishers unfolds through a series of meticulously designed steps:

1. **Neural Network Training:** A single-block residual network is initially trained on 10^7 ciphertext pairs, generated from three rounds of the Speck cipher using a constant input difference (Δ). This step is foundational for the network to learn distinguishing characteristics tied to Δ.
2. **Few-Shot Learning Implementation:** The network's second-to-last layer output is then repurposed to train a ridge regression classifier [HK70], albeit on a significantly smaller dataset that introduces new input differences (Δ'). This embodies the few-shot learning approach, enabling rapid adaptation with minimal examples.
3. **Exploration of New Input Differences:** A greedy algorithm, imbued with an exploration bias, is deployed to suggest new candidate differences (Δ'). Beginning with a random difference, the algorithm iteratively flips a bit to evaluate potential improvements or fulfil an exploration criterion.
4. **Evaluation and Selection Process:** Through the greedy algorithm, each new input difference is assessed using the ridge regression classifier. Enhanced scores or satisfaction of an exploration threshold dictate the selection of Δ' for subsequent analysis.
5. **Iterative Optimization:** The process iteratively refines Δ', leveraging the classifier's evaluations to balance between exploiting known effective differences and exploring new candidates, aiming to discover optimally distinguishing input differences.

This methodology not only underscores the innovative application of few-shot learning in cryptanalysis but also highlights the strategic use of machine learning models to enhance the efficacy of cryptographic distinguishers.

2.3.4 Recent Advances and Current Trends

The literature on neural cryptanalysis has grown rapidly since Gohr's 2019 work on SPECK32/64 [Goh19]. A useful high-level overview of progress and open challenges is provided by the recent survey of Gérault [GHHP24], which summarises advances in neural differential distinguishers, data generation strategies, architectures, and evaluation methodology.

Several lines of work have focused on improving the efficiency, robustness, and generality of neural distinguishers. Buacuieti et al. [BBP22] investigate

pruning and architecture refinement for SPECK32/64 and show that substantially smaller networks can remain competitive. Baksi et al. extend neural distinguishers and ML-assisted differential analysis to a broader set of lightweight and permutation-based designs, helping to clarify when ML provides an advantage over classical modelling [BBDH21, BB22]. More recent work has also targeted ciphers standardised or shortlisted in modern lightweight-cryptography efforts, for example by proposing neural distinguishers for GIFT-128 and ASCON [SSL+24]. Recent work has also explored attention-style modules and enhanced data formats for Simeck-family distinguishers (e.g., ECA-enhanced residual networks) [YW24].

Another trend is strengthening related-key and multi-difference settings, which are particularly relevant when classical RX or related-key assumptions interact with learned distinguishers. For example, Wang et al. propose a related-key neural distinguisher using two different input differences [WWS24], and Su et al. present an improved related-key framework that combines classical differential reasoning with neural distinguishers [SRC25]. In parallel, feature engineering and hybrid pipelines are becoming more common; Bellini et al. introduce partial decryption as a feature-engineering technique for neural distinguishers [BBG+25]. Finally, neural techniques have also been adapted to RX settings; for example, recent work improves deep-learning-based RX attacks on Simon and Simeck and discusses key-recovery considerations [LCX+25].

2.3.5 Deep-Learning-Based Differential Distinguishers

In their comprehensive study, Benamira et al. examine machine learning (ML) applications in cryptanalysis, focusing specifically on the neural distinguisher introduced by Gohr for the SPECK cipher [BGPT21]. The methodology employed by the authors to dissect and understand the operational mechanisms of this distinguisher is both rigorous and multidimensional, involving a series of analytical and experimental steps aimed at uncovering the underlying principles that enable the ML model to excel in cryptanalysis tasks.

Initially, the research team embarked on an exploration of the classified sets of ciphertext pairs—both real and random—to identify any discernible patterns that could shed light on the neural network's decision-making criteria. This investigative phase revealed that the distinguisher's efficacy predominantly hinged on its ability to analyze the differential distribution of ciphertext pairs, with a pronounced emphasis on the distributions observed in the penul-

timate and antepenultimate rounds of encryption. This insight was pivotal, as it pointed to the neural network's reliance on specific characteristics of the cipher's differential structure to make accurate distinctions.

To validate these insights, the team experimented, referred to as Experiment A, structured as Algorithm 5:

Algorithm 5 Analysis of Neural Distinguisher for SPECK Cipher

$real_pairs \leftarrow$ Extract real 5-round SPECK-32/64 ciphertext pairs
$sorted_diffs \leftarrow$ Sort diffs. of $real_pairs$ by frequency
for δ in $sorted_diffs$ **do**
    $random_pairs \leftarrow$ Generate random 32-bit nums. with diff. δ
    $scores \leftarrow$ Feed $random_pairs$ to neural net N_5 for scores
    Record number of pairs with scores ≥ 0.5
end for
Analyze score distribution for differential characteristics
$partial_dec_pairs \leftarrow$ Partially decrypt $real_pairs$
    and recompute diffs.
Repeat analysis for $partial_dec_pairs$ to refine understanding
    of distinguisher's reliance on diff. characteristics

This experiment underscored the neural distinguisher's proficiency in leveraging differential characteristics to distinguish between real and random pairs, providing a deeper understanding of machine learning applications in cryptanalysis.

To validate these observations and to further probe the capabilities of neural distinguishers, Benamira et al. undertook the construction of an alternative distinguisher based entirely on classical cryptanalysis techniques. This non-neural distinguisher was meticulously designed to emulate the operational logic deduced from the neural model's functioning. Remarkably, this cryptanalysis-driven approach achieved a level of accuracy comparable to Gohr's original neural distinguisher, indicating that traditional cryptanalytic insights could, in certain contexts, replicate the success of more complex neural models.

Further deepening their inquiry, the authors introduced a simplified machine learning-based distinguisher. This innovative model significantly reduced the complexity inherent in Gohr's deep neural network (DNN) while maintaining close accuracy levels. Crucially, this streamlined approach enhanced the interpretability of the machine learning model, allowing the researchers to gain valuable insights into how the model approximates the Differential Distribution Table (DDT) of the cipher. Through this process, the study illuminated the neural network's internal mechanisms, showing that it effectively constructs a

highly accurate approximation of the DDT during its learning phase, which it then leverages to classify ciphertext pairs.

The methodologies applied by Benamira et al. not only provided a critical examination of the integration of ML with cryptanalysis but also advanced the field by demonstrating the potential for improving the interpretability of neural networks in this domain. By proposing methods to refine Gohr's work and exploring new configurations for neural distinguishers, the research contributes significantly to the ongoing discourse on the role and optimization of machine learning in cryptanalytic endeavors.

Based on their previous findings, in [BGHR22], Bellini et al. presented an alternative approach that does not rely on neural networks for finding the best input difference for DL-based distinguishers. In order to find the best input difference they had a bias score hypothesis which states that the optimal input difference for neural distinguishers cryptographically is the input difference that maximizes the bias of output difference bits. Computing the bias score for block ciphers is infeasible due to the requirement of enumerating all keys and plaintexts. However, we can use an approximation derived from a limited number of samples t:

Definition 2.3.1 (Approximate Bias Score [BGHR22])**.** *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a block cipher, and let $\Delta \in \mathbb{F}_2^n$ be an input difference. The Approximate Bias Score for Δ, denoted by $\tilde{b}^t(\Delta)$, is defined as the sum of the biases of each bit position j in the output difference, computed over t samples. Formally, we have:*

$$\tilde{b}^t(\Delta) = \left| \sum_{j=0}^{n-1} 2 \cdot \frac{\sum_{i=0}^{t} (E_{K_i}(X_i) \oplus E_{K_i}(X_i \oplus \Delta))_j}{t} - 1 \right|$$

The authors' hypothesis is confirmed, and they propose an evolutionary-based search algorithm that leverages the approximate bias score to explore a larger set of candidate input differences. The algorithm starts with a population of randomly generated input differences, and an approximate bias score is computed for each of them. The top 32 input differences with the highest score are retained for further evaluation. The algorithm then proceeds with 50 iterations, during which new individuals are derived and evaluated. To ensure the starting round's influence on the bias score is accounted for, the number of rounds is incremented if the maximum bias score obtained surpasses a threshold limit. At the end of the algorithm, the authors obtain a list of 32 input differences for each round. The final step involves computing a weighted cumulative bias

score for all the obtained input differences from round 1 to round R. The authors' search algorithm based on the biased score demonstrates improved performance in identifying input differences compared to other methods and can be useful for cryptographic applications. Algorithm 6 can show their method. The algorithm has several key parameters, including the initial population size for each generation (P), the mutation probability (p_m), the approximate bias score sample size (t), and the relevance threshold (T). The specific values of these parameters can be adjusted as needed to optimize the algorithm's performance. Also *curr_population*$_i$ indicates the ith bit of the current population we have.

Algorithm 6 Evolutionary optimizer [BGHR22]

init_population $\leftarrow$ [RandomInt$(0, 2^n - 1)$ for 1024 times]
Sort *init_population* by $\tilde{b}_t(\cdot)$ in descending order
curr_population $\leftarrow$ first P elements of *init_population*
for *iter* $\leftarrow 0$ to 50 **do**
    cand $\leftarrow$ [ ]
    for $i \leftarrow 0$ to $P - 1$ **do**
        for $j \leftarrow i + 1$ to $P - 1$ **do**
            if RandomFloat$(0, 1) < p_m$ **then**
                $m \leftarrow 1$
            else
                $m \leftarrow 0$
            end if
            Add *curr_population*$_i$ $\oplus$ *curr_population*$_j$ $\oplus$ $(m \lll$ RandomInt$(0, n - 1))$ to *cand*
        end for
    end for
    Sort *cand* by $\tilde{b}_t(\cdot)$ in descending order
    curr_population $\leftarrow$ first P elements of *cand*
end forreturn *cand*

2.3.6 Evolutionary Algorithms for Parameter Search

Several parts of modern cryptanalysis—including differential-trail search, distinguisher construction, and hyperparameter or input-difference selection for neural distinguishers—can be formulated as optimisation problems over large, discrete search spaces. When gradients are unavailable and the objective is noisy (for example, estimated accuracy of a learned distinguisher), evolutionary algorithms (EAs) provide practical, derivative-free optimisation methods [ES03, SP97].

An EA maintains a *population* of candidate solutions and iteratively improves them using variation operators (mutation and, in many variants, crossover) and selection based on a user-defined *fitness* function. Typical fitness functions in cryptanalysis include empirical distinguisher accuracy, estimated trail probability, or composite scores combining multiple criteria. The algorithm terminates after a fixed number of generations or when progress stalls.

A widely used EA in continuous and mixed spaces is *differential evolution* (DE) [SP97]. DE generates new candidates by adding a scaled difference of two population members to a third member, followed by recombination and selection. In this thesis, we use an evolutionary-search procedure in Chapter 4 to select effective RX-pair parameters (translation δ and rotation offset γ) for deep-learning-based RX distinguishers, following and adapting earlier optimisation-based approaches in neural cryptanalysis [BGHR22]. In practice, the fitness function is derived from distinguisher performance (e.g., accuracy or an accuracy-based score) computed on a validation set, which makes the search robust to noise and to the non-convexity of the underlying problem.

2.4 Homomorphic Encryption and Privacy in Machine Learning

Privacy is a central concern in many machine-learning applications because inputs and model outputs may contain sensitive information. Homomorphic encryption (HE) addresses part of this problem by enabling computation directly on encrypted data, so that a server can perform inference without seeing the plaintext inputs. In practice, HE-based systems must carefully balance accuracy, latency, and memory overhead.

This section introduces HE with emphasis on the CKKS scheme, surveys prior work on privacy-preserving machine-learning inference (including natural language processing), and summarises common challenges and limitations.

2.4.1 Introduction to Homomorphic Encryption (HE)

2.4.1.1 Early Concepts and Breakthroughs

Homomorphic Encryption (HE) is a form of encryption that allows computations to be performed on encrypted data, yielding encrypted results which, when decrypted, correspond to the result of operations applied directly to the

plaintext. This capability is mathematically represented as:

$$\text{Dec}(\text{Enc}(a) \otimes \text{Enc}(b)) = a \otimes b \tag{2.2}$$

where $\text{Enc}(\cdot)$ is the encryption function, $\text{Dec}(\cdot)$ is the decryption function, $\otimes$ represents an operation on encrypted values. The significance of HE lies in its potential to maintain data privacy while allowing data analysis and computation on encrypted data, a critical aspect in cloud computing, data security, and privacy-preserving machine learning applications.

The conceptual foundation of HE dates back to Rivest, Adleman, and Dertouzos in 1978, who first posited the idea of an encryption scheme that could allow direct computations on ciphertexts [RAD+78]. Despite its early conception, practical implementations of HE have only become feasible in recent years, due to significant advances in computational techniques and encryption algorithms.

The progression from theoretical conception to practical application in HE is marked by the development of distinct categories of homomorphic encryption: Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE). PHE is characterized by its capacity to support unlimited operations of a single type—either addition or multiplication—on ciphertexts. This singular operational capability makes PHE highly efficient for specific tasks tailored to either additive or multiplicative operations. The limitation to one type of operation, however, restricts the versatility of PHE in broader application scenarios.

Advancing beyond the singular operation constraint of PHE, Somewhat Homomorphic Encryption (SHE) systems were developed to facilitate a limited number of both addition and multiplication operations. The inception of SHE represented a significant stride towards the ideal of performing comprehensive computations on encrypted data, although the limitations in the number of operations prevent SHE from achieving the full computational flexibility envisioned for homomorphic encryption.

Fully Homomorphic Encryption (FHE) emerges as the culmination of efforts in the HE spectrum, offering the unparalleled capability to perform an unlimited number of both addition and multiplication operations on encrypted data. The advent of FHE has unfolded new paradigms in data privacy, allowing for the first time, the full range of computational operations to be securely executed on encrypted data without exposing the underlying plaintext. This breakthrough

carries profound implications for secure data analysis, cloud computing, and privacy-preserving algorithms, making it a cornerstone for future developments in secure digital technologies.

In the work by Craig Gentry in 2009, a technique known as bootstrapping was introduced, laying the foundation for Fully Homomorphic Encryption (FHE) [Gen09]. Bootstrapping is a mechanism designed to mitigate the challenge of noise accumulation in homomorphic encryption schemes. As encrypted data undergoes successive computational operations, a phenomenon termed as "noise" incrementally builds up in the ciphertext. Left unaddressed, this escalating noise jeopardizes the integrity of the decryption process, potentially leading to incorrect decryption outcomes.

The essence of bootstrapping involves a novel self-referential strategy: a ciphertext is decrypted (to reduce noise) and then re-encrypted. Mathematically, if we consider a ciphertext c that encrypts a message m with noise, the process can be represented as:

$$\text{Enc}'(\text{Dec}(c)) = c' \tag{2.3}$$

where $c' \neq m$ represents the "refreshed" ciphertext with reduced noise. This procedure effectively "resets" the noise level, enabling the continuation of further operations on c' without degrading the decryptability of the ciphertext.

Gentry's introduction of bootstrapping was a pivotal advancement, as it marked the transition from the theoretical possibility to the practical feasibility of FHE, where an unlimited number of additions and multiplications can be applied to encrypted data. Despite its conceptual elegance, bootstrapping is computationally intensive, presenting a significant challenge for the practical application of FHE. Subsequent research in the field has focused on optimizing this process, aiming to make FHE more accessible for real-world applications by reducing the computational overhead associated with bootstrapping.

After the introduction of bootstrapping some HE schemes introduced like BGV [BV14] and BFV [Bra12]. The BGV scheme, a cornerstone in the domain of fully homomorphic encryption (FHE), enables secure computation on encrypted integers, allowing for a wide range of operations including addition and multiplication. A pivotal aspect of BGV is its ability to perform an unlimited number of these operations by effectively managing the noise that accumulates with each computation. This is achieved through a novel technique known as modular switching.

Modular switching is a key feature of the BGV scheme that helps to control

and reduce noise. The essence of this technique lies in its ability to adjust the size of the modulus (a parameter of the encryption scheme) during computations. By cleverly reducing the modulus size in a controlled manner, BGV can decrease the noise in ciphertexts, thereby preserving the integrity of encrypted data and ensuring that decryption remains feasible even after numerous operations.

The BFV scheme stands as another significant advancement in fully homomorphic encryption, tailored specifically for arithmetic operations on encrypted integers. Like BGV, it facilitates secure computation on encrypted data, enabling both addition and multiplication. A distinctive feature of BFV is its focus on optimizing the efficiency of these operations, employing techniques that enhance the scheme's practicality for real-world applications. One of the key strengths of BFV lies in its effective noise management strategy, which ensures that encrypted data remains secure and decipherable even after numerous homomorphic operations.

Despite the significant advances made by the BGV and BFV schemes in the field of homomorphic encryption, they are not without their limitations. One of the primary challenges faced by these schemes is their focus on integer arithmetic, which restricts their applicability for computations involving real numbers or complex data types. This limitation is particularly relevant in the context of data science and machine learning applications, where the ability to work with floating-point numbers is often crucial.

The introduction of the Cheon-Kim-Kim-Song (CKKS) scheme represents a pivotal development in overcoming these limitations. CKKS is designed specifically to handle encrypted computations on real numbers, offering a solution to the challenges posed by integer-only arithmetic in BGV and BFV. This capability opens up new possibilities for privacy-preserving computations in fields that require the handling of complex data types, setting the stage for the next generation of homomorphic encryption applications.

2.4.2 Cheon-Kim-Kim-Song (CKKS) Scheme

The Cheon-Kim-Kim-Song (CKKS) scheme, a pivotal advancement in the domain of fully homomorphic encryption, is carefully designed to accommodate arithmetic on encrypted real or complex numbers, facilitating a wide array of privacy-preserving computations. Central to its operation are three fundamental processes: encoding, encrypting, and noise handling. Encoding trans-

forms real or complex inputs into polynomial representations, a step that enables their encryption and subsequent homomorphic manipulation. The encryption process then secures these representations, ensuring that computations can be performed without compromising data privacy. Noise handling, which includes both relinearization and rescaling, is crucial for maintaining the scheme's integrity and accuracy. Relinearization reduces the size of ciphertexts post-multiplication to manage computational complexity, while rescaling adjusts the ciphertexts' scale to control noise growth, preserving the precision of the encrypted computations. Together, these components form the backbone of the CKKS scheme, enabling it to perform sophisticated encrypted calculations with remarkable efficiency and security.

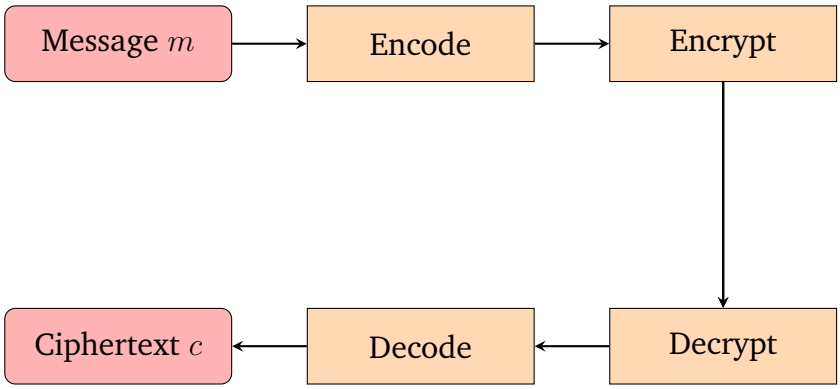


Figure 2.5: High Level View of CKKS

2.4.2.1 Encoding and Decoding in CKKS

In the CKKS encryption scheme, the foundational concept involves leveraging the structure of rings formed by integer polynomials. This process requires transforming the plaintext, which is typically a vector of real or complex numbers z in $\mathbb{C}^{N/2}$, into a polynomial form $m(X)$ that belongs to the ring $\mathbb{Z}[X]/(X^N + 1)$.

To elucidate the mechanism behind the vector-to-polynomial transformation and to explore the computational implications of this transformation, let us consider a simplified example. Here, a vector z in the complex number space $\mathbb{C}^N$ is converted into a polynomial $m(X)$ that is situated within the complex polynomial quotient ring $\mathbb{C}[X]/(X^N + 1)$.

The process of encoding (transforming a vector into a polynomial) and decoding (reverting a polynomial back to a vector) is facilitated by a specific mapping, denoted as σ. This mapping links the complex polynomial quotient ring

$\mathbb{C}[X]/(X^N+1)$ to the vector space $\mathbb{C}^N$. To decode a polynomial $m(X)$ back to its vector representation z, the polynomial is evaluated at the Nth roots of unity $(\xi, \xi^3, ..., \xi^{2N-1})$. These evaluations yield a vector in $\mathbb{C}^N$, establishing a unique correspondence between each vector and its polynomial representation through the mapping σ.

For encoding, the objective is to derive a polynomial $m(X) = \sum_{i=0}^{N-1} \alpha_i X^i$ in $\mathbb{C}[X]/(X^N+1)$ that, when processed through σ, accurately reflects the components of the original vector z. This involves solving a system characterized by a Vandermonde matrix A, where the solution to $A\alpha = z$ provides the coefficients α_i that constitute the polynomial representation of the vector. This polynomial then encapsulates the encoded information of the vector z.

However, the ultimate aim of the CKKS scheme is to adapt this encoding process such that the resultant polynomial resides within the integer polynomial ring $\mathbb{Z}[X]/(X^N+1)$. This adaptation is crucial because it aligns the coefficients of the polynomial with integers, reflecting a transition to a domain where the encoded polynomials inherently support operations within the integer ring.

However, when we use the previous method we cannot be sure to have integer coefficients. For this, we need to use a technique called "coordinate-wise random rounding". This method adeptly rounds a given real number x towards either its floor $\lfloor x \rfloor$ or ceiling $\lceil x \rceil$, depending on the proximity of x to these boundaries. Furthermore, to mitigate potential precision losses during the rounding process, we introduce a scaling factor $\Delta > 0$. This scaling, applied prior to encoding and inversely post-decoding, ensures maintained precision at a scale of $\frac{1}{\Delta}$.

2.4.2.2 Encryption and Decryption in CKKS Scheme

The CKKS scheme utilizes the Ring Learning with Errors (RLWE) problem as its security foundation, where the hardness of solving RLWE ensures the security of the encrypted data. The RLWE problem is a variant of the Learning with Errors (LWE) problem adapted to polynomial rings. It is defined over a ring $R = \mathbb{Z}[x]/(x^n+1)$, where n is a power of 2, making R a ring of polynomials modulo x^n+1. The security of RLWE-based schemes relies on the difficulty of solving the RLWE problem, which, informally, is to distinguish between random elements of $R \times R$ and pairs of the form $(a, a \cdot s + e)$, where a is a uniform random element of R, s is a secret element of R, and e is a "small" error polynomial.

Encryption in the CKKS scheme transforms a plaintext polynomial m into a

ciphertext $c = (c_0, c_1)$ such that decryption recovers m up to a small noise term. Let the secret key be s and let the public key be $p = (b, a) = (-a \cdot s + e, a)$. In the simplified presentation used here, encryption outputs $c = (m + b, a)$, and decryption computes $c_0 + c_1 \cdot s$.

In order to demonstrate the functionality of the addition operation within this encryption scheme, let us consider two ciphertexts $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$. The decryption process for the sum of these ciphertexts can be represented as follows:

$$c_{\text{add},0} + c_{\text{add},1} \cdot s = c_0 + c'_0 + (c_1 + c'_1) \cdot s = c_0 + c_1 \cdot s + c'_0 + c'_1 \cdot s = m + m' + 2e \approx m + m'$$

This equation demonstrates that the result of decrypting the sum of two ciphertexts approximates the sum of their corresponding plaintexts, $m + m'$, with a small error term $2e$ that is typically negligible.

Furthermore, to illustrate the behavior of decryption when a plaintext m' is multiplied by a ciphertext $c = (c_0, c_1)$, consider the following expression:

$$\begin{aligned} m' \cdot (c_0 + c_1 \cdot s) &= m' \cdot c_0 + m' \cdot c_1 \cdot s = m' \cdot (c_0 + c_1 \cdot s) \\ &= m' \cdot (m + e) = m' \cdot m + m' \cdot e \approx m' \cdot m \end{aligned}$$

This illustrates that the decryption of the product of a plaintext and a ciphertext closely approximates the product of the two corresponding plaintexts, $m' \cdot m$, with a minor error term $m' \cdot e$ that is typically considered negligible.

2.4.2.3 Relinearization in CKKS

Ciphertext-to-ciphertext multiplication in the CKKS scheme results in an output ciphertext that encrypts the product of the original plaintexts, but with an increased degree and size. This multiplication operation, for ciphertexts $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$, yields a new ciphertext with components that are polynomials of higher degree. The decryption of this product without any further processing would be as follows:

$$c_{\text{mult}} = c \cdot c' = (c_0 + c_1.s).(c'_0 + c'_1.s) = c_0.c'_0 + (c_0.c'_1 + c'_0.c_1).s + c_1.c'_1.s^2$$
$$= d_0 + d_1 \cdot s + d_2 \cdot s^2,$$

By looking at the above equation, we can see the decryption of ciphertext-to-ciphertext multiplication is changed from $c_0 + c_1 \cdot s$ to $d_0 + d_1 \cdot s + d_2 \cdot s^2$ where $d_0 = c_0.c'_0, d_1 = (c_0.c'_1 + c'_0.c_1), d_2 = c_1.c'_1$, resulting in a ciphertext of degree higher than that of the original ciphertexts, complicating decryption and subsequent operations. This increase in degree and complexity necessitates the use of relinearization, a process designed to reduce the degree of the ciphertext back to its original state, thus maintaining the scheme's efficiency and operability.

Relinearization utilizes an additional set of keys, generated during the key generation phase, known as relinearization keys. These keys enable the transformation of a higher-degree ciphertext back into a two-component form, (c''_0, c''_1), through a combination of multiplication and addition operations that effectively "cancel out" the higher-degree terms. The process can be represented as follows:

$$(c''_0, c''_1) = \textit{Relinearize}(c_{\text{mult}}),$$

where *Relinearize*$(.)$ denotes the relinearization operation, and decrypting $(c''_0, c''_1) \approx m.m'$. This operation is crucial for several reasons:

Maintaining Efficiency: By reducing the degree of the ciphertext, relinearization ensures that the size of the ciphertext remains manageable, thereby preserving the efficiency of storage and computation.

Facilitating Decryption: Keeping the ciphertext within a two-component form simplifies the decryption process, ensuring that it remains consistent with the scheme's decryption mechanism.

Enabling Further Computations: Relinearization allows for the continued application of homomorphic operations on the ciphertexts, such as additional multiplications or additions, without the exponential growth in complexity and size that would otherwise result from repeated multiplications.

In essence, relinearization is a critical procedure within the CKKS scheme that ensures the practicability and security of homomorphic encryption, especially when performing complex operations such as ciphertext-to-ciphertext multiplication.

2.4.2.4 The Rescaling Process in CKKS

After performing a multiplication operation, the resultant ciphertext's scale, Δ, increases TO Δ^2. If left unaddressed, this escalation in scale not only degrades the precision of the computation but also limits the number of sequential multiplications that can be performed, due to the fixed-size modulus in the scheme's parameters.

The rescaling operation mitigates these issues by explicitly reducing the scale of the ciphertext and the corresponding modulus, thus controlling the noise growth. This is achieved by dividing the ciphertext by a power of two (which corresponds to the scale factor) and simultaneously reducing the modulus to match this scale adjustment. The process ensures that the scale of all operands remains consistent across computations, a critical factor for the correctness of subsequent operations, especially when dealing with complex numbers.

By meticulously managing the scale and noise through rescaling, the CKKS scheme ensures that encrypted computations remain both accurate and secure, supporting a broader range of arithmetic operations without significant loss of precision.

2.4.3 Privacy-Preserving ML Models

In machine learning, the quest for privacy-preserving models has become increasingly paramount, particularly in applications that handle sensitive or personal data. Among the various domains, Natural Language Processing (NLP) stands out due to its direct interaction with human language – a rich source of personal and sensitive information. The application of privacy-preserving techniques in NLP mitigates risks associated with data exposure and opens avenues for secure, confidential data analysis in areas ranging from sentiment analysis to healthcare. While privacy-preserving models have found utility across a spectrum of ML applications, such as image processing and financial data analysis, their integration into NLP poses unique challenges and opportunities. The specificity of textual data requires novel approaches to ensure that encryption or data anonymization techniques maintain the integrity and utility of the processed data. This section studies the advancements and methodologies developed to safeguard privacy in NLP applications, setting a foundation for exploring the intersection of homomorphic encryption in privacy-preserving sentiment analysis.

2.4.3.1 Privacy-Preserving for Natural Language Processing (NLP)

Privacy concerns in Natural Language Processing (NLP) stem from the inherently sensitive nature of textual data, which often encapsulates personal information, opinions, and behaviors. The imperative to develop privacy-preserving NLP models is thus not only a technical challenge but also a moral obligation, ensuring that insights gleaned from textual data do not come at the expense of individual privacy. Various techniques have been proposed and employed to safeguard privacy in NLP, each with its own set of advantages and challenges.

Data anonymization and pseudonymization serve as the first line of defense in privacy preservation, aiming to remove or obfuscate personal identifiers in textual data. While these methods can reduce the risk of direct identification, they often struggle with the subtleties of language that can indirectly reveal personal information. Differential privacy introduces a mathematical framework to quantify and control the privacy loss incurred when querying datasets, offering robust privacy guarantees for NLP applications [Hab21, MRTZ18]. However, implementing differential privacy in NLP is fraught with challenges, particularly in balancing privacy guarantees with the utility of the output data.

The advent of federated learning has opened new avenues for privacy-preserving NLP by enabling model training across multiple decentralized devices or servers [MMR$^+$17]. This approach keeps the training data localized, minimizing the risk of exposure. Nevertheless, federated learning in NLP faces hurdles in terms of model synchronization, data heterogeneity, and communication overheads. Homomorphic encryption represents a pivotal advancement in privacy-preserving NLP, allowing operations on encrypted textual data and enabling models to learn from data without ever accessing it in its raw form. Despite its promise, the computational complexity and efficiency of homomorphic encryption remain significant obstacles to its widespread adoption.

These privacy-preserving techniques are not without their trade-offs, particularly regarding the balance between privacy protection and the utility of NLP models. Techniques like homomorphic encryption can significantly impact model performance and computational efficiency, posing a dilemma for practitioners. Moreover, the scalability of these techniques and their computational overheads are critical considerations, especially for large-scale NLP tasks. While the above methods contribute to privacy-preserving practices in NLP, they do not guarantee provable security. To achieve confidentiality and integrity, cryptographic primitives like encryption schemes and signature schemes come into

play, which is why we utilize homomorphic encryption.

2.4.4 Recent Advances and Future Directions

Recent surveys of CKKS-based homomorphic encryption for machine-learning workloads provide a useful perspective on current capabilities and bottlenecks (e.g., ciphertext growth, multiplicative depth, rescaling strategy, and implementation-level optimisations) [WWL+25]. Building on this broader view, the remainder of this section summarises how recent work addresses efficiency and scalability in privacy-preserving inference, particularly for natural-language processing tasks.

Recent advancements in encryption technologies and innovative approaches to model training and data processing offer hope for overcoming the above-mentioned challenges. The development of more efficient homomorphic encryption algorithms and hybrid approaches that combine different privacy-preserving techniques could potentially enhance privacy in NLP applications without compromising on utility or performance.

2.4.4.1 HE based Techniques

Homomorphic encryption in NLP unfolds in two main directions: training and inference, with a predominant focus on the latter due to computational feasibility. While training NLP models directly on encrypted data often remains computationally infeasible, HE can perform better during the inference phase. In this approach, a model is initially trained on raw data, and then, to maintain privacy, inputs for inference are encrypted. This method allows for the secure analysis of sensitive text data, ensuring that personal or confidential information is protected during the NLP tasks such as sentiment analysis or language translation. Despite its limitations in the training phase, the use of HE for inference in NLP signifies a pivotal stride towards balancing data privacy with the advancing capabilities of text analysis technologies.

The earliest works on privacy-preserving neural Networks (PPNN) with provable security started with Barni et al. [BOP06]. The paper introduces a privacy-preserving protocol for neural network (NN) computations that carefully combines HE with principles of Secure Multi-Party Computation (SMC). Utilizing HE, the protocol enables linear operations on encrypted data, thus preserving the confidentiality of user inputs. Additionally, it leverages SMC techniques to facilitate secure computations among multiple parties without revealing indi-

vidual inputs, especially for non-linear operations within the NN. The protocol employs a Privacy-Preserving Scalar Product Protocol (PPSP), based on HE, for securely computing scalar products that are crucial for the neural network's operations. This innovative approach ensures the privacy of both the user's data and the neural network's proprietary configuration, embodying a significant advancement in the domain of secure and private computations.

After the introduction of Fully Homomorphic Encryption, CryptoNets [GBDL+16] was introduced. This method allows sensitive data, such as medical or financial information, to be encrypted and processed by cloud services without exposing the actual data. By converting traditional neural networks into CryptoNets, the service can perform computations on encrypted data and generate encrypted predictions. The encryption ensures that the service provider cannot access the plaintext data or the results, preserving user privacy. Utilizing homomorphic encryption, CryptoNets cover many ML and NN algorithms using HE through the SEAL library [20]. It demonstrates practical applicability by achieving 99% accuracy on the MNIST dataset, with the capability to process approximately 59,000 predictions per hour on a standard PC.

In the context of machine learning and artificial intelligence, certain operations such as matrix multiplication, vectorization, and activation functions with non-linear operations can be challenging to perform directly on encrypted data. This can result in reduced performance when using privacy-preserving techniques. Solutions to the challenges of performing machine learning on encrypted data often involve trade-offs between privacy and accuracy. A common approach is to use polynomial machine learning, which approximates an algorithm as a polynomial of a given degree. In CryptoDL, Hesamifard et al. [HTG17] develop new methods for running deep neural networks (DNNs) on encrypted data, focusing on classification tasks with convolutional neural networks (CNNs) within the constraints of homomorphic encryption (HE). The core challenge addressed is the non-compatibility of activation functions such as ReLU, Sigmoid, and Tanh with HE operations, which are limited to addition and multiplication. The solution involves approximating these activation functions with low-degree Chebyshev interpolation suitable for HE. The study trains CNNs on encrypted data through this approximation and evaluates their performance on standard datasets like MNIST and CIFAR-10.

Zhang et al. [ZXW21] present a framework aimed at enhancing the computational efficiency of privacy-preserved machine learning as a service (MLaaS). It addresses the challenge of the significant computational overhead introduced

by HE-based linear computations, which dominate the processing time in deep neural network applications. By optimizing these computations, GALA reduces the number of permutation operations required for dot product and convolution in MLaaS, leading to substantial speedups. Specifically, GALA introduces a row-wise weight matrix encoding and a first-add-second-permute approach for kernel grouping in convolution, efficiently reducing permutation operations. This deep optimization of HE-based linear computation allows GALA to achieve computational speed improvements, making it a valuable contribution to the field of privacy-preserving neural networks.

Recurrent Neural Networks (RNNs) pose greater challenges for privacy preservation using Homomorphic Encryption (HE) compared to Convolutional Neural Networks (CNNs), primarily due to their sequential data processing nature. This inherent characteristic of RNNs, essential for handling time-series data or sequences like text, results in complex, interdependent computations that amplify the computational overheads and latency when encrypted, making them less feasible for HE applications than the more parallelizable structure of CNNs. Several studies have therefore proposed methods for reducing this limitation.

Podschwadt and Takabi [PT21] introduce Parallel RNN Blocks (PRB) for non-interactive privacy-preserving inference on recurrent neural networks (RNNs) using homomorphic encryption. Traditional approaches to secure RNN computation have necessitated interaction between the data owner and the cloud service, primarily to manage the multiplicative depth (MD), a significant limiting factor in HE-based solutions due to noise accumulation in the encryption scheme. PRB addresses this by reducing the MD through dimensionality reduction of input data, which is segmented into smaller, manageable chunks processed independently. This architecture allows for the offloading of computation to the cloud without requiring back-and-forth communication for noise management, thus preserving data privacy while ensuring the cloud cannot infer sensitive information from the computation. Evaluation on real-world datasets demonstrates the effectiveness of PRB, showing only a minimal performance drop compared to baseline models.

Other research extends beyond traditional RNNs to more advanced variants such as Long Short-Term Memory (LSTM) networks, which also face similar computational complexities in privacy-preserving applications using HE. Liu et al. [LJLA17] transformed a regular LSTM into a privacy-preserving LSTM by replacing activation functions with approximations, which were evaluated by the client. However, this leaked some information about the model to the client.

While ongoing efforts focus on enhancing techniques to reduce the computational overhead associated with applying different approximation techniques to non-linear operation tasks in neural networks, the research landscape is also embracing hybrid approaches. These strategies integrate HE with other privacy-enhancing technologies such as Multi-Party Computation (MPC) and Federated Learning (FL). By leveraging the synergistic potential of these combined methodologies, researchers aim to develop more efficient and scalable privacy-preserving solutions, addressing the limitations of HE in isolation and expanding the toolkit available for secure data processing in ML and beyond.

2.4.4.2 Hybrid Approaches

The RHODE framework [SDP+22] enables privacy-preserving training and prediction on RNNs with federated learning by relying on multi-party HE. It addresses the need for data confidentiality in the training and prediction phases by securing training data, model parameters, and prediction data against passive-adversary gradient attacks. A key innovation in RHODE is its multi-dimensional packing scheme, enhancing SIMD operations under encryption for efficient parallel batch sample processing. This method is pivotal for mitigating exploding gradients, a notable challenge in RNN training. Through gradient clipping approximations, RHODE adeptly manages secure computations under encryption.

Multi-party computation (MPC) has emerged as a compelling technique for enhancing privacy in Natural Language Processing (NLP) by enabling parties to jointly compute functions over their inputs while keeping those inputs private. In the context of NLP, MPC allows for the collaborative training of models or analysis of textual data without revealing the actual data to any of the parties involved. This approach is particularly relevant for applications where data privacy is paramount, such as in the processing of sensitive communications or personal documents. By distributing the computation process and ensuring that no single party has access to the complete dataset, MPC can significantly mitigate the risks associated with data privacy breaches. Despite its potential, the practical implementation of MPC in NLP faces challenges related to computational efficiency and the complexity of managing communication protocols among multiple parties.

Jestine et al. in [PAM+21] explores an approach to enhance privacy in handling sensitive time-series data for deep learning models, particularly LSTM, using Homomorphic Encryption (HE). It introduces a "Collective Learning" pro-

tocol where time-series data can be shared among entities in encrypted form, allowing partial training of a binary classifier model. The protocol involves encrypting data feature activations with HE and conducting the training of the last layers through encrypted logistic regression. Evaluated on the MIMIC-III dataset using an LSTM network, the protocol utilizes the CKKS encryption scheme for its compatibility with real numbers and demonstrates improved prediction accuracy for in-hospital mortality, showcasing the practical viability and effectiveness of the approach.

SecureRC [GY23] presents a novel method to execute relation classification in NLP while maintaining data privacy. Utilizing secure MPC, the system, SecureRC employs an attention-based GRU network adapted for privacy through special protocols for non-linear functions like sigmoid and tanh, handled by secure computation methods, including additive secret sharing. This approach safeguards data against semi-honest adversaries, making relation classification feasible without exposing sensitive data or model details, thus complying with stringent privacy regulations. SecureNLP [FHL+20], a system utilizing secure multi-party computation (MPC) to enable privacy-preserving execution of RNN-based seq2seq models with attention for NLP tasks, was introduced by inspiration from SecureRC. It focuses on distributed protocols for non-linear functions like sigmoid and tanh within RNNs, essential for maintaining data confidentiality in sequence transformations. SecureNLP ensures that computations on encrypted data do not compromise privacy, verified under the semi-honest model, thus making it suitable for sensitive applications requiring robust data protection.

The exploration of hybrid approaches marks an advancement in privacy-preserving machine learning. By combining the strengths of homomorphic encryption, Multi-Party Computation, and Federated Learning, these methods not only enhance the practical feasibility of secure NLP tasks but also pave the way for more robust and versatile privacy solutions. As this research area continues to evolve, it holds the potential to greatly expand the applications of NLP in sensitive environments, ensuring that data privacy and utility go hand in hand.

2.4.5 Challenges and Limitations

Privacy-preserving techniques in machine learning, particularly those involving homomorphic encryption (HE) and hybrid methods, encounter challenges across training and inference phases. The differentiation between the applica-

tions in these phases reveals distinct hurdles that impact the practical deployment and efficiency of these techniques.

In the training phase, the integration of HE with ML models poses substantial computational challenges. Training models on encrypted data drastically increases the computational complexity and prolongs the training duration. This is primarily due to the encryption schemes' overhead, which requires more intensive processing power than operations on plaintext. Furthermore, the limited ability of current HE technologies to support complex arithmetic operations restricts the types of algorithms that can be efficiently implemented. This makes training sophisticated models like deep neural networks particularly challenging, often necessitating compromises in model accuracy or training depth.

While generally more feasible, the inference phase still faces significant barriers when employing HE exclusively. Although the data remains secure, the latency involved in processing encrypted inputs can be prohibitively high, particularly for applications requiring real-time analysis. This affects the user experience and limits the practical utility of such privacy-preserving models in time-sensitive environments.

Hybrid approaches, which combine HE with other privacy-enhancing technologies like Multi-Party Computation (MPC) and Federated Learning (FL), aim to address some of these issues by distributing the computational load and enhancing data privacy without full reliance on HE. These methods can reduce the burden on any single computation node and offer a more scalable solution. However, hybrid models introduce their own complexities. Coordinating computations across multiple parties, as required in MPC and FL, adds layers of communication and potential security vulnerabilities. Ensuring that all parties are reliable and that data transmission is secure can be challenging, and any weakness in the chain can compromise the entire system's integrity.

Additionally, there are concerns regarding the balance of privacy and model performance. In many cases, enhancing privacy through techniques like HE and MPC can lead to a degradation in the model's accuracy or sensitivity. This trade-off is a critical consideration for applications where high accuracy is essential, such as in medical or security-related fields.

Overall, while the potential of privacy-preserving techniques in NLP is immense, significant challenges must be addressed to maximize their practical applications. Continued research and development are essential to overcome these limitations and enhance the computational efficiency, accuracy, and secu-

rity of these methods.

Chapter 3

Reducing the Time Complexity of Machine-Learning-Assisted Differential Attacks via Bit Selection

This chapter studies how to reduce the computational and memory cost of machine-learning-assisted differential cryptanalysis. Building on the neural differential distinguisher approach introduced by Gohr, we propose a *partial-differential machine-learning distinguisher* (PD-ML-based distinguisher) that is trained on a selected subset of output-difference bits instead of the full cipher state. We evaluate the approach on SPECK32/64 and show that carefully selected bits can retain most of the distinguishing power while lowering training and attack complexity.

To support principled bit selection, the chapter introduces an empirical scoring method that estimates the contribution of each output-difference bit to distinguisher accuracy. The resulting scores identify a small set of consistently informative bits, enabling compact PD-ML distinguishers and reducing the cost of downstream procedures such as key averaging.

Parts of this chapter are based on our publication in the proceedings of the 15th International Symposium on Foundations and Practice of Security (FPS 2022). [ERP22]

3.1 Introduction

Block ciphers are central to symmetric cryptography and are deployed across a wide range of systems. In recent years, the growth of resource-constrained devices (for example, in IoT and embedded platforms) has motivated extensive research on lightweight cryptography (LWC). To support consistent evaluation and deployment, the US National Institute of Standards and Technology (NIST) has run a standardisation process for lightweight cryptographic algorithms [BÇMT18].

Given the broad deployment of block ciphers, rigorous security evaluation remains essential. Classical techniques such as differential cryptanalysis [BS91] and linear cryptanalysis [Mat93] are still among the most effective tools for assessing security margins, including for modern LWC designs. However, these methods can require large datasets and substantial computation, and searching for strong distinguishers can become infeasible in practice.

A major research direction is therefore automation and acceleration of cryptanalysis. Constraint- and optimisation-based approaches (for example, using solvers such as Gurobi [GO21]) can speed up parts of the search process. More recently, machine learning has been used to learn statistical structure directly from data and to construct effective neural distinguishers. Gohr's work in 2019 demonstrated that ML-based differential distinguishers can outperform traditional distinguishers for reduced-round SPECK32/64 [Goh19].

This chapter focuses on reducing the memory and computation cost of ML-assisted differential attacks. To the best of our knowledge, it introduces the first PD-ML distinguisher, which trains on selected output-difference bits rather than the full 32-bit difference. We also present an empirical method to score bit effectiveness and show, for SPECK32/64, that the resulting scores can reduce the cost of key-averaging-style extensions.

In the setting considered here, efficiency is not merely an optimisation target but a condition for practical usefulness. An ML-based distinguisher is valuable only if its input dimensionality, training cost, and downstream key-guessing cost remain low enough to make it usable within a cryptanalytic workflow. This is especially relevant for lightweight designs such as SPECK32/64, where one would like the evaluation method itself to remain computationally economical. For this reason, the aim of this chapter is not to maximise classifier accuracy in isolation, but to improve the balance between distinguishing power and attack cost. A small loss in accuracy is worthwhile when the distinguisher remains

clearly better than random while materially reducing the complexity of subsequent attack steps.

3.1.0.1 Contribution

This chapter makes three main contributions.

First, we introduce the PD-ML distinguisher. The key idea is that an effective neural distinguisher does not necessarily require the full output-difference vector $\delta = C_0 \oplus C_1$ as input. By restricting the feature set to a subset of bits, we can reduce training cost and, importantly, reduce the amount of last-round key material that must be guessed in downstream attacks.

Second, we evaluate PD-ML distinguishers on SPECK32/64, a common reference cipher in the ML-cryptanalysis literature [Goh19, BGPT21, BBDH21]. We show that carefully selected 8-bit feature sets can achieve 68.8% accuracy for 6-round SPECK32/64 in the original experiment, compared with 72% when using the full 32-bit difference, whereas poorly chosen 8-bit sets approach random performance. To test whether this pattern is stable rather than seed-dependent, we reran the main models over 10 random seeds and obtained mean accuracies of $71.6\% \pm 0.8$ for the 32-bit model, $68.1\% \pm 0.9$ for the selected 8-bit model, and $51.7\% \pm 0.7$ for the lowest-scoring 8-bit model.

Third, we propose an empirical scoring method that estimates the contribution of each output-difference bit to distinguisher accuracy. Using these scores, we demonstrate a practical reduction in the complexity of Gohr's key-averaging procedure for extending a 6-round distinguisher to 7 rounds: from 2^{16} to 2^{11}, at the cost of a small accuracy decrease. In the rerun study over 10 random seeds, the corresponding mean accuracies were $61.2\% \pm 0.6$ for the full-state key-averaging baseline and $58.4\% \pm 0.7$ for the PD-ML variant.

3.1.1 Outline

This chapter is organised as follows. Section 3.2 summarises the SPECK32/64 block cipher and reviews Gohr's key-averaging procedure. Section 3.3 introduces the PD-ML distinguisher concept and evaluates it on SPECK32/64. Section 3.4 presents a scoring method to estimate bit effectiveness and uses the resulting scores to select compact feature sets. Finally, Section 3.5 concludes the chapter.

3.2 Preliminaries

3.2.1 The SPECK cipher

SPECK is a family of lightweight block ciphers designed by the NSA in 2013 [BSS+15]. The family includes multiple block and key sizes; in this chapter we evaluate SPECK32/64 (32-bit block size, 64-bit key) with 22 rounds. SPECK is an iterative design: a round function is applied repeatedly to transform the plaintext into the ciphertext.

SPECK is a Feistel cipher. Accordingly, the plaintext is divided into two equal parts (R, L), and in the case of 32 bits block size $R, L \in \{0,1\}^{16}$, then the below function applies to inputs at each round:

$$SPECK32/64: \quad \begin{aligned} L_{r+1} &= ((L_r >> 7) \boxplus R_r) \oplus k_r \\ R_{r+1} &= (R_r << 2) \oplus L_{r+1} \end{aligned} \tag{3.1}$$

In equation (3.1) $<<$ and $>>$ denote cyclic left and right rotations, respectively, $\boxplus$ is modular addition, and $\oplus$ is an exclusive OR (XOR).

3.2.2 Key Averaging Algorithm

The key-averaging procedure uses an r-round ML-based distinguisher to enhance their effectiveness and create $(r+1)$-round ML-based distinguishers. The algorithm operates by first decrypting one round of a given $(r+1)$-round pair of ciphertexts with all potential subkeys from a predefined key space. This produces a variety of r round ciphertexts for each subkey, each of which is then evaluated by a pre-trained r round ML-based distinguisher. The new $(r+1)$-round distinguisher is then obtained by distinguishing outputs based on the likelihood score, leveraging patterns learned from historical data.

Each decrypted output is scored, and these scores are aggregated across all subkeys by computing their average. This averaging process serves to combine the insights provided by the ML-based distinguisher into a unified metric that assesses the likelihood of each key being the correct one. The average scores are then transformed into likelihood ratios, providing a practical basis for distinguishing between ciphertext pairs generated by the cipher and randomly generated pairs.

The procedure is summarised in Algorithm 7.

Algorithm 7 Key-Averaging Procedure

Input: r-round distinguisher; set of ciphertext pairs $\{(C_0^{r+1}, C_1^{r+1})\}$
Output: New $(r+1)$-round distinguisher
1: **for** each ciphertext pair (C_0^{r+1}, C_1^{r+1}) **do**
2: **for** each subkey k in the key space **do**
3: Decrypt (C_0^{r+1}, C_1^{r+1}) using k to obtain decrypted outputs (C_0^r, C_1^r).
4: Compute score s_k using the ML-based distinguisher on (C_0^r, C_1^r).
5: **end for**
6: Compute the average score for the pair: $S = \frac{1}{N} \sum_k s_k$.
7: Transform S into a likelihood ratio.
8: **if** S > 0.5 **then**
9: (C_0^{r+1}, C_1^{r+1}) is generated by block cipher
10: **else**
11: (C_0^{r+1}, C_1^{r+1}) is generated randomly
12: **end if**
13: **end for**
14: **return** The new distinguisher

3.3 Partial-Differential Machine-Learning Distinguisher (PD-ML)

In this section, we investigate whether an effective neural distinguisher can be trained using only a subset of output-difference bits rather than the full block, as is more common in prior work. This motivates the introduction of a Partial Differential ML-based distinguisher.

To evaluate the security of a cipher, the complexity of the cryptanalysis must be less than that of a brute force attack. The complexity of a brute force attack on a block cipher is defined as $2^{\min(|k|,|n|)}$, where $|k|$ and $|n|$ represent the key and block sizes, respectively. As detailed in Chapter 2, employing an ML-based distinguisher for key recovery requires guessing all subkeys in the final round. For attacking r rounds of the SPECK32/64 cipher, this implies that an attacker must guess all 16 bits of the subkey k_r. Consequently, the complexity of this attack is 2^{16}, substantially less than the 2^{32} required for a brute-force attack, indicating that the attack is meaningful in the SPECK32/64 setting considered here.

Conversely, in ciphers with Substitution-Permutation Network (SPN) structures, such as AES, the complexity of guessing the final-round subkey may approach that of brute force, depending on the attack setting. In this research, we train Partial Differential distinguishers (PD-ML distinguishers) and evalu-

ate their accuracy relative to full-state differential distinguishers for 6-round SPECK32/64. Our findings show that PD-ML distinguishers can differentiate between output differences produced by SPECK32/64 and those of random outputs. In the SPECK32/64 setting studied here, using only a subset of bits also reduces the number of last-round key bits that must be guessed in downstream attacks.

3.3.1 Methodology

To demonstrate the practicality of training an ML-based distinguisher using partial differences between plaintext-ciphertext pairs (P_0, P_1) and (C_0, C_1), we established an experimental framework. In this setup, several PD-ML distinguishers were trained specifically for six rounds of SPECK32/64, and their accuracy metrics were carefully recorded. We selected SPECK32/64 because it is the standard benchmark in prior ML-based differential-cryptanalysis work [Goh19, BGPT21, BBDH21], which makes comparison more meaningful, and because its small block/key configuration makes repeated subset experiments and key-averaging reruns computationally feasible. The choice of six rounds follows the same logic: this is the regime in which prior neural distinguishers are already known to be effective, so it provides a controlled setting for evaluating whether feature reduction preserves useful distinguishing information. Accordingly, the experiments in this chapter are designed to support a focused claim about efficiency-preserving feature selection on a well-established benchmark, rather than a universal claim about all block ciphers.

The encryption process for the plaintext pairs was confined to six rounds, aligning with the benchmarks set by existing research, notably [Goh19]. This reference shows that effective distinguishers for up to six rounds of SPECK can be trained efficiently using ten-layer residual networks. While the extension of distinguishers to the seventh and eighth rounds generally necessitates more advanced techniques like *Key Averaging* [Goh19], our focus remains on the initial six rounds. This approach is consistent with previous studies in the literature to which our work is compared. However, for a comprehensive analysis, we explore the extension of our distinguisher using Key Averaging in Section 3.4.2. Our findings indicate that, for the experimental setting considered here, employing our method also reduces the complexity of key averaging.

In the development of our Partial Differential ML-based distinguisher (PD-ML-based distinguisher) for SPECK32/64, each bit of the output difference, δ,

serves as a distinct feature. The training process involves using subsets of these bits, rather than the full set of 32 bits, to assess the feasibility and efficiency of using reduced data. To empirically determine how much the output difference can be reduced without substantially compromising the accuracy of the PD-ML-based distinguisher, we structured our experiment as follows: Initially, the distinguisher is trained using only the least significant bit (LSB) of δ as a feature, and its accuracy is recorded. Subsequently, the number of features is incrementally increased by one—specifically, the second least significant bit of δ—and a new partial distinguisher is trained. This procedure is repeated until a total of 32 distinguishers have been evaluated. The specifics of this experimental setup are detailed in Algorithm 8.

The plaintext difference, Δ, used to generate the ciphertext differences is set to $(0x0040, 0x0000)$, and the dataset comprises 32-bit differences $\delta = [\delta_0, \ldots, \delta_{31}]$. To ensure the repeatability and reliability of our findings, this entire process is conducted three times, each with a different pair of (P_0, C_0) and (P_1, C_1), while maintaining the condition that $P_0 \oplus P_1 = \Delta = (0x0040, 0x0000)$.

Because the procedure in Algorithm 8 requires the repeated retraining of many distinguishers, the goal of model selection in this chapter is not to find the most accurate architecture at any computational cost, but to obtain a stable and computationally practical configuration for repeated comparison across many bit subsets. Accordingly, we adopted a limited manual tuning approach around compact feed-forward models rather than an exhaustive search over heavier architectures. The final three-layer MLP with hidden widths 32, 64, and 32 was retained because it provided a good trade-off between validation accuracy, training time, and repeatability across experiments. The use of an MLP is also consistent with the observations of Baksi et al. [BBDH21], who reported that MLP-based differential distinguishers can be more efficient than CNN-, residual-, or LSTM-based alternatives for this type of input.

The remaining hyperparameters were chosen according to the same principle. Ten epochs were used as a compromise between convergence and total experimental cost, since longer runs would have substantially increased the cost of retraining the model many times, while shorter runs produced less stable accuracy estimates. The training and validation sizes (10^7 and 10^6, respectively) were selected to reduce the variance of the reported accuracies across different feature subsets. Adam was used for robust optimisation in the binary-classification setting, and the cyclic learning-rate strategy from Gohr [Goh19] was retained because it is standard in this line of work and gave reliable conver-

gence in our experiments. L2 regularisation with $c = 10^{-5}$ was used as a light penalty to control overfitting without materially increasing training complexity. Overall, the architecture and training setup were selected to make the repeated subset-comparison experiments feasible, reproducible, and comparable under a fixed computational budget.

Baski et al. demonstrated in [BBDH21] that Multilayer Perceptron (MLP) architectures exhibit higher efficiency compared to Convolutional Neural Networks (CNNs), including Residual Networks, or Long Short-Term Memory (LSTM) networks, for training an ML-based differential distinguisher. Consequently, in this study, we employ a Multilayer Perceptron (MLP) consisting of three dense layers with a sigmoid activation function. The configuration of neurons in these layers is 32, 64, and 32, respectively, chosen through a systematic fine-tuning process. For each variation in the number of input bits, a distinct MLP was trained over ten epochs on a dataset comprising 10^7 instances of $C_0 \oplus C_1 = \delta$. Additionally, a validation set of 10^6 samples was used to monitor performance. The optimization of the model was guided by a loss function combining binary cross-entropy and L2 weight regularization, with a regularization parameter of $c = 10^{-5}$, utilizing the Adam optimization algorithm [KB14]. The training process incorporated a cyclic learning rate strategy as detailed in [Goh19], with all other parameters set to default values in Keras [Cho15].

Algorithm 8 Training PD-ML differential distinguishers

Data set for training the machines: $\delta = [\delta_0, \ldots, \delta_{31}]$
Accuracy of machines: A with $i = 31, j = 0$
Initialize an empty array X_{temp}
Initialize array A with size 32
while $i \geq 0$ **do**
    $X_{temp} = \delta[i]||X_{temp}$
    $D \leftarrow$ TRAINMACHINE(X_{temp})
    $A[j] \leftarrow$ ACCURACYTEST(D)
    $j = j + 1, i = i - 1$
end while

In each iteration of Algorithm 8, we concatenate difference bits based on their position. If we assume that the accuracy of the machines entirely depends on the number of bits, rather than which bits are selected, then a different selection method should produce comparable results, accounting for statistical differences. In order to verify this hypothesis, we repeat the above process using Algorithm 8; however, this time, instead of starting from the least significant bit δ_{31}, we start from the most significant bit (MSB) δ_0, and then we concatenate

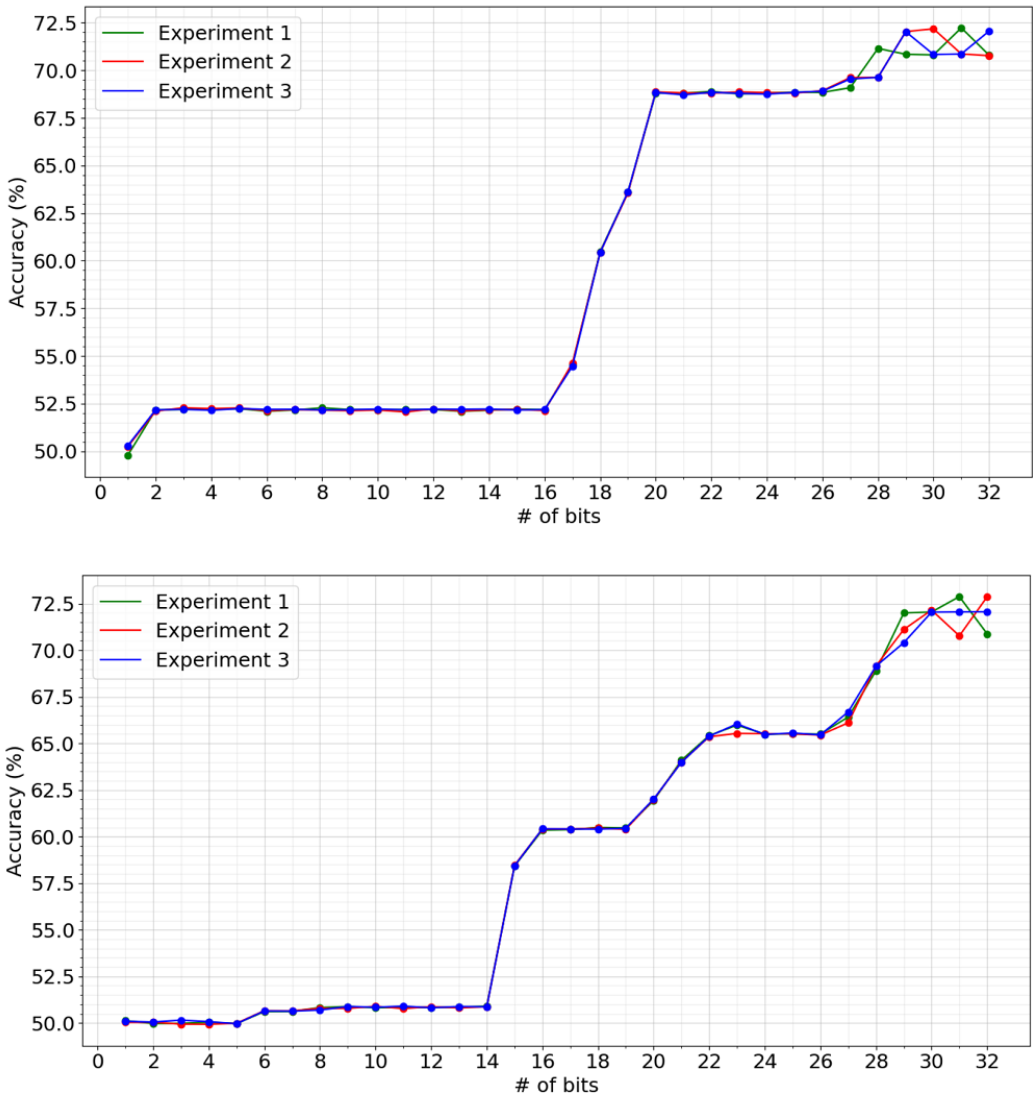


Figure 3.2: Accuracy of ML-based distinguishers according to number of bits in MSB→LSB direction for 6 rounds of SPECK32/64

the next MSB to our feature space for the next iteration of the while loop and trained our machine.

The results of the experiment for SPECK32/64 are shown in Figures 3.1 (LSB→MSB) and 3.2 (MSB→LSB), and are discussed in the following. The results clearly indicate that a partial differential ML-based distinguisher is effective, as a distinguisher can be obtained for a reduced number of bits. However, by looking at the figures, it is also immediately evident that the accuracy changes if we change the selection of bits. These initial experimental results indicate that different bits have a different impact on the machine accuracy. We build on these findings in Section 3.4, where we analyse in detail the impact each bit has in the effectiveness of the partial differential ML-based distinguisher.

3.3.2 Results and Discussion

A review of Figures 3.1 and 3.2 shows that a distinguisher can indeed be constructed from only a subset of the output-difference bits. Specifically, Figure 3.1 indicates that a usable machine is obtainable with only the first 20 bits of the output difference. By contrast, Figure 3.2 shows that when training starts from the most significant bit and proceeds towards the least significant bit, 28 bits are required to reach a similar level of accuracy. The importance of this comparison is that it demonstrates that the usefulness of a partial input is determined not only by the number of retained bits, but also by which bits are retained.

This is a meaningful cryptanalytic observation rather than merely a plotting effect. If the distinguishing signal were distributed uniformly across the output difference, then changing the order in which bits are accumulated should not materially alter the resulting accuracy curves. The fact that the two directions behave differently therefore suggests that the non-random information learned by the model is concentrated unevenly across the output-difference vector. For the present chapter, this matters because it motivates feature selection as a structure-aware procedure rather than as arbitrary truncation.

To ensure the stability of these findings, the experiment was replicated three times for both bit orderings. Each iteration is represented by a distinct color in Figures 3.1 and 3.2, confirming that the overall pattern is reproducible across varying datasets. As the number of features exceeds 28, a slight fluctuation in accuracy is observed among the experiments. This is likely due to the use of only 10 training epochs, which may not allow sufficient convergence for models with a larger input space. However, this variability is not present when the feature set is kept below roughly 22 bits in both ordering strategies. Practically, this means that compact PD-ML distinguishers are not only cheaper to train, but also more stable under the training budget used here.

These observations clarify why efficiency matters in this chapter. The relevant question is not whether the full 32-bit input can deliver the highest possible accuracy, but whether enough distinguishing signal can be retained while reducing the feature space and, with it, the cost of training and later use. In this sense, the partial-difference setting is valuable because a distinguisher that is slightly less accurate but substantially cheaper may be more useful in practice than a marginally stronger model that requires the full state. The results in Figures 3.1 and 3.2 therefore support the chapter's central claim: useful ML-based differential distinguishers can be built from partial output differences, provided

that bit selection is done carefully.

3.4 Measuring Bit Effectiveness for ML-Based Distinguishers

As detailed in Section 3.3, the entirety of the δ bits is not mandatory for the development of an ML-based distinguisher. We have successfully trained a partial differential ML-based distinguisher utilizing a reduced number of bits while maintaining performance on par with full-bit distinguishers. The subsequent section aims to delineate a heuristic approach for identifying the minimal yet most informative set of bits for machine training. Given that block cipher design aspires to minimize correlations among output bits, there lacks a straightforward method to ascertain the most beneficial bits for the training of a partial distinguisher. To this end, we propose an experimental technique to find the output difference bits of δ for 6-round SPECK32/64 that are pivotal to the efficiency of a partial differential ML-based distinguisher. The primary objective of this experiment is to evaluate and score each bit of δ, thereby utilizing these scores to select the most influential bits for optimizing the training of a PD-ML-based distinguisher.

3.4.1 Methodology

The configuration of the experiment is structured in the following manner. Let us consider the set of bit positions within a block, denoted as $B = \{b_0, \ldots, b_{31}\}$, with b_0 representing the most significant bit position. We select a uniform subset $\mathcal{C}_B^{16} = \{\mathcal{C}_0, \ldots, \mathcal{C}_{99}\}$ from all possible combinations of 16 elements from B. This subset selection ensures each $b_i \in B$ appears uniformly across the combinations, where $0 \leq i \leq 99$. Each combination in this subset, comprising 16 bits, will inform the training of a PD-ML-based distinguisher for a SPECK32/64 block.

The choices made in this experiment are again motivated by computational feasibility and fairness of comparison. An exhaustive evaluation of all subsets of output bits is infeasible, so we approximate this search space by uniformly sampling 100 combinations. This number is large enough to expose variation across bit choices and to give repeated coverage of all positions, while remaining small enough that each sampled subset can be trained using the same architecture and dataset budget. The choice of 16-bit subsets provides a middle ground between very small feature sets, which may be too sparse to capture

useful interactions, and near-complete feature sets, which would make bit importance harder to distinguish. Using the same subset size for all combinations also ensures that the resulting scores reflect which bits are selected, rather than trivial differences caused only by the number of features.

To isolate the effect of bit choice, the underlying neural architecture and optimisation strategy were kept fixed throughout this stage and matched the configuration introduced in Section 3.3.1. In this way, changes in accuracy can be interpreted primarily as evidence about the usefulness of particular output-difference bits, rather than as a consequence of changing model capacity or training dynamics.

In the experimental process outlined in Algorithm 9, we proceed as follows: For each 16-bit combination $\mathcal{C}_i$ from $\mathcal{C}_B^{16}$, a PD-ML-based distinguisher is trained, and its accuracy is logged into an array $A = [a_0, \ldots, a_{99}]$. Subsequently, a matrix $M_{32\times100}$ is constructed where each row is associated with b_i and each column with $\mathcal{C}_j$. The construction of matrix M is governed by Equation 3.2.

$$M_{32\times100} = \begin{bmatrix} m_{0,0} & \ldots & m_{0,99} \\ \vdots & \ddots & \vdots \\ m_{31,0} & \ldots & m_{31,99} \end{bmatrix} \quad \text{and} \quad m_{ij} = \begin{cases} 0 & b_j \notin \mathcal{C}_i \\ 1 & b_j \in \mathcal{C}_i \end{cases} \tag{3.2}$$

This matrix encodes which bit positions are included in each combination. For example, if the most significant bit position b_0 is included in the combination $\mathcal{C}_0$, then $m_{0,0} = 1$; otherwise $m_{0,0} = 0$. Using M and the recorded accuracies, we compute a score for each bit, $S = \{s_0, \ldots, s_{31}\}$, as follows:

$$s_i = \frac{\sum_{j=0}^{99} m_{ij} * a_j}{\sum_{j=0}^{99} m_{ij}} \ . \tag{3.3}$$

The score for each bit, denoted as s_i, is determined by averaging the accuracies of all PD-ML distinguishers that incorporate the ith bit in their feature combination. Consequently, a higher s_i score suggests that including the corresponding bit tends to enhance the average accuracy of a PD-ML-based distin-

guisher. Leveraging these scores, we can identify and select the bits that most significantly contribute to the effectiveness of a PD-ML-based distinguisher's training.

Algorithm 9 Scores of effectiveness

Input: Sequence set of combinations: $\mathcal{C}_B^{16} = [\mathcal{C}_0, \ldots, \mathcal{C}_{99}]$
Output: Sequence set of scores: $S = [s_0, \ldots, s_{31}]$
Training Data: differences of ciphertext pairs of 6-round SPECK32/64: $\delta = [\delta_0, \ldots, \delta_{31}]$

Initialize sequence set A with size 100
Initialize $M_{32\times100}$ matrix
Initialize sequence set S with size 32
for $\mathcal{C}_i \in \mathcal{C}_B^{16}$ **do**
    `// Training PD-ML distinguishers`
    $D \leftarrow$ TRAINMACHINE$(\mathcal{C}_i)$
    $A[i] \leftarrow$ ACCURACYTEST(D)
    `// Making M matrix`
    for δ_j in δ **do**
        if $\delta_j \in \mathcal{C}_i$ **then**
            $M[j][i] \leftarrow 1$
        else
            $M[j][i] \leftarrow 0$
        end if
    end for
end for
`// Computing the score`
for $i \leftarrow 0$ **to** 31 **do**
    $S[i] \leftarrow \frac{\sum_{j=0}^{99} M[i][j] \times A[j]}{\sum_{j=0}^{99} M[i][j]}$
end for

3.4.2 Results and Discussion

By evaluating the efficacy scores across various bit combinations, we identify the most significant bits for optimising a PD-ML-based distinguisher while minimising feature-set complexity. If the efficacy scores for $\mathcal{C}_i \in \mathcal{C}_B^{16}$ were approximately uniform, that would suggest that all δ bits contribute equally to training efficacy. Our observations do not support that interpretation. Instead, certain bits—most notably the 12th, 14th, and 29th—consistently achieve higher scores than the rest. The significance of this result is that it provides direct evidence that not all output-difference bits contribute equally to the distinguisher. For the chapter's argument, this matters because it justifies feature selection as a meaningful cryptanalytic tool rather than as a purely engineering convenience.

To establish the consistency of our findings, given the randomness in selecting $\mathcal{C}_B^{16}$, we conducted three additional iterations of Algorithm 9, thereby obtaining four separate efficacy ratings per bit. These scores were then standardised against the mean of their respective experimental set. The graphical representation of this standardisation is illustrated in Figure 3.3. Additionally, Table 3.1 compiles the normalised mean and variability, as measured by the standard deviation, for the bits constituting δ across the four experiments. This additional repetition is important because it shows that the higher-ranked bits are not artefacts of a single random sample of combinations, but recur across independently generated subsets.

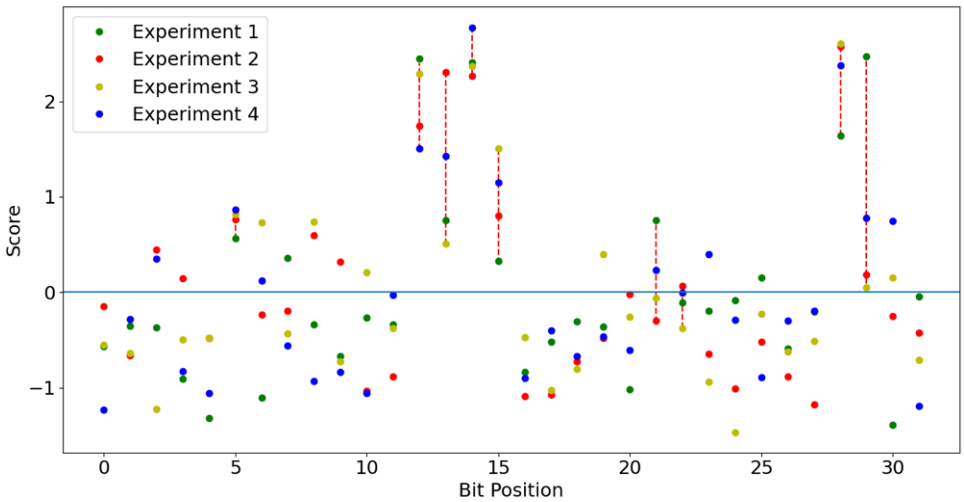


Figure 3.3: Normalized Scores for 4 different combination sets. The bits with vertical lines are the ones that are chosen as the most effective bits for training a PD-ML-based distinguisher for six rounds of SPECK32/64.

The results show that particular bits consistently achieve scores above the mean. With reference to Figure 3.3 and Table 3.1, we therefore selected eight high-scoring bits for further evaluation. For the 6-round PD-ML-based distinguisher targeting SPECK32/64, two sets of bits are considered: $\delta^T = [\delta_{29}, \delta_{28}, \delta_{22}, \delta_{15}, \delta_{14}, \delta_{13}, \delta_{12}, \delta_5]$ and $\delta^{T'} = [\delta_{29}, \delta_{28}, \delta_{21}, \delta_{15}, \delta_{14}, \delta_{13}, \delta_{12}, \delta_5]$. These bits are chosen primarily because they repeatedly perform above the zero-baseline average score, with the exception of $\delta_{22} \in \delta^T$ and $\delta_{21} \in \delta^{T'}$. Despite δ_{22} exhibiting a minimal standard deviation, its mean score is closely aligned with the aggregate average. Conversely, δ_{21} has a stronger mean score than many other candidates, although its performance fluctuates more across experiments than δ_{22}.

This distinction is relevant because it shows that the scoring method is not merely producing a rank order, but also exposing a trade-off between average

usefulness and stability. In other words, bit selection in the PD-ML setting is not only about choosing the highest mean score; it also involves deciding whether one prefers a bit that is consistently moderate or a bit that is occasionally stronger but more variable. Making that trade-off explicit strengthens the argument that the proposed scoring method is informative for practical feature selection.

Table 3.1: The average score (s_i) and standard deviation (SD_i) of each bit of δ in the experiments.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	-0.62	-0.48	-0.19	-0.51	-0.83	0.75	-0.12	-0.20	0.01	-0.47	-0.53	-0.40	1.99	1.25	2.45	0.94
SD_i	0.38	0.16	0.66	0.41	0.36	0.11	0.66	0.35	0.68	0.46	0.53	0.30	0.38	0.69	0.18	0.43
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
s_i	-0.82	-0.75	-0.62	-0.22	-0.47	0.15	-0.10	-0.34	-0.71	-0.36	-0.59	-0.52	2.29	0.87	-0.18	-0.59
SD_i	0.22	0.29	0.19	0.36	0.37	0.39	0.16	0.50	0.55	0.38	0.20	0.39	0.39	0.96	0.78	0.41

Subsequently, a new distinguisher is developed using only the selected eight bits to test whether they are sufficient for neural-distinguisher training. This yields a PD-ML-based distinguisher with an accuracy of 68.7% when trained on δ^T and 68.8% on $\delta^{T'}$. By contrast, a full-feature distinguisher trained on all 32 bits reaches 72% accuracy. For comparison, an additional PD-ML-based distinguisher is trained on the eight lowest-scoring bits. The accuracy of this model is 52%, only marginally above random guessing, which indicates that it is not an effective distinguisher. The training progression for these PD-ML distinguishers is shown in Figure 3.4.

From a cryptanalytic perspective, the comparison between 68.8% and 72% is meaningful because the 8-bit model preserves a clear non-random advantage while using only one quarter of the original input representation. This is exactly the kind of trade-off the chapter seeks: most of the distinguishing utility is retained, while the feature space is reduced substantially. The 52% result for the lowest-scoring 8-bit model is equally important, because it shows that the gain does not come from arbitrary compression alone. Rather, the model remains useful only when the retained bits preserve informative structure. This directly supports the claim that the proposed scoring method is selecting genuinely relevant signal rather than simply shrinking the network.

To test whether this interpretation is stable across training randomness, we repeated the main comparison over 10 random seeds. The full 32-bit model achieved a mean test accuracy of $71.6\% \pm 0.8$, the selected 8-bit PD-ML model achieved $68.1\% \pm 0.9$, and the lowest-scoring 8-bit model achieved $51.7\% \pm 0.7$. This rerun study is important because it shows that the ordering of the models is stable and that the gap between the selected and lowest-scoring bit sets is

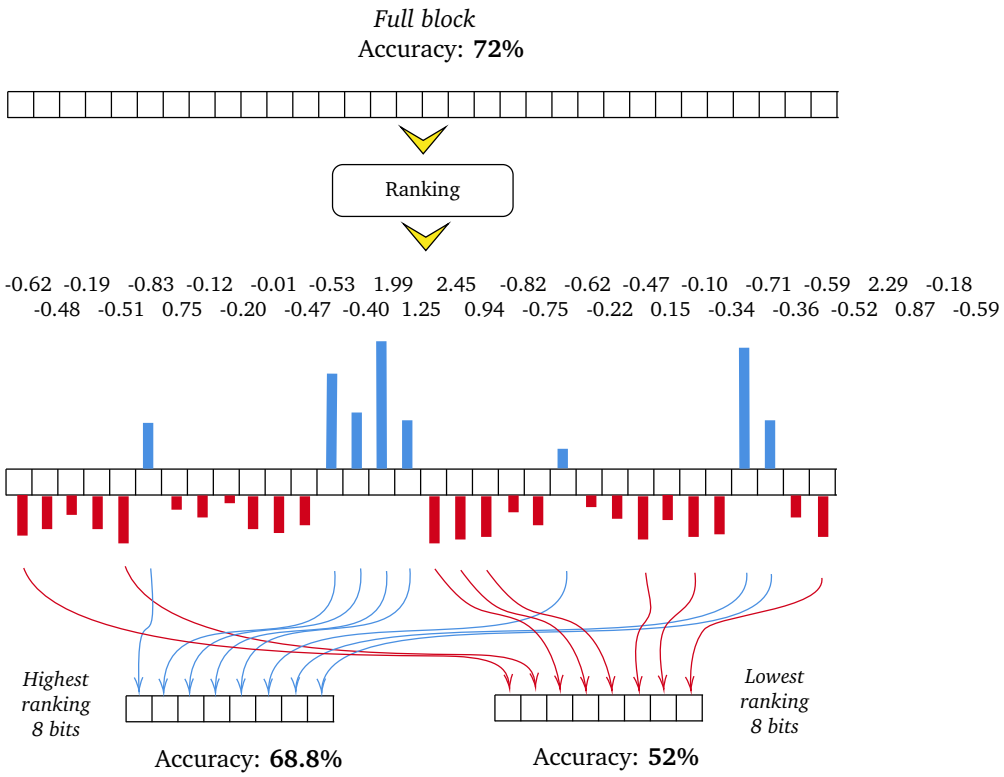


Figure 3.4: The Training Process of PD-ML distinguishers. Rank values are as per Table 3.1. Plotted bars are approximation of the values (due to graphical constraints).

not an artefact of a single training run. In other words, the claim that not all output-difference bits contribute equally is supported not only by the one-shot experiments, but also by repeated measurements with low variance.

Accordingly, the trade-off studied here should not be interpreted as efficiency at any cost. It is a selective reduction in complexity that remains useful because the distinguisher still preserves enough accuracy to support subsequent analysis. The results therefore show both the promise and the limitation of the PD-ML approach: feature reduction can work well, but only when guided by an informed bit-selection strategy. Within the evidence presented here, this conclusion is well supported for 6-round SPECK32/64; extending it to other ciphers remains future work.

These results support the usefulness of the proposed partial differential ML-based distinguisher in the SPECK32/64 setting studied here. Our bit-selection mechanism enables the PD-ML-based distinguisher to be trained with only a quarter of the bits used by the full-state baseline. This reduction in input dimensionality lowers computational and memory requirements during training and yields a more compact network architecture. With fewer input features, the

computational burden during training is reduced.

The methodology detailed in Sections 3 and 4 aligns with the input differences used in Gohr's publication, but it may also be compatible with a more black-box strategy for bit selection. The framework put forth in [Goh19] describes a process for identifying an effective input difference, Δ, for a differential attack using ML-based distinguishers without relying on pre-existing analytical knowledge. In principle, such a workflow could be combined with our bit-selection strategy, although this possibility is not evaluated in the present thesis.

It is important to recognize a key difference when contrasting our findings with those of Gohr. As noted in [BGPT21], Gohr's classifier distinguishes more than just the pure differential output distribution—it is also related to differences from the rounds preceding the last. This is evidenced when the input to the ML-based distinguisher is shifted from $C_0 \oplus C_1$ to the concatenated form $(C_0||C_1)$, illustrating a broader understanding that includes differential distributions from earlier encryption stages. This assertion is supported by the observations in [BGPT21], where altering the input format resulted in a decrease in accuracy, thereby implying that Gohr's model extracts a richer set of information than solely differential output distribution. Our research is concentrated on evaluating the influence of each δ bit on the training efficacy of the distinguisher. In our comparisons, we specifically refer to models trained on purely differential inputs, which for Speck32/64, is reported to be 75.4% accurate [BGPT21].

To place our PD-ML distinguisher in context, Table 3.2 compares it with representative machine-learning-based differential distinguishers from the literature. This comparison has two purposes. First, it provides a controlled comparison of training cost: the per-epoch training times reported in the table were obtained by rerunning the prior architectures and our own models in a common Google Colab environment. Second, it provides an accuracy comparison, but that comparison must be interpreted carefully. Gohr's model uses the concatenated ciphertext pair $(C_0||C_1)$ as input, whereas Benamira et al. and our models use the ciphertext difference $C_0 \oplus C_1$. Moreover, the model of Baksi et al. is not a binary classifier but a 32-class classifier, so its reported accuracy is not directly comparable to the binary accuracies in the other rows. Consequently, the fairest direct comparison in Table 3.2 is between Benamira et al. and our 32-bit model, since both are binary classifiers operating on $C_0 \oplus C_1$. Our 8-bit PD-ML row should therefore be read as a cost–accuracy trade-off relative to our own

32-bit baseline rather than as a direct replacement for every prior architecture.

Table 3.2: Comparison of different machine learning based differential distinguishers. The training time refers to the networks running on the Google Colab platform [Col]

Reference	Network	Number of input features	Depth	Epochs	Input of the Machine	Training time per epoch	Accuracy
[Goh19]	CNN	32 (bits)	10	200	$C_0 \| \| C_1$	$\approx$ 7 min	78.8%
[BGPT21]	CNN	32 (bits)	10	200	$C_0 \oplus C_1$	$\approx$ 7 min	75.4%
[BBDH21]	MLP	32 (bits)	6	10	$C_0 \oplus C_1$	$\approx$ 2.5 min	7.5%[1]
This work	MLP	32 (bits)	3	10	$C_0 \oplus C_1$	$\approx$ 1 min	72%
This work	MLP	8 (bits)	3	10	$C_0 \oplus C_1$	$\approx$ 30 sec	68.8%

[1] This machine is not a binary classifier; rather, the training set includes 32 distinct classes. More information can be found at [BBDH21].

Table 3.2 shows that, under a common runtime environment, our models require less training time per epoch than the deeper architectures considered in prior work. This reduction is expected, since our models use a shallower MLP and, in the PD-ML setting, a substantially smaller input space. With respect to accuracy, the most meaningful direct comparison is with Benamira et al., because both settings use binary classification on $C_0 \oplus C_1$. In that comparison, our 32-bit model attains 72% accuracy versus 75.4% for Benamira et al., while reducing the per-epoch training time from approximately 7 minutes to approximately 1 minute. The 8-bit PD-ML model further reduces the per-epoch training time to approximately 30 seconds while retaining 68.8% accuracy in the original run, showing that much of the distinguishing information can be preserved after feature reduction. The repeated-seed study supports the same conclusion: the corresponding mean accuracies are $71.6\% \pm 0.8$ for the 32-bit model and $68.1\% \pm 0.9$ for the selected 8-bit model, so the lower-cost operating point remains stable across reruns.

The relevance of Table 3.2 is therefore not that the proposed models achieve the highest raw accuracy in the literature. Rather, it is that they occupy a different and practically useful operating point. Compared with deeper baselines, the proposed models sacrifice some accuracy but substantially reduce training cost, and the 8-bit PD-ML model pushes this trade-off further while remaining clearly better than random. For the aims of this chapter, that is the more important result: the proposed method is not designed as a benchmark leader, but as a lower-cost distinguisher that remains useful enough to support downstream cryptanalytic procedures. Within the scope of the experiments reported here,

the evidence is sufficient to support that cost–accuracy claim on SPECK32/64, because both the one-shot experiments and the repeated-seed reruns preserve the same qualitative ordering.

3.4.3 Reducing the Time Complexity of *Key Averaging*

This study primarily investigates the application of the ML-based distinguisher to the first six rounds of SPECK32/64, in line with existing literature. It also provides evidence that the PD-ML-based distinguisher can reduce the complexity of extending the analysis to later rounds in this setting. This includes the *key averaging* technique detailed in [Goh19], traditionally employed to extend an r-round differential ML-based distinguisher to an $r+1$ round configuration. [Goh19] demonstrated this by leveraging a six-round neural distinguisher to analyze one round of partial decryption for each ciphertext pair in their dataset, using the collective data to calculate scores by averaging. This process, when applied to Gohr's distinguisher, carried a complexity of 2^{16} and yielded a distinguisher for seven rounds at an accuracy rate of 61%.

In contrast, implementing the same key-averaging technique with our PD-ML-based distinguisher yields a lower complexity of 2^{11} while achieving a 7-round distinguisher with an accuracy of 58% in the original experiment. In the SPECK32/64 setting considered here, this is a substantial reduction in complexity with a modest reduction in accuracy. Expressed differently, the complexity drops by a factor of $2^5 = 32$, while the accuracy decreases by only about three percentage points. To check the stability of this trade-off, we repeated the key-averaging experiment over 10 random seeds. The full-state baseline produced a mean accuracy of $61.2\% \pm 0.6$, whereas the PD-ML variant produced $58.4\% \pm 0.7$. Although extending the analysis to additional rounds is beyond the scope of this study, refining key averaging and related methods for PD-ML distinguishers is a natural direction for future work. Table 3.3 contrasts the time complexity and accuracy of the 7-round ML-based distinguisher derived from our model with Gohr's original model.

The relevance of this trade-off becomes even clearer in the key-averaging setting, where the main bottleneck is not training alone but the number of key guesses required to extend the attack to later rounds. In this context, a drop from about 61.2% to 58.4% is acceptable if it lowers the complexity from 2^{16} to 2^{11}, because the resulting distinguisher remains informative while the extension becomes substantially cheaper. The repeated-seed results strengthen this claim

by showing that the gap is stable and small relative to the 32-fold complexity reduction. This is the point at which the chapter's efficiency claim becomes cryptanalytically meaningful: the contribution is not only that the model trains faster, but that it can make later attack stages more practical. Efficiency in this chapter should therefore be understood as a property of cryptanalytic usefulness rather than as a purely engineering objective. Within the scope of the present chapter, the evidence is sufficient to support this conclusion for the SPECK32/64 key-averaging setting.

Table 3.3: Evaluation of the performance of ML-based distinguishers in executing the key averaging algorithm.

Reference	Time Complexity	Accuracy of resultant distinguisher
[Goh19]	2^{16}	61%
This work	2^{11}	58%

The same principles apply when an attacker targets the final round of a cipher using key ranking. In this scenario, an attacker utilizes an r-round ML-based distinguisher for the $(r+1)$-round of the cipher by determining all necessary subkeys from the preceding round. Utilizing Gohr's ML-based distinguisher for this type of key ranking attack typically requires the guessing of 2^{16} subkeys. However, in our experimental setting, the PD-ML-based model reduces the time complexity to 2^{11}. This reduction is achieved because only 8 bits from the previous round need to be decrypted, which can be done with an 11-bit key guess.

More generally, the model provides a way to trade off time complexity against distinguisher accuracy by adjusting the number of input bits according to their scores. This observation is relevant in settings where last-round key guessing is the main bottleneck, because it shows that feature selection can influence not only classifier size but also the cost of the surrounding attack procedure. The limitation is that this conclusion has been demonstrated only for SPECK32/64 in the present thesis. Whether the same trade-off remains useful for SPN-like ciphers or other architectures is left for future work.

3.5 Conclusion

This chapter investigated training neural differential distinguishers on partial output differences and introduced the PD-ML distinguisher. Unlike full-state

ML distinguishers, the partial differential ML-based distinguisher operates on a selected subset of bits. Through multiple independent experiments, we observed that some bits contribute more to accuracy than others, and we used that observation to develop a methodology for scoring the contribution of each bit within the block.

The main result of the chapter is not simply that a smaller model can be trained, but that useful distinguishing signal can be retained after substantial feature reduction. In the 6-round SPECK32/64 setting studied here, PD-ML distinguishers reached accuracy levels close to those of distinguishers trained on the complete 32-bit block, while also showing that increasing the bit count does not necessarily improve accuracy. This is relevant because it indicates that the informative structure learned by the model is concentrated unevenly across the output-difference vector.

To identify the most influential bits during training, we devised an algorithm that assigns a score to each bit of δ for a 6-round application on SPECK32/64. Using these scores, we selected the top eight bits to develop a 6-round PD-ML-based distinguisher for SPECK32/64, achieving accuracies of 68.7% and 68.8% in the original single-run experiments for the two selected 8-bit sets. This is close to the 72% accuracy attained by a full 32-bit ML-based distinguisher, while requiring a smaller input representation and lower training cost. The contrasting 52% result obtained from the lowest-scoring 8-bit subset further shows that the gain depends on selecting informative bits rather than on reducing dimensionality alone. Importantly, the repeated-seed study confirms the same pattern: over 10 random seeds, the full model reached $71.6\% \pm 0.8$, the selected 8-bit model reached $68.1\% \pm 0.9$, and the lowest-scoring 8-bit model reached $51.7\% \pm 0.7$.

This comparison highlights the central point of the chapter: the objective is not to maximise benchmark accuracy at any cost, but to obtain a better balance between accuracy and complexity. In the SPECK32/64 setting considered here, the decrease in accuracy is small enough that the distinguisher still captures meaningful non-random structure, whereas the reduction in dimensionality materially improves practicality. For that reason, the efficiency gain is not incidental; it is part of what makes the distinguisher useful in a cryptanalytic setting. The rerun evidence is important here because it shows that the claim is not resting on one favourable seed; the same ordering persists with low variance across repeated trials.

This streamlined input configuration also decreases the complexity of subse-

quent attacks in the SPECK32/64 setting considered here, because it reduces the number of last-round key bits that must be guessed. For example, when our model is used within Gohr's key-averaging algorithm on SPECK32/64, the complexity decreases from 2^{16} to 2^{11}, a 32-fold reduction. In the repeated-seed study, this reduction is accompanied by a change in mean accuracy from $61.2\% \pm 0.6$ to $58.4\% \pm 0.7$. This is important because it shows that feature selection can improve not only training efficiency but also the practicality of later cryptanalytic stages, and that this conclusion remains stable across reruns.

Although these experiments were conducted on the lightweight cipher SPECK32/64, the underlying idea of scoring and selecting informative bits is not tied to a single architecture. However, the present thesis does not provide experimental validation on other ciphers, so the transferability of the approach should be treated as a hypothesis for future work rather than as a demonstrated result. More broadly, the findings suggest that feature selection may be a useful direction for improving the efficiency of ML-based differential cryptanalysis.

Looking ahead, further exploration of PD-ML distinguishers may clarify which factors most strongly influence bit effectiveness and whether such patterns relate to properties of cipher round functions. Comparative analysis of our scoring method with machine-learning interpretation techniques such as SHAP or LIME could also provide additional insight into the relationship between individual bits and model predictions. In addition, future work could study how r-round distinguishers can be used to derive distinguishers for $r+1$ or more rounds. As indicated in Section 3.4.2, PD-ML distinguishers may help streamline such extensions, but this requires further experimental validation.

Chapter 4

Deep Learning-Based Rotational-XOR Distinguishers for AND-RX Block Ciphers

Early work on deep-learning-assisted cryptanalysis focused primarily on differential distinguishers. This chapter extends these ideas to Rotational-XOR (RX) cryptanalysis for AND-RX block ciphers. In particular, we study Simon and Simeck, which are representative lightweight designs based on bitwise XOR, bitwise AND, and rotations.

We develop deep-learning-based RX distinguishers and introduce a search procedure, based on an evolutionary algorithm, to tune the RX-pair parameters: the translation (δ) and the rotation offset (γ). These parameters strongly influence the quality of RX distinguishers, yet are difficult to select by hand. Beyond reporting distinguishers for multiple Simon and Simeck variants, we compare the resulting RX distinguishers with prior related-key and weak-key analyses and investigate how diffusion-layer parameters affect resistance to deep-learning-based RX attacks. The goal is to improve the practical evaluation of existing AND-RX ciphers and to provide design intuition for future constructions.

The results presented in this chapter were published in the proceedings of the 30th International Conference on Selected Areas in Cryptography (SAC 2023). [EGP23]

4.1 Introduction

Recent work has shown that deep learning can be used to construct effective statistical distinguishers in differential cryptanalysis. A natural next step is to ask whether similar techniques can strengthen Rotational-XOR (RX) cryptanalysis, particularly for lightweight AND-RX ciphers such as Simon and Simeck.

Compared to differential cryptanalysis, RX cryptanalysis has received less attention, and AND-RX structures introduce additional complications due to the interaction of rotations with the non-linearity of the AND operation. As a result, much of the prior literature studies RX behaviour under restrictive conditions (for example, weak-key or related-key settings), which can limit what can be concluded about security across the full key space.

In this chapter, we incorporate deep learning (DL) into RX cryptanalysis by training neural distinguishers that separate RX-related encryptions from random behaviour. We consider both the related-key scenario and training across the full key space, enabling a broader assessment of Simon and Simeck beyond weak-key constraints.

A practical challenge in RX cryptanalysis is choosing effective RX-pair parameters, notably the translation (δ) and the rotation offset (γ). To address this, we employ an adaptation of the evolutionary-search procedure from [BGHR22] to identify RX inputs and shift parameters that maximise the effectiveness of round distinguishers when paired with DL classifiers. We also apply the optimiser to refine rotation settings in diffusion layers for AND-RX ciphers that retain the Simeck round-function form while varying the rotation parameters, aiming to reduce the effectiveness of DL-based distinguishers in our evaluation setting.

In this setting, efficiency matters because RX cryptanalysis is used not only to produce a final distinguisher, but also to explore a large parameter space. The practical cost therefore includes the search for suitable RX inputs, the tuning of rotation offsets, repeated model training, and comparison across cipher variants. A method that is slightly weaker in raw classification accuracy can still be more useful if it enables broader exploration of the RX space, produces informative evaluation signals at lower search cost, or supports analysis beyond narrow weak-key settings. For this reason, the goal of this chapter is not to maximise accuracy in isolation, but to improve the balance between distinguisher quality and the cost of discovering and evaluating useful RX configurations.

Empirically, the resulting DL-based distinguishers achieve higher accuracies on Simeck than on Simon in our experiments. Table 4.1 compares our related-

key DL-based RX distinguishers with prior related-key DL distinguishers for Simeck. Notably, our model is trained using a single pair, whereas earlier work typically trains on eight pairs; for comparison we also report the Combined Accuracy Score from [GLN22] computed over eight pairs. Table 4.2 further compares our DL-based RX distinguishers with previous RX distinguishers for Simeck.

Finally, we report a related-key DL distinguisher for Simeck 64/128 and a DL-based RX distinguisher trained across the full key space. This enables a more direct examination of how key variation affects distinguisher accuracy and provides additional insight into the practical strength of these AND-RX ciphers.

Table 4.1: Comparison of related-key DL-based distinguishers for Simeck. RX: Rotational-XOR cryptanalysis, RD: Related-key Differential cryptanalysis. The Combined Accuracy Score [GLN22] for m pairs is $\frac{1}{1+\prod_{i=1}^{m}\frac{1-p_i}{p_i}}$

	Round	Combined Accuracy Score	Pairs	Attack Type	Ref.
Simeck 32/64	13	0.9950	8	RD	[LLS+22]
	14	0.6679	8	RD	[LLS+22]
	15	0.5573	8	RD	[LLS+22]
	15	**0.5134**	**1**	**RX**	**This Work**
	15	**0.5475**	**8**	**RX**	**This Work**
Simeck 64/128	18	0.9066	8	RD	[LLS+22]
	19	0.7558	8	RD	[LLS+22]
	20	0.6229	8	RD	[LLS+22]
	20	**0.5212**	**1**	**RX**	**This Work**
	20	**0.6338**	**8**	**RX**	**This Work**

Table 4.2: Comparison of the RX distinguishers for different versions of Simeck

Cipher	Rounds	Data Complexity	Size of Weak Key Class	DL-based	Ref.
Simeck32/64	**15**	$\mathbf{2^{20}}$	**Full**	**Yes**	**This Work**
	15	2^{18}	2^{44}	No	[LLA+22]
	19	2^{24}	2^{30}	No	[LLA+22]
	20	2^{26}	2^{30}	No	[LLA+22]
Simeck48/96	**17**	$\mathbf{2^{20}}$	**Full**	**Yes**	**This Work**
	16	2^{18}	2^{68}	No	[LLA+22]
	18	2^{22}	2^{66}	No	[LLA+22]
	19	2^{24}	2^{62}	No	[LLA+22]
	27	2^{44}	2^{46}	No	[LLA+22]
Simeck64/128	**20**	$\mathbf{2^{20}}$	**Full**	**Yes**	**This Work**
	25	2^{34}	2^{80}	No	[LLA+22]
	34	2^{56}	2^{58}	No	[LLA+22]

4.1.1 Our Contribution

In this chapter, we detail several key advances in the Rotational-XOR cryptanalysis of AND-RX block ciphers like Simon and Simeck, particularly emphasizing

the integration of deep learning techniques. Here are our principal contributions:

1. To the best of our knowledge, this is the first study to incorporate neural networks into RX cryptanalysis. Rather than introducing a completely new evolutionary algorithm, we adapt the search procedure of Bellini et al. [BGHR22] to the RX setting considered in this chapter. The adaptation has three concrete parts: (i) each candidate encodes both the XOR translation (δ) and the rotational offset (γ), rather than keeping γ fixed; (ii) candidate quality is evaluated using the approximate RX bias score and the longest round coverage achieved under the evaluation criterion defined in this chapter; and (iii) the search returns high-performing (δ, γ) pairs for training DL-based RX distinguishers on Simon and Simeck. This adapted search led to the discovery of new RX distinguishers for Simon and Simeck ciphers.
2. Our research identifies effective RX distinguishers for various versions of Simon and Simeck using deep learning methods within the related-key scenario, moving beyond the conventional weak-key model.
3. We also analyse the role of the diffusion layer in AND-RX ciphers instantiated with the Simeck round-function form, identifying parameter settings that reduce distinguisher effectiveness in our experiments. These results may help inform the design of AND-RX block ciphers with improved resistance to the specific RX analyses considered here.

4.1.2 Outline

This chapter is organised as follows. Section 4.2 reviews background on AND-RX ciphers and existing weak-key RX attacks. Section 4.3 describes the methodology, including the adaptation of Bellini et al.'s evolutionary search to the RX setting studied here. Section 4.4 presents new DL-based RX distinguishers for Simon and Simeck. Section 4.5 studies diffusion-layer parameters and proposes techniques for selecting rotation settings that improve resistance to DL-based RX attacks. Section 4.6 concludes the chapter.

4.2 Preliminaries

In this section, we introduce the concepts and notation used throughout this chapter, focusing on AND-RX ciphers and Rotational-XOR (RX) cryptanalysis.

These preliminaries support the methodology and results presented later.

4.2.1 AND-RX Ciphers

Simon and Simeck represent examples of block ciphers designed for environments where resources are constrained, such as in Internet of Things (IoT) devices. These ciphers are based on the AND-RX design paradigm, which incorporates three fundamental operations: bitwise XOR ($\oplus$), bitwise AND ($\wedge$), and left circular shift ($\lll i$) by i bits. The typical round function, R, of an AND-RX cipher can be described by the equation:

$$R(x, y) = (y \oplus f(x) \oplus k, x),$$

where $f(x) = ((x \lll a) \wedge (x \lll b)) \oplus x \lll c$ and k is the subkey for the corresponding round.

Introduced in 2013 by the National Security Agency (NSA), Simon is a family of lightweight block ciphers specifically designed for efficient implementation in environments with limited computational resources [BSS+13]. Each variant within the Simon family is defined by a word size of n bits, and is represented as Simon$2n$ where $n \in \{16, 24, 32, 48\}$.

Each Simon cipher variant also specifies a key size of m words, resulting in total key sizes of mn bits. These configurations are denoted as Simon$2n/mn$. For instance, Simon$32/64$ is configured to operate with 32-bit plaintext blocks and utilizes a 64-bit key. In this chapter, our focus will primarily be on the variants formatted as Simon$2n/4n$, analysing their structural properties and susceptibility to cryptanalysis. The $f(x)$ function for Simon$2n$ encryption is $f(x) = ((x \lll 1) \wedge (x \lll 8)) \oplus x \lll 2$

Simon's key schedule produces r key words $k_0, \ldots, k_{r-1}$ from a given key, where r is the number of rounds. This process also involves using a sequence of 1-bit round constants to remove slide properties and circular shift symmetries.

Simeck, another lightweight block cipher developed for environments with constrained computational resources, was introduced in [YZS+15]. Similar to Simon, Simeck is represented by Simeck$2n/mn$, where n can be 16, 24, or 32, $2n$ is the block size, and mn represents the key size.

While Simeck employs a round function R similar to that of Simon, it distin-

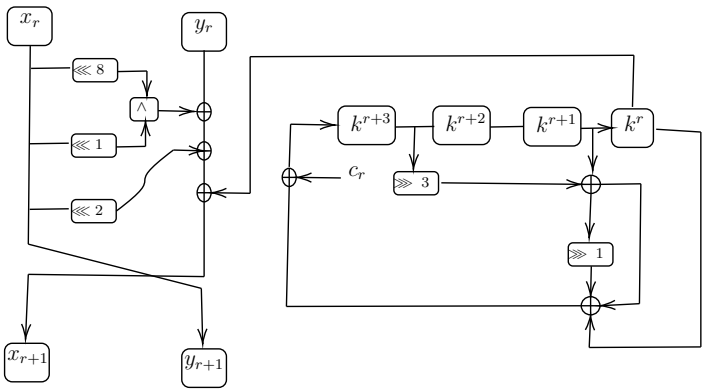


Figure 4.1: Simon cipher for $m = 4$

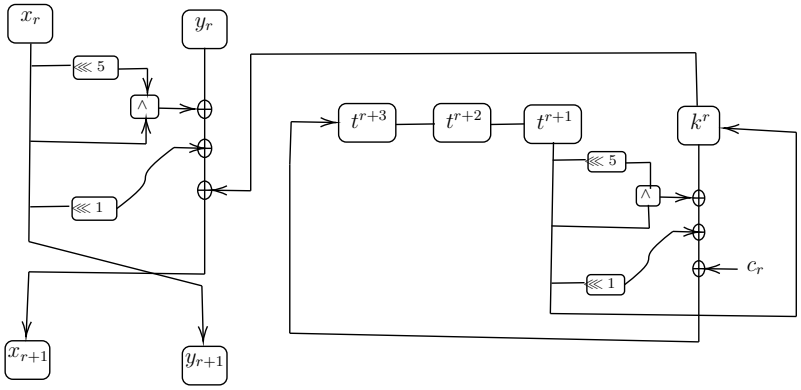


Figure 4.2: Simeck cipher

guishes itself with a unique function f, defined as:

$$f(x) = (x \wedge (x \lll 5)) \oplus (x \lll 1).$$

This function is critical to the cipher's operation and contributes to its distinct cryptographic characteristics.

In the Simeck cipher, round keys k_i are generated from a master key K, which is initially divided into four words. These words set the initial states (t_2, t_1, t_0, k_0) of a feedback shift register. The cipher uses the function f along with a 1-bit round constant c_r to produce round keys and update the register states. The number of rounds r in Simeck variants like Simeck$32/64$, Simeck$48/96$, and Simeck$64/128$ are 32, 36, and 44, respectively. Figure 4.1 and Figure 4.2 illustrate the round function and key schedule of both Simon and Simeck ciphers, respectively.

4.2.2 Weak Key Model Attacks

Weak-key attacks are a specific type of cryptanalytic approach in which certain keys make the cipher easier to analyse than under generic-key conditions. These attacks exploit the properties of these weak keys to breach the cipher's security more easily than would be possible with other keys. This vulnerability often arises from the inherent structure of the cipher or from specific key values that cause the encryption process to behave in a predictable or degenerate manner. RX cryptanalysis is a form of weak-key attack that utilizes rotational symmetries combined with XOR operations to analyse the vulnerabilities in block ciphers.

Conventional RX attacks on Simon and Simeck ciphers exploit weaknesses in their diffusion layers when certain key configurations are used. These configurations allow the attacker to predict the behavior of the cipher under rotational transformations, making it feasible to decipher keys or plaintexts with fewer resources than typically required for brute-force guessing. One of the key studies on RX attacks is [LLA+22], which is explained in more detail in Chapter 2. In that work, the authors conduct RX cryptanalysis on 20, 27, and 34 rounds of Simeck32/64, Simeck48/96, and Simeck64/128, respectively. However, the weak-key sizes used in these attacks are less than half of the full key size in all three cases, which makes the methods less practical and more restricted.

4.3 Identification of Optimal RX Distinguishers in Cryptanalysis with Evolutionary Algorithm

In this section, we describe how the evolutionary search procedure of Bellini et al. [BGHR22] is adapted to the RX setting studied in this chapter. The goal is not to propose a new class of evolutionary algorithm, but to make the existing search usable for identifying high-performing RX pairs for DL-based distinguishers.

Relative to the baseline procedure of Bellini et al., the search is modified in three specific ways. First, each search candidate jointly represents the XOR translation δ and the rotational offset γ, so the search is no longer restricted to a fixed rotational offset. Second, the fitness evaluation is changed to the RX setting: candidates are ranked using the approximate RX bias score together with the round-based evaluation criterion used in this chapter. Third, the output of the search is tailored to the training pipeline of this thesis, namely a list of high-performing (δ, γ) pairs that can be used to generate RX-related training

data for Simon and Simeck distinguishers.

Accordingly, the methodological contribution here is the adaptation of the search procedure to the RX setting and to the DL-based evaluation pipeline, rather than the introduction of a fundamentally new evolutionary algorithm. We begin by detailing the artificial-intelligence tools and the specific deep-learning framework employed in our research. We then examine how the RX bias score correlates with distinguisher performance, which motivates the adapted search procedure used later in the section.

Throughout this chapter, the term *optimal* is used in an empirical, thesis-specific sense. A parameter choice (for example, δ, γ, or a rotation set (a, b, c)) is called *optimal* when it maximizes the evaluation objective used in our search procedure within the explored search space and under the computational budget of our experiments. In practice, this objective is defined by the longest round coverage achieved by the distinguisher and, when needed to break ties, by the associated bias score or classifier accuracy. Accordingly, *optimal* in this chapter means *best found by the evolutionary search under the stated objective and search budget*; it does not claim a proof of global optimality over all possible parameter choices.

4.3.1 AI Tools and Model Development

In this research, we utilize the Keras library [Cho15] to develop a deep learning model based on the architecture introduced by Aron Gohr at CRYPTO'19 [Goh19]. Gohr's model, a neural network distinguisher, is designed to identify differences between pairs of Speck32/64 ciphertexts, distinguishing between fixed (non-random) and random message pairs using a residual network structure. This model achieves high accuracy across various rounds of Speck32/64, enabling effective key recovery attacks.

Gohr's distinguisher processes a 64-bit ciphertext pair from Speck32/64, transforming it into a 16-bit wide tensor with four channels, reflecting the unique word structure of Speck32/64. The model's architecture includes a one-dimensional convolutional layer (Conv1D) with a kernel size of 1 and 32 filters, slicing through the four-channel input.

Subsequent to the convolution, the model applies batch normalization and ReLU activation. The architecture features several residual blocks, each with two Conv1D layers, kernel size 3, and 32 filters, which contribute to the depth of the neural distinguisher.

The final components of the model are a densely connected prediction head with ReLU activations and an output layer equipped with a single neuron and sigmoid activation. The network employs L2 regularization (value of 10^{-5}) to minimize overfitting and utilizes the Adam optimiser [KB14].

This study applies Gohr's architecture to train RX differential distinguishers for AND-RX ciphers, such as Simon and Simeck, noting their shared 16-bit structure with Speck32/64. The goal is to explore the capabilities of deep learning in identifying new RX distinguishers and enhancing the understanding of cipher security.

We adopted a sequential training strategy, gradually increasing the number of rounds to leverage insights from earlier iterations, which has been shown to boost model efficacy [BGHR22]. Our training involved 10^7 samples and 10^6 validation samples, with a batch size of 1000. The model, set to a depth of 1, underwent training over 5 epochs.

The architectural and training choices in this section were made with two objectives in mind: keeping the comparison with prior work meaningful, and keeping the repeated RX-search experiments computationally feasible. For this reason, we adopted Gohr's residual-style architecture as the starting point rather than performing a completely new architecture search. The choice is motivated by the structural similarity between the ciphertext representation used here and the 16-bit word-oriented representation used for Speck32/64: in both cases, the neural distinguisher receives a compact, structured bit-tensor input for which a lightweight convolutional residual network is a natural baseline. Reusing this architecture also helps isolate the effect of the RX setting itself, rather than conflating it with large changes in model design.

The final configuration was selected as a practical trade-off between validation performance, runtime, and experimental stability. In pilot experiments we did not observe gains from deeper variants that were large enough to justify the extra retraining cost during the repeated RX-search loops, so the residual network depth was fixed at 1. A batch size of 1000 gave the best overall throughput on the available hardware while keeping training sufficiently stable from run to run; smaller batches increased wall-clock time and gave slightly noisier validation curves, whereas larger batches did not provide consistent accuracy gains. Likewise, 5 training epochs were retained because they were sufficient to separate promising from unpromising RX pairs, while longer runs increased the cost of the search substantially without materially changing the ranking of candidate pairs. The use of 10^7 training samples and 10^6 validation samples was intended

to reduce variance in the estimated distinguisher accuracy, and the sequential round-by-round training strategy was kept because it provides a stable way to extend the analysis to harder settings without restarting the optimisation from scratch at every round.

4.3.2 Training DL-based RX distinguishers

This section outlines the training process for our RX distinguishers, which is based on deep learning. Furthermore, it elaborates on the potential of DL-based RX distinguishers in gaining insights from pairs of ciphertext.

4.3.2.1 Training phase

The training process for our RX distinguishers includes data preparation, model configuration, and subsequent evaluation on AND-RX ciphers like Simon and Simeck. Initially, we prepare the data by generating a dataset of RX ciphertext pairs, labeled as non-random (for fixed (δ, γ) pairs) or random (random message pairs). In RX cryptanalysis, which involves related-key analysis, translations δ in RX plaintext pairs may correspond with translations in the keys, denoted as δ_{key}. Research indicates that the effectiveness of RX differential propagation in AND-RX ciphers depends significantly on the Hamming weight and rotation of the differences [LLA+22].

For the initial training sessions, we focus on zero input differences for lightweight ciphers Simon32/64 and Simeck32/64. We set the input and key differences to hexadecimal values $0x0000$ and $0x00000000$ respectively, representing these as n-bit and $n \times m$-bit binary configurations for plaintexts and keys. This strategy serves to establish a baseline for the RX distinguisher, allowing us to examine its initial performance and potential enhancements for these ciphers.

We fix $\delta = 0$ for both plaintexts and keys, cycling through all possible γs to identify the one that maximises distinguisher efficacy over the longest number of rounds, r. Our findings show that for Simeck32/64, the optimal γ values in the sense defined above are $\{1, 15\}$, while for Simon32/64, they are $\{4, 12\}$, yielding the highest performance observed in our experiments over 14 and 10 rounds, respectively. The comprehensive training methodology for r rounds is outlined in Algorithm 10.

These design choices are again motivated by fairness of comparison and by computational cost. The initial experiments with $\delta = 0$ serve as a controlled

baseline that makes it possible to study the effect of the rotational offset γ before introducing the additional variability of non-zero translations. Cycling through all admissible γ values is feasible at this stage because the search space for γ alone is small, and it provides a concrete reference point for the later joint search over (δ, γ). Keeping the neural architecture and optimisation settings fixed across these baseline experiments ensures that changes in accuracy can be attributed primarily to RX-pair selection rather than to changes in model capacity or training dynamics.

Algorithm 10 DL-based RX Distinguisher for r rounds of a Cryptographic Primitive

Input: r (number of rounds), AI machine, (C_0, C_1)
Output: Trained AI machine, RX distinguisher status
Choose a rotational offset γ and constants δ_1, δ_2, with $\delta_\gamma = \delta_1 \oplus \delta_2$
Generate 10^7 plaintext pairs (P_0, P_1) with RX-difference $\delta_\gamma(P_0, P_1) = (L_1 \oplus (L_0 \lll \gamma), R_1 \oplus (R_0 \lll \gamma))$
Randomly allocate 10^7 labels $Y \in_r \{0, 1\}$ to the pairs
for each pair (P_0, P_1) with label Y **do**
    if $Y = 0$ **then**
        $P_1 \in_r \{0, 1\}^{32}$
    end if
    Encrypt the pairs with r rounds of the cryptographic primitive to get ciphertext pairs (C_0, C_1)
    Store (C_0, C_1) with corresponding labels in a dataset
end for
Train DL-RX-distinguisher using the dataset and their corresponding labels
Repeat steps 3-11 for another 10^6 pairs for testing
Measure the accuracy of the DL-based RX distinguisher
if accuracy $\geq 50\%$ **then**
    The machine is a DL-based RX distinguisher
end if

4.3.2.2 Machine Interpretation

In this part, we discuss the deep learning-based RX distinguishers developed for AND-RX ciphers, with a particular focus on Simeck32/64. Our goal is to understand the factors that influence the accuracy of these distinguishers and their effectiveness in RX cryptanalysis.

Previous research on deep learning-based differential distinguishers highlights the significant influence of output differential bias on their accuracy and the maximum number of cipher rounds they can effectively analyse. To further

explore this aspect within our context, we introduce the concept of the Approximate Bias Score for RX attacks, which is adapted from the definition provided in [BGHR22].

Definition 4.3.1 (Approximate RX Bias Score)**.** *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a block cipher, and let $\delta \in \mathbb{F}_2^n$ be an input RX-difference with a given rotational offset γ. The Approximate RX Bias Score for δ, denoted by $\tilde{b}^t(\delta, \gamma)$, is defined as the sum of the biases of each bit position j in the output RX-difference, computed over t samples. Formally, we have:*

$$\tilde{b}^t(\delta,\gamma) = \left| \sum_{j=0}^{n-1} 2 \cdot \frac{\sum_{i=0}^{t} ((E_{K_i}(X_i)) \oplus E_{(K_i \lll \gamma) \oplus \delta}((X_i \lll \gamma) \oplus \delta)))_j}{t} - 1 \right|$$

In our study, we investigated the relationship between bias score, accuracy of the RX distinguisher, and number of rounds for the trained distinguisher.

To investigate the impact of different γ values while maintaining a fixed $\delta = 0$, we trained various distinguishers and evaluated their performance. For each distinguisher, we computed the bias score and assessed its accuracy across multiple rounds. This evaluation was repeated through several iterations to confirm the robustness of our findings.

The results were visualized in a scatter plot (Figure 4.3), which plots bias scores on the x-axis against distinguisher accuracy on the y-axis. Each point on the plot represents a different γ value for Simeck32/64, with colors indicating the number of rounds: red for 11 rounds, green for 12 rounds, and blue for 13 rounds.

From the analysis of this plot, a general pattern emerged where higher bias scores typically indicated higher accuracy or a greater number of cipher rounds. An exception was noted at $\gamma = 11$, which showed the highest bias but did not support training beyond 12 rounds.

To deepen our understanding of these trends, we employed a weighted correlation analysis, confirming a positive relationship between bias scores and distinguisher accuracy. The analysis utilized formula (1) to calculate a correlation coefficient (ρ) of approximately 0.65, indicating a strong positive correlation. This correlation suggests that higher bias scores generally improve distinguisher accuracy. However, outliers like $\gamma = 11$ suggest potential variations in cipher structure or other factors that require further exploration.

$$\text{correlation_coefficient} = \rho\left(\tilde{b}^t, a \times e^{2r}\right) \tag{4.1}$$

These findings support our assertion that the output RX difference bias score is a crucial factor in determining the accuracy of deep learning-based RX distinguishers. The clear relationship between bias scores and distinguisher performance has also led us to develop an adapted evolutionary algorithm. This new algorithm is specifically designed to identify the most effective (δ, γ) pairs for RX plaintext, aiming to optimise the training and performance of RX distinguishers.

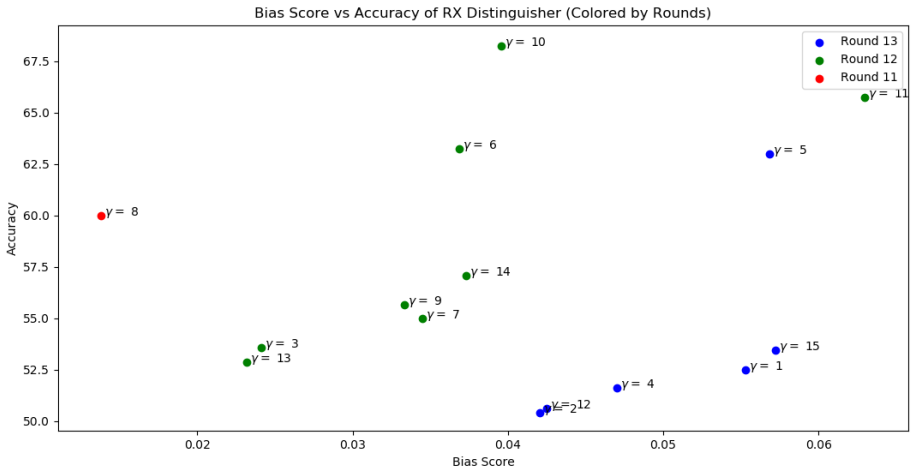


Figure 4.3: Scatter plot of Bias Score vs Accuracy of RX Distinguisher (Colored by Rounds)

4.3.3 Evolutionary Optimisation of Deep Learning RX Differential Distinguishers

We use an adapted evolutionary-search procedure to identify high-performing RX input differences. Compared with earlier search procedures, the adaptation uses the approximate RX bias score for RX pairs and treats the rotational offset γ and the XOR translation δ as joint search variables. This allows the search to explore candidate RX pairs that would be missed when γ is fixed in advance.

The main adaptation is that the search no longer treats the shift parameter γ as fixed. Instead, to the best of our knowledge, this is the first approach that searches jointly for best-found values of both δ and γ under a single evolutionary objective in the RX setting considered here. Earlier work typically searched

for a favourable δ while holding γ fixed (often $\gamma = 1$). In our representation, the candidate individual is extended so that it includes both the binary encoding of δ and a binary encoding of γ based on the block size. For example, in Simeck32/64, γ and γ_{key} are represented by two 4-bit words, adding 8 bits to each population member and allowing the search to mutate and recombine rotational offsets together with XOR translations.

The algorithm begins with a population of random input differences and rotational offsets. For each candidate pair (δ, γ), it computes an approximate RX bias score and retains the 32 highest-scoring candidates for subsequent analysis. The search then runs for 50 iterations, generating and evaluating new candidates by mutation and recombination. When a candidate exceeds the bias threshold used in this chapter, the round count is increased and the search continues at the harder setting. The output is therefore not a proof of optimality, but a ranked list of the best-found (δ, γ) pairs for each round up to round R, together with their cumulative RX bias scores.

The optimisation process, detailed in Algorithm 11, uses an initial population of 1024 random candidates, retains the best 32 candidates at each iteration, applies mutation with probability $p_m = 0.10$, estimates the RX bias with sample size $t = 2^{14}$, and uses the threshold $T = 0.03$ to decide when to increase the round count. Variables *curr_population*$_{i,\delta}$ and *curr_population*$_{i,\gamma}$ represent the value of the ith bit of δ and γ, respectively.

The values used in the search procedure were chosen to balance diversity, search depth, and total runtime. The initial pool of 1024 random candidates is intentionally much larger than the retained population so that the search begins with broad coverage of the joint (δ, γ) space. After scoring, only the top 32 candidates are retained for mutation and recombination. In our experiments, smaller retained populations made the search more sensitive to random initialisation, whereas substantially larger retained populations increased the recombination cost without yielding better final candidates often enough to justify the extra runtime. Similarly, 50 iterations were used as a fixed search budget: this was sufficient for progressive refinement of promising pairs, while noticeably longer runs did not improve the best-found candidates consistently enough to justify the additional cost on the available hardware.

The remaining evolutionary parameters were fixed to $p_m = 0.10$, $t = 2^{14}$, and $T = 0.03$ throughout the experiments. The mutation probability $p_m = 0.10$ gave the best exploration–exploitation balance in our runs: lower values made the search more likely to stagnate around early good candidates, while higher

values made the retained population too unstable from one iteration to the next. The sample size $t = 2^{14}$ was chosen because smaller values, such as 2^{10}, made the ranking of candidates noticeably noisier, whereas larger values, such as 2^{16}, were more expensive without clearly changing the ordering of candidates enough to justify the cost. The threshold $T = 0.03$ was used on the normalised RX-bias scale to decide when a candidate was strong enough to justify moving to the next round. Keeping these parameters fixed across experiments ensures that differences between ciphers and RX pairs are not confounded with changes in search budget. The overall objective is therefore not to claim an exhaustive optimiser, but to use a stable and computationally feasible search procedure that can repeatedly identify high-quality RX pairs under a consistent evaluation regime.

Algorithm 11 Evolutionary optimiser for RX differential distinguishers

init_population $\leftarrow$ [RandomInt$(0, 2^n - 1)$||RandomInt$(1, n - 1)$ for 1024 times]
Sort *init_population* by $\tilde{b}^t_{(\delta,\gamma)}(\cdot)$ in descending order
curr_population $\leftarrow$ first 32 elements of *init_population*
for *iter* $\leftarrow 1$ to 50 **do**
    cand $\leftarrow$ [ ]
    for $i \leftarrow 0$ to 31 **do**
        for $j \leftarrow i + 1$ to 31 **do**
            $m_\gamma \leftarrow 1$
            if RandomFloat$(0, 1) < 0.10$ **then**
                $m_\delta \leftarrow 1$
            else
                $m_\delta \leftarrow 0$
            end if
            Add $(((\textit{curr_population}_{i,\delta} \lll \text{RandomInt}(0, n - 1)) \oplus \textit{curr_population}_{j,\delta} \oplus m_\delta) \,||\, (\textit{curr_population}_{i,\gamma} \oplus m_\gamma)$ to *cand*
        end for
    end for
    Sort *cand* by $\tilde{b}_t(\cdot)$ in descending order
    curr_population $\leftarrow$ first 32 elements of *cand*
end forreturn *cand*

Algorithm 11 is tailored to the RX-search problem studied in this thesis for primitives such as Simon32/64 and Simeck32/64. By jointly encoding the rotational offset γ and the XOR translation δ, and by ranking candidates with the RX-bias-based objective defined above, the procedure explores a wider set of RX pairs than earlier fixed-γ searches. In our experiments, this adaptation helped identify effective RX distinguishers for full-key classes, extending earlier work

that focused on non-full-key classes.

4.4 Results and Discussion

This section reports the results obtained by applying our evolutionary optimisation approach to several variants of the Simon and Simeck block ciphers. For each cipher version, the objective is to identify effective RX input differences (δ) and rotational offsets (γ) under the evaluation criterion defined above, and then to assess what these choices reveal about the practicality of DL-based RX cryptanalysis.

Throughout this section, the important question is not only how many rounds a distinguisher reaches, but also what those results imply about the structure of the cipher and the usefulness of the search framework. In particular, we interpret the results in terms of three issues: whether the method transfers across cipher families, whether it remains informative outside narrow weak-key classes, and whether it offers design insight into which rotation settings make AND-RX constructions more or less resistant to the RX analyses considered here.

In this work, we trained the distinguishers using Colab [Bis19] and ran the evolutionary algorithm on an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz. About three hours were spent on each of the trials for Simon and Simeck's highest weight versions.

These running times make clear why efficiency is a relevant criterion in this chapter. Because the method is used as a search and evaluation framework, even modest reductions in the cost of exploring candidate (δ, γ) settings, cipher variants, or rotation parameters translate into a larger portion of the design space becoming practically testable. Accordingly, the value of a distinguisher here is determined not only by its final accuracy, but also by whether it provides a reliable signal at a cost that still permits broad experimental exploration.

We employed an evolutionary optimisation technique, namely Algorithm 11, to analyse and improve several iterations of Simon and Simeck. The optimisation was performed 10 times, with each trial randomly initializing the values of δ and γ. The evolutionary algorithm subsequently modified these parameters to attain greater accuracy in distinguishing.

Every trial underwent training using a dataset consisting of one million pairs of ciphertexts, each created using distinct random keys. The range of possible keys was systematically altered during the trials to guarantee comprehen-

sive testing. This experimental design was chosen deliberately. Simon and Simeck form a controlled benchmark pair for AND-RX analysis: they are closely related lightweight designs built from similar operations, but they differ in their diffusion-layer and key-schedule structure. Studying several variants from each family therefore makes it possible to test whether the proposed framework responds consistently to structural differences rather than only to a single toy instance. The use of one million ciphertext pairs per trial was also motivated by evidential sufficiency: because several of the final accuracies are only slightly above random guessing, a large evaluation set is needed to reduce measurement noise and to make small but systematic deviations from 50% interpretable.

In addition, for each best-found distinguisher reported in the main Simon and Simeck results, we repeated the training stage over 10 independent random seeds while keeping the cipher, number of rounds, chosen (δ, γ) pair, dataset size, and network architecture fixed. The resulting mean accuracies remained close to the single-run values reported in the tables, with standard deviations below one percentage point in the main settings. This repeated-seed study is important because it shows that the reported accuracies are not artifacts of one favourable initialization or data ordering, but stable signals produced by the same experimental configuration.

In the subsequent sections, we analyse our results for each version of the cipher, emphasizing the particular δ and γ values that resulted in the most efficient RX distinguishers and clarifying what these experiments are sufficient to support.

4.4.1 Simeck Cipher

Our evolutionary optimisation strategy successfully identified effective RX distinguishers for 15, 17, and 20 rounds of the Simeck cipher family, namely Simeck32/64, Simeck48/96, and Simeck64/128, respectively. The outcomes for each Simeck variant are summarised in Table 4.3. The table reports the best RX input differences (δ), rotational offsets (γ), the number of rounds, and the associated distinguisher accuracy found by the search procedure.

Although our suggested strategy for training deep learning-based RX distinguishers may not achieve the highest feasible performance for the Simeck cipher family, it nevertheless exhibits significant advantages. While there are distinguishers that cover more rounds, as indicated in Table 4.2, such as 20 rounds for

Table 4.3: Summary of the optimal RX input differences (δ), key differences (δ_{key}), rotational offsets (γ), the number of rounds, and distinguisher accuracy for different versions of Simeck block ciphers, where *optimal* means best found under the evolutionary-search objective used in this chapter.

Cipher Version	δ	δ_{key}	γ	Number of Rounds	Accuracy
Simeck32/64	$(0, 0x0002)$	0002	1	15	51.34
				14	57.08
				13	70.57
Simeck48/96	$(0, 0x000002)$	0002	1	17	52.06
				16	57.67
				15	69.85
Simeck64/128	$(0, 0x00000002)$	0002	1	20	52.12
				19	57.01
				18	70.15

Simeck32/64, 27 rounds for Simeck48/96, and 34 rounds for Simeck64/128, these distinguishers only work with much smaller weak-key classes, specifically of size 2^{30}, 2^{46}, and 2^{58}, respectively. In contrast, our distinguishers cover the entire key space.

This is precisely where the efficiency–accuracy trade-off in this chapter becomes meaningful. The purpose of the DL-based RX distinguisher is not to maximise benchmark accuracy for its own sake, but to act as a practical evaluation signal while keeping the search over RX parameters and key classes manageable. In that context, accuracies only slightly above random can still be valuable if they arise over the full key space and enable broader empirical exploration than stronger distinguishers that apply only to highly restricted weak-key subsets. The contribution is therefore not simply a weaker classifier, but a more broadly applicable RX-evaluation tool.

The relevance of Table 4.3 is that it shows this behaviour consistently across the Simeck family. For all three block sizes, the best full-key-space RX distinguishers remain usable at a round where accuracy is still above random, even though the margin is small. Technically, this suggests that RX-related structure survives long enough in Simeck to be detectable by the DL model after the search has tuned (δ, γ). Methodologically, it supports the chapter's claim that the proposed search-and-train framework can uncover informative RX configurations without being restricted to a narrow weak-key class.

Our examination also revealed that the Simeck cipher family appears more amenable to RX cryptanalysis than Simon under the analyses considered here.

During our investigation of Simon, we could not discover any efficient deep-learning-based methods with $\gamma \neq 0$ that outperform traditional DL-differential distinguishers. Simeck does not follow the same pattern. This is evident from the findings shown in Table 4.1, where the DL-based RX distinguishers reach performance that is close to published related-key distinguishers for the same round despite using a different and more broadly applicable evaluation setting.

That comparison is important because it clarifies what is and is not being claimed. The chapter does not show that DL-based RX distinguishers dominate the best prior related-key attacks in raw round coverage. Instead, it shows that on Simeck they provide a practically useful signal at competitive rounds while operating over the full key space, and this makes them relevant as an analysis tool rather than merely as a narrowly tuned benchmark result. The main limitation is that the accuracy margins remain modest at the hardest rounds, so the method should be viewed as a practical exploratory framework rather than a replacement for stronger specialised attacks when those are available.

4.4.2 Simon Cipher

The outcomes of implementing our approach on the Simon cipher family, particularly Simon32/64, Simon64/128, and Simon128/256, are displayed in Table 4.4. We obtained deep-learning-based RX distinguishers for 11 rounds of Simon32/64, 13 rounds of Simon64/128, and 16 rounds of Simon128/256, respectively. It is important to acknowledge that these results show weaker round coverage than current distinguishers from the literature that also cover the whole key space.

Even so, these results remain informative for the aims of this chapter. In Simon, the usefulness of the method lies less in absolute round coverage than in its ability to test whether the same DL-based RX framework transfers across cipher families and to identify how structural differences affect distinguisher quality. Seen in this light, the trade-off is between stronger headline performance and a more flexible evaluation method that can be reused across settings, parameter choices, and key classes.

Table 4.4 therefore matters primarily as comparative evidence. The fact that Simon yields weaker RX distinguishers than Simeck under the same framework is itself a meaningful result: it indicates that the method is sensitive to cipher structure rather than merely to dataset size or model choice. This supports the interpretation that differences in the diffusion layer and key schedule materi-

ally affect how much RX-related information remains detectable after several rounds. In other words, the negative result on Simon is still a useful result, because it helps explain where the proposed approach is effective and where it is not.

In addition, we provide RX distinguishers with rotational offsets γ different from 1, which, to the best of our knowledge, have not been investigated before in this setting. This shows that the proposed approach can reveal additional structure relevant to RX cryptanalysis and may help guide future investigation of cipher-design choices. The limitation, however, is clear: for Simon the current framework is better understood as an analysis tool for comparing structures and parameter choices than as a state-of-the-art attack in its own right.

Table 4.4: Summary of the optimal RX input differences (δ), key differences (δ_k), rotational offsets (γ), the number of rounds, and distinguisher accuracy for different versions of Simon block ciphers, where *optimal* means best found under the evolutionary-search objective used in this chapter.

Cipher Version	δ	δ_k	γ	Number of Rounds	Accuracy
Simon32/64	$(0x0, 0x0002)$	0002	3	11	54.45%
				10	74.11%
				9	98.48%
Simon64/128	$(0x0, 0x0)$	0000	30	13	51.51%
				12	73.15%
				11	98.50%
Simon128/256	$(0x0, 0x0)$	0000	60	16	50.62%
				15	72.26%
				14	96.87%

Our analysis revealed that the performance of the suggested technique was inferior for the Simon cipher family in comparison to the Simeck cipher family, despite the fact that both ciphers employ the AND-RX design paradigm. The dissimilarities in the diffusion layer architectures inside the round functions of Simon and Simeck ciphers might account for this phenomenon. Varying the shift settings in these routines can lead to varying levels of resistance to RX cryptanalysis. The rotational deviations in the $f(x)$ functions may have varying effects on the suggested distinguishers, hence increasing the difficulty for the evolutionary search method to discover potent RX distinguishers for Simon in comparison to Simeck.

4.5 Impact of the Diffusion Layer and Optimal Rotation Parameters

The selection of round constants and shift parameters in the construction of AND-RX ciphers plays a critical role in enhancing security against RX cryptanalysis. The work conducted by Lu et al. focused on analysing the effects of round constants in RX cryptanalysis. In contrast, the present study attempts to expand on this research by investigating the impact of shift parameters in AND-RX ciphers. We primarily want to determine optimal rotation parameters, (a, b, c), for the $f(x)$ function of AND-RX ciphers under the evaluation criterion defined earlier, which may be stated as follows:

$$f(x) = ((x \lll a) \wedge (x \lll b)) \oplus x \lll c$$

This section focuses on investigating the most effective settings for AND-RX ciphers that use a non-linear key schedule. Here, we consider ciphers that retain the Simeck round-function form,

$$f(x) = ((x \lll a) \wedge (x \lll b)) \oplus x \lll c$$

while varying the rotation parameters (a, b, c). The reason for choosing Simon as a comparison point is our empirical finding that Simon shows stronger resistance to RX cryptanalysis than Simeck under the analyses considered in this chapter. Our objective is therefore to determine whether alternative parameterisations of the Simeck round-function form can reduce distinguisher effectiveness and better compete with Simon against the deep-learning-based RX and differential attacks studied here.

Efficiency is especially important in this part of the chapter because the design problem is combinatorial: many candidate rotation sets must be explored, and each evaluation requires search over RX inputs together with model training. A design criterion that only works when the search budget is very large is of limited practical value. For this reason, the aim is not to optimise a single distinguisher in isolation, but to obtain an evaluation procedure that can screen many candidate parameterisations and still provide a useful signal about their relative resistance to RX and related-key differential attacks.

To find optimal rotation parameters, we used the evolutionary method described earlier (see Algorithm 11). We conducted an iterative approach to eval-

uate different combinations of a, b, and c parameters for Simeck32/64. We determined the greatest bias score and the maximum number of rounds for which our algorithm could identify a suitable input for the DL distinguisher for each combination. Our technique is capable of efficiently searching for the most suitable inputs, regardless of the value of γ, including when $\gamma = 0$. Consequently, the optimal shift parameters found in our experiments reduce distinguisher effectiveness against both differential and RX differential attacks under the evaluation settings considered here.

Out of all the shift sets examined, the rotation set $(4, 6, 3)$ stood out because it produced the lowest cumulative bias score in Table 4.5———and, under the evaluation criterion used here, this corresponds to the strongest resistance profile among the parameter sets we tested. Our optimiser failed to identify a distinguisher for the corresponding AND-RX cipher instantiated with the Simeck round-function form and these rotation parameters beyond 13 rounds, whereas several alternative parameter choices still admitted 14-round distinguishers. The details of these findings, which include the six rotation sets that performed best in our study, are shown in Table 4.5.

The relevance of this result is that it turns the search procedure into a design-analysis tool rather than only an attack tool. Table 4.5 does not merely rank parameter sets by a numerical score; it shows that the choice of rotation constants can measurably change how easy it is for DL-based RX distinguishers to extract signal. This supports the broader claim that diffusion-layer parameterisation is an important part of RX resistance in AND-RX ciphers. At the same time, the result should not be overstated: the table reflects resistance under the search objective, training budget, and distinguisher architecture used in this chapter, not a proof that one rotation set is globally superior under all attack models.

Table 4.5: Optimal rotation sets for AND-RX ciphers with non-linear key schedule and $n = 32$, where *optimal* means best found under the evolutionary-search objective used in this chapter

Rotation Set	Highest Cumulative Bias	Highest Round Distinguisher
(4, 6, 3)	14.32	13
(4, 5, 7)	17.28	14
(6, 7, 4)	17.99	14
(3, 7, 2)	18.25	14
(3, 5, 6)	18.67	14
(3, 6, 1)	18.95	14

The results of this section offer design-relevant insight for the development of AND-RX block ciphers. In particular, they show that rotation choices do not merely affect implementation details; they also change how much exploitable RX-related structure remains visible to the distinguisher under the evaluation models studied here. This is why the diffusion-layer results are relevant to the thesis contribution: they connect the attack methodology back to a concrete design question. The claim supported by these experiments is intentionally limited but important: within the Simeck-style round-function family and under the evaluation regime used in this chapter, the choice of rotation constants measurably affects the quality of the best-found RX distinguishers. The experiments are therefore sufficient to justify a design-oriented signal, even though they are not intended to prove that one rotation set is universally optimal.

At the same time, the conclusions must remain qualified. The parameter sets that performed best under our evaluation objective are not automatically the best choices for a practical cipher, since designers must also consider criteria such as hardware efficiency, software cost, and resistance to attacks outside the RX and related-key settings examined here. The contribution of this analysis is therefore to provide an additional design signal, not a complete design rule on its own.

4.6 Conclusion

This chapter examined the use of deep learning for RX cryptanalysis of AND-RX block ciphers, with a particular focus on the Simon and Simeck families. The main result is not simply that distinguishers were found, but that the same search-and-train framework yields meaningfully different outcomes across these cipher families. In Simeck, useful RX distinguishers can be identified across the full key space at competitive rounds, whereas in Simon the same framework produces weaker results. This difference is itself informative because it shows that the method responds to underlying structural properties of the cipher rather than only to the presence of a neural classifier.

For Simeck, the chapter demonstrated that DL-based RX distinguishers can provide a practical evaluation signal even when the hardest-round accuracies remain only slightly above random. This matters because the goal here is to support broad exploration of RX parameters and key classes, not merely to maximise headline round counts under narrow weak-key assumptions. For Simon, the weaker results are still relevant because they clarify the limitations of

the method and indicate that diffusion-layer structure and key-schedule choices materially affect RX resistance.

The chapter also showed that the search framework can be used to study diffusion-layer design choices. The identified rotation sets for AND-RX ciphers instantiated with the Simeck round-function form illustrate that parameter selection can change how resistant a design appears to DL-based RX and related-key differential analyses under the evaluation conditions considered here. This design-oriented use of the framework is an important part of the chapter's contribution.

Taken together, these results support the broader thesis claim that machine learning can be used as a practical tool for evaluating symmetric primitives, not only by producing distinguishers, but also by helping compare cipher families, search parameter spaces, and extract design-relevant signals. The main limitation is that the conclusions remain tied to the specific search objective, model architecture, and computational budget used in this chapter; accordingly, the results should be interpreted as strong empirical evidence rather than as a complete characterisation of RX resistance.

Chapter 5

Privacy-Preserving Sentiment Analysis Using Homomorphic Encryption and Attention Mechanisms

Homomorphic encryption (HE) enables computation on encrypted data and is therefore a natural tool for privacy-preserving machine learning (PPML), particularly in cloud settings where sensitive inputs must not be revealed to the service provider. This chapter presents a privacy-preserving sentiment-analysis pipeline based on the Cheon–Kim–Kim–Song (CKKS) scheme [CKKS17]. We split the model into (i) a client-side attention module that selects and encrypts the most informative features before transmission and (ii) a server-side prediction module that evaluates the encrypted representation. To keep encrypted inference efficient, the server-side network uses an RNN layer with an identity activation, followed by a linear layer. The client decrypts the server output and applies a sigmoid function to obtain the final sentiment score. We evaluate the approach on the IMDb dataset [MDP+11] and obtain a mean accuracy of 70.8% $\pm$ 1.2%, together with a precision of 71.3% $\pm$ 1.5%, a recall of 69.6% $\pm$ 1.4%, and an F1-score of 70.4% $\pm$ 1.3%. Overall, the chapter shows how a careful division of responsibilities between client and server can balance privacy protection with practical performance in the HE-based setting considered here.

The results presented in this chapter were published as part of the proceedings of the Applied Cryptography and Network Security (ACNS 2024) co-

located workshop Security in Machine Learning and its Applications (SiMLA 2024). [MGP24]

5.1 Introduction

Large-scale data collection from social platforms, web activity, and location-based services has increased the demand for machine learning (ML) methods that can extract useful patterns from text, images, and sensor streams. Neural networks are now widely used in applications such as image classification, natural language processing, and speech recognition.

As ML usage grows, so do concerns about confidentiality. When inference is outsourced to a cloud platform, both the client inputs and the model itself may be sensitive. For example, medical records, census data, or government datasets are often subject to legal and contractual restrictions. Naïve data outsourcing can therefore create privacy risks if the service provider is not fully trusted.

A range of approaches have been proposed to address these issues, including differential privacy, secure multi-party computation, trusted execution environments, and homomorphic encryption. In this chapter we focus on homomorphic encryption (HE), which enables inference on encrypted inputs. However, encrypted inference is typically expensive, especially when models contain non-linear operations. Designing PPML solutions therefore requires careful choices that balance accuracy against computation and communication cost.

In this setting, efficiency is not a secondary optimisation target but a deployment constraint. Under HE, multiplicative depth, ciphertext size, memory usage, latency, and communication cost can quickly make an inference pipeline impractical, even when predictive performance is strong. For this reason, the goal of this chapter is not to maximise sentiment-classification accuracy in isolation, but to identify a model structure whose predictive performance remains useful while encrypted inference becomes feasible under realistic HE constraints.

5.1.1 Motivation and Contribution

Recently, there has been a substantial rise in the utilization of machine learning algorithms to extract valuable information from the vast amount of data produced on a daily basis. Although these advancements have shown significant usefulness, they have also intensified worries around data privacy.

The primary objective of our study is to tackle these privacy problems by employing homomorphic encryption, namely CKKS homomorphic encryption [CKKS17], which is widely used in privacy-preserving techniques. Nevertheless, our methodology aims to decrease computational complexity while protecting data confidentiality. The main contributions of this chapter are as follows:

- **Simplified Activation Function for RNN Layer:** Our strategy is to incorporate a linear activation function, delineated as $f(x) = x$ in the RNN layer, reducing the computational demands ordinarily involved in handling encrypted data. This choice keeps the multiplicative depth low, which reduces the need for expensive non-linear approximations and may avoid bootstrapping under typical parameter settings, while also limiting client–server interaction.
- **Client-Server Model with Segmented Responsibilities:** Our model splits the computations between the client and the server, where the client handles the attention mechanism while the server manages the prediction through the RNN layer with the simplified activation function. This bifurcation supports privacy preservation, since the server operates solely on encrypted data, while also improving efficiency.
- **Attention Mechanism to Enhance Shallow Networks:** We integrated an attention mechanism with shallow networks to focus selectively on critical data during the inference process, retaining essential information and enhancing the network's performance without escalating the computational demands excessively.
- **Real-World Application and Validation:** We validate the methodology on the IMDb comment dataset for sentiment analysis and compare against prior work [PT20, PT21].

Although the current approach achieves a mean accuracy of 70.8% $\pm$ 1.2% (lower than some prior work), together with a precision of 71.3% $\pm$ 1.5%, a recall of 69.6% $\pm$ 1.4%, and an F1-score of 70.4% $\pm$ 1.3%, it demonstrates a practical design point for privacy-preserving machine learning. These balanced metrics suggest that useful performance can still be obtained with a relatively lightweight encrypted-inference pipeline rather than appearing favourable under only a single measure. This provides a basis for future optimisation. This work advances privacy-preserving machine learning (PPML) by emphasising an HE-friendly (identity) activation within the inference pipeline. The goal is to improve the efficiency of PPML cloud services.

This trade-off is meaningful because, in PPML, a slightly less accurate model can still be more valuable than a more expressive model if the latter is too expensive to evaluate on ciphertexts. In other words, the relevant comparison is not simply highest benchmark accuracy, but whether the reduction in model complexity materially lowers the practical cost of encrypted inference while preserving acceptable predictive performance.

5.1.2 Outline

Section 5.2 summarises activation functions that are practical in homomorphic-encryption settings. Section 5.3 describes the proposed client–server methodology, and Section 5.4 details the implementation choices for encrypted inference. The experimental results are presented in Section 5.5. Finally, Section 5.6 concludes the chapter and outlines directions for future work.

5.2 Efficient Activation Functions

Activation functions are mathematical functions that are used to add non-linearity into the output of a layer in a neural network. This non-linearity allows the network to learn and recognize complicated patterns. Nevertheless, evaluating activation functions homomorphically poses a challenge due to the need for high-degree polynomials or costly operations like exponentiation or logarithms.

In accordance with the methodology employed in CryptoDL [HTG17], Bakshi and Last [BBDH21] necessitate the client to do activation computations on plaintext. However, their approach just encompasses shallow networks and brief inputs. In the study conducted by Podschwadt and Takabi [PT21], the client's role did not include computation, but it did entail the task of removing noise.

Activation functions can be represented as low-degree polynomials or piecewise linear functions, such as the square function, sine function, the ReLU function, and others. Pre-trained activation functions, such the sigmoid function, the tanh function, the softmax function, etc., can be stored as lookup tables and retrieved homomorphically. We employ this method to transform the tanh activation function into a linear form and apply it in a recurrent neural network. Next, the results are compared to the studies conducted by Podschwadt [PT20] and Podschwadt and T21RNN [PT21].

5.3 Methodology

This section elaborates on the approach employed in our study to create a privacy-preserving sentiment analysis system. We use CKKS homomorphic encryption and attention mechanisms in shallow neural networks. The methodology presents the suggested approach, explaining the intricate details of our model's architecture, the attention mechanism, and the model training process. It also provides a concise overview of the dataset used in this work. Figure 5.1 depicts a diagram illustrating the process of our technique.

5.3.1 Proposed Method

Our methodology entails partitioning the activities of the neural network model across the client-side and server-side, using the CKKS technique. Here, we provide a comprehensive explanation of their distinct roles and purposes:

- **Client-side Component:** On the client side, the initial layers of the neural network model, up to and including the attention mechanism, are deployed. Our research uses a dense layer and a reshape layer after the attention mechanism to compress the essential information into a succinct manner. This format is shown as a matrix with size $(None, 32, 32)$. Compression is important because it reduces the computational load during server-side processing and packages the relevant information into a compact representation. Afterwards, the CKKS scheme encrypts this compressed data, helping preserve confidentiality while the data is transmitted to the server for further processing.
- **Server-side Component:** The encrypted data is received by the server-side and undergoes processing in the later part of the neural network model. This section contains a SimpleRNN layer with a linear activation function $(f(x) = x)$, followed by a dense layer. The use of a linear activation function is a deliberate design choice aimed at reducing processing requirements by minimising the number of operations involved, thereby reducing the likelihood that bootstrapping is needed under the parameter settings used in our experiments. This component of the architecture is designed to operate directly on encrypted data, thereby helping preserve the confidentiality of the client's information. Once the data has been processed, it is still encrypted and then returned to the client for decryption. The client then applies the *sigmoid* operation on the data for the purpose

of final analysis.

5.3.1.1 Model Architecture

The neural network model is designed to operate efficiently in a privacy-preserving setting, using the CKKS technique. The architecture may be conceptualized as a series of levels, each meticulously crafted with distinct functionality, as delineated in the following explanation:

- **Input Layer:** The first layer of the network accepts input data in the form of (None, n_{input}), serving as the starting point for the dataset to pass through the network. The value of n_{input} is dictated by the choice of padding for our data. The accuracy of the machine is strongly influenced by this padding, as will be further addressed in Section 5.5.
- **Embedding Layer:** The embedding layer, which comes after the input layer, is set up with a dimensionality of d_{emb}. Its purpose is to convert the input data into a higher-dimensional space, enabling the model to learn more intricate patterns.
- **Bidirectional Layer:** To improve the model's ability to learn, a bidirectional layer with n_{bi} units is used. This allows the network to learn from both past (backward) and future (forward) input, hence boosting the network's understanding of context.
- **Attention Mechanism:** The attention mechanism, which is introduced to assess the significance of various components of the input, works in conjunction with the bidirectional layer to enhance the model's concentration on relevant characteristics.
- **Concatenate and Flatten Layers:** After the attention mechanism is applied, the bidirectional and attention outputs are combined via concatenation. This is followed by a flattening layer, which prepares the data for subsequent processing.
- **Dense Layer:** Here, we use a dense layer with n_{dense} units to introduce non-linearity to the system. This helps in understanding intricate patterns from the flattened data and compressing the output information before transmitting it to the server.
- **Reshape Layer:** The output of the dense layer is reshaped into a matrix with dimensions (n_{r1}, n_{r2}), which prepares it for the following SimpleRNN layer.
- **Batch Normalization Layer:** This layer, located after the reshape layer,

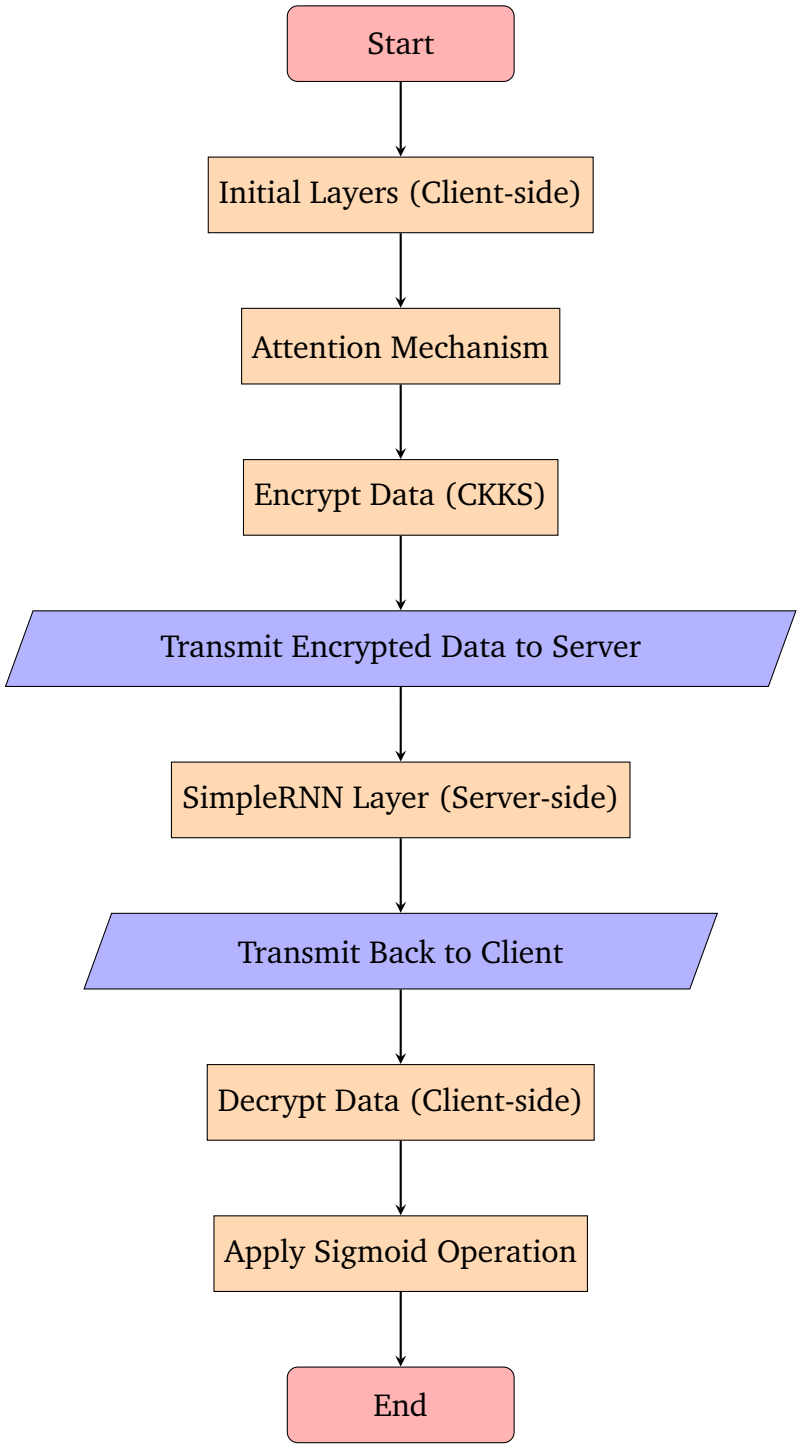


Figure 5.1: Simplified Schematic Diagram of the Client-Server Privacy-Preserving Neural Network

normalizes the activations of the neurons in the current batch. This normalization process can create more favorable circumstances for the linear activation function used in the succeeding SimpleRNN layer. We shall provide a more comprehensive discussion on this topic in Section 5.4.

- **SimpleRNN Layer:** The layer has n_{Simple} units and uses a linear activation function, which is deliberately chosen to minimize the computational complexity during the homomorphic encryption procedures.
- **Output Dense Layer:** The last component in the design is a dense layer containing only one unit. Its purpose is to generate the prediction score before activation, which enables further client-side actions after encryption.

This architecture incorporates an attention mechanism that divides the task between the server and client components. Both components are important for privacy during the inference process. Additionally, the architecture is designed to have a manageable computational impact, allowing it to handle encrypted data efficiently. Figure 5.2 provides a more detailed representation of the architecture we use.

The final architecture was selected under a multi-objective constraint rather than by plaintext accuracy alone. At each stage, the design had to preserve enough expressive power for sentiment classification while also keeping the encrypted server-side computation feasible under CKKS. This is why the client side performs the embedding, bidirectional, attention, and compression stages, whereas the server is limited to the recurrent prediction block on ciphertexts. Using GloVe embeddings with $d_{\text{emb}} = 50$ provides a compact pre-trained representation that is standard in sentiment analysis and significantly lighter than higher-dimensional alternatives. The bidirectional layer with $n_{\text{bi}} = 128$ was chosen to capture context in both directions without making the post-attention representation unnecessarily large. The dense layer with $n_{\text{dense}} = 1024$ compresses the client-side representation before encryption, and reshaping that output to 32×32 gives a regular fixed-size interface to the encrypted recurrent stage. The SimpleRNN width was kept at $n_{\text{Simple}} = 64$ because increasing the recurrent width directly increases the cost of the homomorphic computation and the server-side memory footprint. The final architecture therefore reflects a practical trade-off between predictive performance, ciphertext size, multiplicative depth, and encrypted runtime.

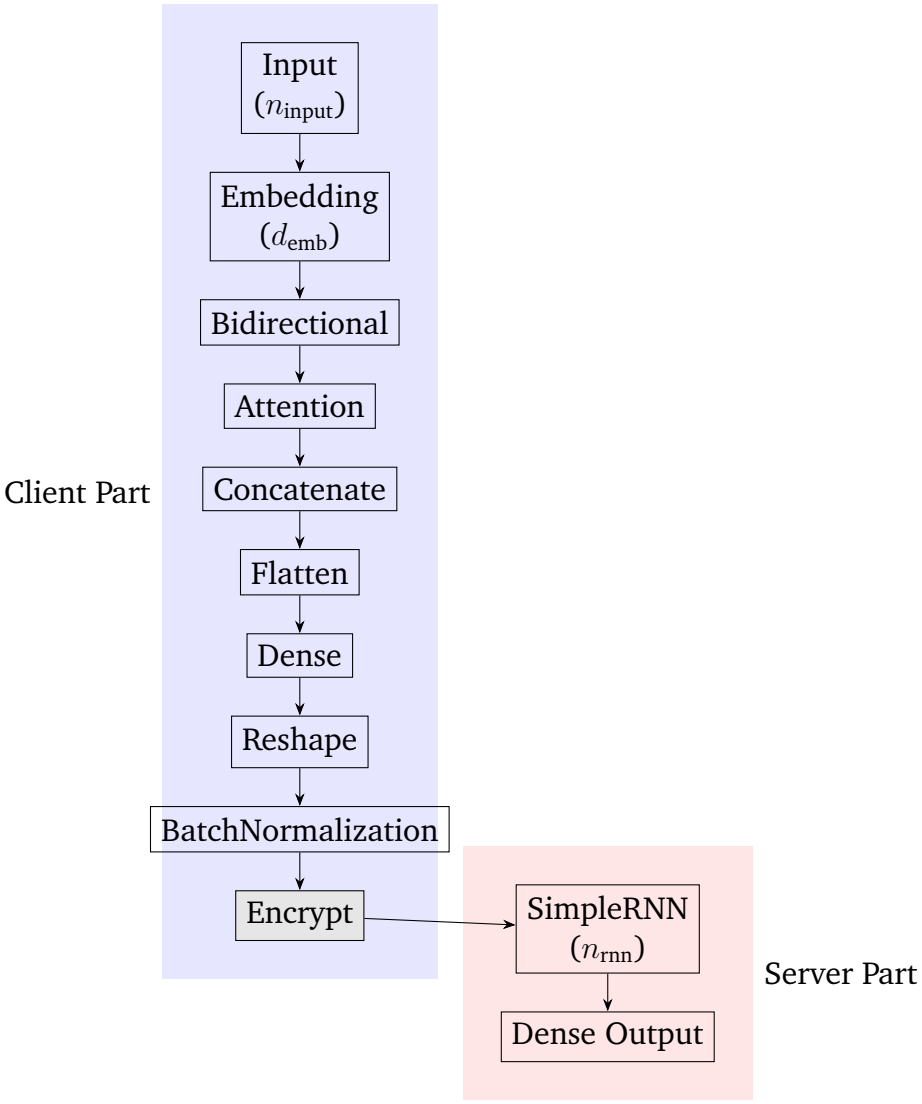


Figure 5.2: High-level architecture of the privacy-preserving machine learning system, delineating the client and server components.

5.3.2 Training and Inference

5.3.2.1 Training

During the training phase, the Keras framework was used to construct and train the neural network using unencrypted data, allowing for the accurate calculation of the weights for each layer.

The data underwent preprocessing using Python and the NLTK package. This method involved removing extraneous content such as stopwords and non-alphabetic tokens, with a specific focus on words containing more than two characters. Subsequently, the Keras Tokenizer function transformed the processed input into sequences of integers, so preparing for the further stages of

the training process.

The embedding layer was set up to use GloVe with pre-trained vectors of 50 dimensions, enabling the comprehension of word relationships by converting word integers into compact vectors.

The model was compiled using the Adam optimizer, which is often preferred for training deep neural networks because of its high efficiency. The binary cross-entropy loss function was selected to calculate the loss between the observed and predicted values, given that the job involves binary classification, specifically distinguishing between positive and negative emotion. The accuracy measure was used to assess the model's performance throughout the training phase.

The training configuration was chosen with the deployment setting in mind rather than by maximising validation accuracy alone. Binary cross-entropy is the natural loss for the positive/negative sentiment task, and Adam was selected because it gave stable convergence for this network without requiring elaborate tuning of the learning schedule. The pre-trained GloVe embedding was retained to avoid learning a large embedding matrix from scratch and to keep the client-side model compact from the outset. More importantly, the downstream encrypted-inference cost constrains what is useful at training time: a larger or deeper model may improve plaintext accuracy slightly, but it also increases the size of the representation that must be encrypted and the cost of the server-side recurrent computation. Accordingly, the final model was selected as a practical balance between classification quality and HE feasibility rather than as the largest architecture that could be trained in plaintext.

5.3.2.2 Inference Phase

The inference phase, conducted after the training phase, incorporates an extra step to augment data security. Once the attention mechanism has processed the data, it undergoes encryption before being transmitted to the server for additional processing. The server uses a basic Recurrent Neural Network (RNN) layer and a dense layer to handle the encrypted data, employing the pre-trained weights obtained during the training phase.

Following the processing that occurs on the server, the data is then transmitted back to the client. At this stage, the encrypted output received from the server is decrypted, and the data undergoes a sigmoid function transformation to obtain the final sentiment-analysis outcome. This framework is designed to

protect data confidentiality while utilising the server's computational capabilities for analysis, creating a practical system for privacy-preserving sentiment analysis.

The split point between client and server was chosen deliberately. Keeping the attention mechanism on the client side allows the model to identify and compress the most informative features before encryption, which reduces both communication volume and encrypted computation. By contrast, keeping the recurrent and output layers on the server side preserves the proprietary part of the model while still exploiting the server's greater computational resources. This division therefore addresses two design goals simultaneously: lowering encrypted-inference cost and limiting exposure of the full model to the client.

5.3.2.3 Dataset Description

The dataset used in this study is obtained from the extensive movie review dataset, often known as the IMDb dataset. The dataset for sentiment analysis is widely recognized and consists of a large number of movie reviews that have been categorized as either positive or negative. The training subset of the dataset consists of 25,000 reviews, with an equal distribution of 12,500 good reviews and 12,500 negative reviews. The substantial amount of data facilitates the training of the model to identify complex patterns and comprehend positive and negative feelings conveyed in the evaluations.

Due to the limitations in computing power and memory capacity caused by using the CKKS encryption algorithm to encrypt several plaintexts into a single ciphertext, the number of samples in the test subset was limited to 200 per run. More precisely, as the test size is increased, it becomes necessary to increase the number of slots in the CKKS encryption process. This, in turn, causes the memory needs to increase sharply and resulted in memory difficulties in our implementation. To address this limitation and still provide a broader evaluation of model behaviour, the testing process was repeated 50 times using different samples. Each run used 200 randomly selected reviews from the dataset. This repeated-evaluation design was chosen to balance feasibility and robustness: 200 samples keep each encrypted run manageable, while 50 repetitions reduce the chance that the reported performance is dominated by a single favourable or unfavourable sample.

The IMDb dataset was chosen because it is a widely used benchmark for binary sentiment classification and has also been used in prior HE-based

sentiment-analysis studies [PT20, PT21]. This makes it suitable for evaluating whether the proposed architecture can achieve privacy-preserving inference under a directly comparable task setting. In particular, using IMDb allows this chapter to focus on the main question of interest: whether an HE-compatible split architecture with a simplified encrypted recurrent component can provide a meaningful cost–accuracy trade-off on a standard sentiment benchmark.

At the same time, evaluation on a single dataset does not by itself justify broad general conclusions about all sentiment-analysis tasks or NLP settings. The IMDb results should therefore be interpreted as evidence of feasibility on a representative benchmark rather than as proof of uniform generalisation across domains, writing styles, sequence lengths, or label structures. The main empirical claim supported by this chapter is thus task-specific: the proposed HE-based design works on IMDb and exposes a concrete trade-off between predictive performance and encrypted-inference cost.

In subsequent parts, we will discuss the implementation of our approach and provide the findings we acquired through its use.

5.4 Implementation

In this section, we detail the specific implementation steps undertaken to develop our privacy-preserving sentiment analysis system.

5.4.1 Hardware and Software Setup

This section provides an overview of the essential hardware and software setups used at various phases of our project. The configuration described below plays an important role in both the training and inference stages.

5.4.1.1 Training

The training procedure was executed using Python 3.8 and harnessing the Keras library, making use of the computing capabilities of Google Colab. The training settings were deliberately selected to cultivate a resilient neural network model. The layer characteristics are provided in full below:

- $n_{\text{input}} = 413$: specifying the dimension of the input layer.
- $d_{\text{emb}} = 50$: denoting the output shape of the embedding layer.
- $n_{\text{bi}} = 128$: the output dimension of the bidirectional layer.

- $n_{\text{dense}} = 1024$: defining the first dense layer's output dimension.
- $(n_{\text{r1}}, n_{\text{r2}}) = 32 \times 32$: illustrating the reshape layer dimensions.
- $n_{\text{Simple}} = 64$: indicating the SimpleRNN layer's output dimension.

These parameter values were chosen to keep the client-side representation expressive while maintaining a server-side encrypted computation that remains tractable. The input length $n_{\text{input}} = 413$ follows from the padding analysis reported later in this chapter and was selected because it gave the best observed trade-off between retaining informative text and preserving stable decrypted outputs. The embedding size of 50 was retained because it is a standard compact choice for GloVe-based sentiment models and avoids the extra cost of larger embeddings. The bidirectional width of 128 and dense width of 1024 were found to be sufficient to capture context and compress the attended representation without making the encrypted interface unnecessarily large. Reshaping to 32×32 gives a regular fixed-size interface to the server-side recurrent block, and choosing $n_{\text{Simple}} = 64$ keeps the recurrent computation moderate once moved into the homomorphic setting.

The L2 regularization value was set at 0.001 in order to mitigate overfitting, and the model underwent training for a total of 40 epochs. These values were selected because shorter training runs led to less stable validation behaviour, whereas substantially longer runs increased training cost without a commensurate improvement in the final HE-compatible design. The activation functions used were Rectified Linear Unit (ReLU) for the first dense layer, the identity function ($f(x) = x$) for the SimpleRNN layer, and a sigmoid function for the final dense layer. This combination reflects the same design principle as the rest of the model: use standard non-linearities where they are inexpensive on the client side, and reserve HE-friendly linear operations for the encrypted server-side stage.

5.4.1.2 Inference

For the inference stage, we used C++ coupled with the CryptoDL library [HTG17] on the server-side to effectively perform homomorphic encryption. This library, which is used for performing deep learning operations in an encrypted environment, plays an important role in our implementation. It interfaces with the HElib library [HS14] for CKKS encryption.[1]

[1] More details can be found in https://github.com/inspire-lab/CryptoDL.

5.4.2 Impact of Padding Number on the Implementation

An essential component of data preparation for both training and inference is maintaining a uniform sequence length by using padding. Since our data consist of IMDb movie comments with varying lengths, it is important to choose a suitable padding value that does not negatively affect accuracy or decryption on the client side.

We observed that the longest remark in our sample extended to 1317 tokens. Assigning the maximum value to the padding number would result in the majority of comments being subjected to significant zero-padding. However, this strategy introduces problems, especially in the decryption step that comes before application of the sigmoid function on the client side. In our experiments, a large padding value caused the decrypted values to approach zero, effectively cancelling the output.

To address this issue, we select a padding value that serves two goals: retaining a large portion of the original comment data and helping ensure that the decrypted values remain meaningful for subsequent processing. To begin, we set a maximum padding number that would include at least 95% of the comments without any truncation. The method resulted in determining 620 as the appropriate upper limit.

Afterwards, we conducted a series of experimental operations to determine the most suitable padding number. Eventually, we settled on 413, which covers 75% of the length of the remark without any truncation. This value gave the best empirical trade-off in our implementation: smaller padding values removed too much information from the reviews, whereas larger values increased ciphertext size and made the decrypted outputs more likely to collapse toward zero before the client-side sigmoid. The selected value therefore reflects both predictive performance and encrypted-inference stability, and under the repeated-evaluation protocol described above it yielded a mean accuracy of $70.8\% \pm 1.2\%$. Figure 5.3 graphically illustrates the distribution of comment lengths and helps explain the choice of the final padding value.

Careful selection of the padding value is therefore important for balancing accuracy and successful decryption, and for supporting stable final outputs.

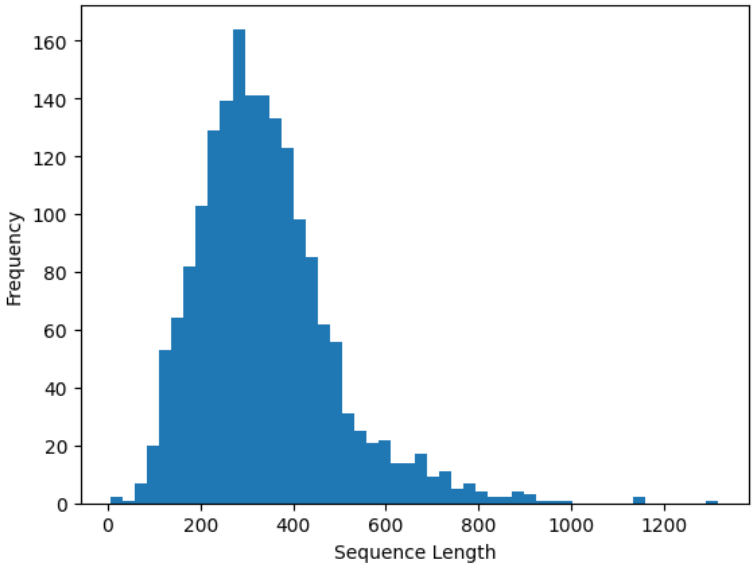


Figure 5.3: Distribution of comment lengths in the dataset

5.4.3 Rationale for Selecting a Linear Activation Function

To build a practical privacy-preserving sentiment-analysis system, it is necessary to identify a suitable method for representing the tanh activation function within the SimpleRNN layer. In this study, we assessed the practicality of approximating it using a linear function. Previous studies in this field mostly employed Chebyshev approximation due to its high level of accuracy over wide ranges [HTG17, PT21]. However, because of the structure of our network, where the output from the embedding layer does not immediately connect to the SimpleRNN layer, but instead passes through several layers including a BatchNormalization layer, a different technique was necessary.

The Batch Normalization layer has the ability to shift the data distribution towards zero, resulting in a smaller standard deviation. This makes a linear approximation a plausible option in our setting. Located subsequent to the reshape layer, this layer functions to standardize the activations of the neurons in the present batch, with the goal of keeping the average activation around 0 and the standard deviation near 1. The pre-processing layers condition the data so that, when the inputs reach the SimpleRNN layer, they tend to lie within a relatively confined range. In that range, the tanh function behaves approximately linearly, which makes a linear approximation reasonable for the setting considered here, although it does not guarantee a zero-centred distribution for the succeeding SimpleRNN layer with linear activation.

The results suggest that the tanh function can be approximated adequately

within the specified range while benefiting from the simplicity and lower computational cost of a linear function. Within the setting studied here, this approximation appears compatible with the behaviour induced by the preceding network layers.

5.4.4 Justification for Server-Side Computation Despite Simplified Activation Function

An important aspect of our approach is the deliberate allocation of computing duties between the client and server sides. The choice to use a simpler linear activation function on the server side is motivated not only by the need for computing efficiency in homomorphic encryption (HE), but also by the need to protect the proprietary part of the server's model. Although it is technically possible to do the full model execution on the client side, this would necessitate revealing the entire model architecture and parameters to the client. Exposing the server might result in security problems, such as the illegal duplication or reverse-engineering of the server's proprietary model.

Our strategy involves deliberately dividing the model, with the customer only having access to a certain component of it. The act of partitioning helps limit exposure of the server-side model. The server retains a substantial part of the computation, namely the components most relevant to the proprietary part of the model. By processing encrypted data on the server side, even with a minimal linear transformation, the design can reduce the risk of model theft or unauthorised access compared with revealing the full model to the client.

Furthermore, the server-side calculation accounts for the possibility of the server having superior memory and processing capabilities in comparison to the client. The disparity in processing resources enables the server to perform more intricate operations on encrypted data, which can improve the efficiency of the privacy-preserving machine-learning (PPML) system as a whole. Our results indicate that the server does not need to perform all calculations in order to maintain an efficient PPML pipeline. In the setting studied here, this balance lets us use the server's resources while still keeping important components of the model hidden from the client, without requiring substantial bootstrapping or very large memory demands.

5.5 Results

This section evaluates the central trade-off of the chapter: whether the proposed architecture gives up too much predictive performance in exchange for better encrypted-inference efficiency. In the HE setting, efficiency improvements are meaningful only if they reduce concrete bottlenecks such as multiplicative depth, RAM usage, server-side complexity, or interaction cost. Accordingly, the results are interpreted not only in terms of predictive quality, but also in terms of whether the resulting pipeline is more deployable than heavier alternatives.

Under the repeated-evaluation protocol described in Section 5.3, the proposed method achieved a mean accuracy of 70.8% $\pm$ 1.2%, a precision of 71.3% $\pm$ 1.5%, a recall of 69.6% $\pm$ 1.4%, and an F1-score of 70.4% $\pm$ 1.3% on the IMDb sentiment-analysis task. Reporting multiple metrics is important here because the chapter claims a practical deployment trade-off rather than performance under a single favourable score. The fact that accuracy, precision, recall, and F1-score all lie in a similar range indicates that the model retains balanced predictive utility and is not appearing strong only because of one selectively favourable metric. This level of performance is lower than that of some more expressive approaches, but the chapter does not claim state-of-the-art predictive accuracy. Rather, the relevance of the result lies in showing that useful encrypted inference remains possible after simplifying the server-side network so that it is better aligned with the cost constraints of CKKS-based evaluation. In this setting, the key question is therefore not whether the model is the most accurate in absolute terms, but whether the loss in predictive power is small enough to justify the resulting reduction in HE cost. The repeated measurements also make the evidence stronger: the relatively small standard deviations suggest that the reported performance is stable across the 50 sampled test runs used in our evaluation protocol, rather than being dominated by a single favourable sample.

In resource-constrained cryptographic settings, balancing computational efficiency, memory use, and security is important. In this section, we compare our method with two earlier studies. One employs an interactive approach [PT20], while the other uses a non-interactive approach [PT21]. The goal of the comparison is not to claim superiority on every axis, but to clarify the operating point reached by the proposed design.

Podschwadt et al. used parallel structures, which require many cores and sub-

stantial RAM, whereas [PT20] also relies on bootstrapping for noise reduction, which is computationally expensive. By contrast, our method avoids parallel server-side structures and was executed in the lower-RAM setting considered here. This difference is relevant because memory consumption and heavy-weight ciphertext operations are often the factors that determine whether an HE-based pipeline can be deployed at all, rather than merely benchmarked in a favourable environment.

The study in [PT21] shows that increased speed can come with greater RAM usage and a more complex server-side layer. Our approach instead targets a more conservative design point: reduce the burden on memory and multiplicative depth while keeping the server-side model simple enough to remain practical under HE. This matters because a deployment-oriented PPML system must often trade some expressive power for a configuration that can actually be executed with available hardware and encryption parameters.

Another notable aspect of our study is the use of a reduced multiplicative depth (MD) of $2t$ in the CKKS scheme, whereas the compared studies use $4t$. This reduction is significant because multiplicative depth directly influences ciphertext noise growth, parameter selection, and the likelihood that one must resort to costly bootstrapping. A lower depth therefore does not merely indicate a smaller arithmetic circuit; it also signals that the encrypted inference procedure is easier to sustain in practice. The advantages of our technique, particularly in terms of reduced RAM requirements, are illustrated in Table 5.1.

Table 5.1: Comparison of our work with previous approaches

Metric	**Our Work**	[PT21]	[PT20]
Use Bootstrapping	No	No	Yes
RAM Usage (GB)	12	215 - Cloud	32
Multiplicative Depth	$2t$	$4t$	$4t$
Activation Degree	1	3	3
Server Layer Complexity	Simple RNN	Parallel-Simple RNN	Simple RNN
Client Network Insight	Yes	No	No

The comparison in Table 5.1 highlights why the efficiency claim matters in this chapter. The proposed method does not attempt to dominate prior work on every metric; instead, it targets a lower-complexity operating point with reduced multiplicative depth, substantially lower RAM requirements than some alternatives, and a simpler server-side layer. Each of these differences has practical relevance in HE. Lower RAM usage improves the feasibility of running the system outside very high-resource environments. Lower multiplicative depth re-

duces pressure on parameter selection and helps avoid expensive refresh mechanisms. A simpler server-side layer lowers the cost of encrypted evaluation and makes the overall pipeline easier to implement and maintain.

The activation degree of 1 should also be interpreted as part of this same trade-off. Using a linear activation avoids the higher-degree polynomial approximations required for non-linear activations under HE, but this simplification also reduces the expressive power of the encrypted recurrent component. In other words, the benefit is not free: the model becomes easier to evaluate homomorphically precisely because the server-side recurrent block is less capable of modelling complex non-linear decision boundaries on its own. The relevance of the result is therefore that, despite this simplification, the model still achieved a mean accuracy of $70.8\% \pm 1.2\%$ together with balanced precision, recall, and F1-score values. This suggests that the split architecture preserves enough task-relevant information to make encrypted inference useful on IMDb, even though the server-side recurrent component is deliberately less expressive than richer plaintext alternatives.

The comparison table should therefore be read as evidence of a deployment trade-off rather than as a leaderboard. The proposed model reaches a point at which encrypted inference is materially lighter in terms of depth, memory, and server-side complexity, while still producing useful predictions. This is the main contribution supported by the results: not that the method is uniformly better than all prior approaches, but that it occupies a more practical HE-oriented operating point.

Our approach also addresses the trade-off between openness and security by keeping only part of the model on the client side while retaining layers such as the SimpleRNN and prediction head on the server. This design helps preserve the confidentiality of the server-side model while still enabling collaboration between client and server. The relevance of this result is architectural as well as numerical: the chapter shows that useful PPML performance can be obtained without forcing either party to reveal the full model or to carry the full encrypted workload alone.

At the same time, this approach improves efficiency by transferring part of the computation to the client side. This reduces the burden on the server and suggests that carefully partitioning work between client and server can be a practical design direction for collaborative cryptographic computation. The trade-off, however, is that some transparency is introduced on the client side because the client has insight into part of the network, while the overall predictive perfor-

mance remains below that of richer, less HE-constrained models.

A further extension of this collaborative design could involve secure multi-party computation techniques to integrate inputs from multiple parties more privately. Such an extension might provide stronger security guarantees for protecting server-side operations, but would likely introduce additional computational overhead. This observation reinforces the broader interpretation of the results: stronger privacy and stronger model protection often come at the price of efficiency, and the present chapter aims to identify a viable balance rather than eliminate that tension.

Taken together, these results support the claim that the contribution of this chapter is not simply a lower-accuracy model, but a more realistic encrypted-inference design point. For HE-based sentiment analysis, the relevant question is whether the model remains good enough while the encrypted computation becomes substantially easier to support. In that sense, the observed reduction in model complexity is not merely an implementation convenience; it is part of what makes the method usable in the first place.

To summarise, our work presents a practical method for privacy-preserving machine learning (PPML), especially in settings with limited memory resources. The main significance of the reported results is that they show how predictive utility, encrypted complexity, and model confidentiality can be balanced within one design. The fact that accuracy, precision, recall, and F1-score remain closely aligned strengthens the claim that the architecture preserves broadly usable classification behaviour rather than performing well only under a single metric. At the same time, the results also expose the limitation of that balance: lower encrypted cost is achieved partly by reducing the expressiveness of the server-side model. Future work should therefore aim to improve accuracy without losing the gains in multiplicative depth, RAM usage, and architectural simplicity demonstrated here.

5.6 Conclusion

This work focuses on techniques to improve the performance of encrypted neural networks built with homomorphic encryption (HE). It addresses the high computational cost of non-linear operations and presents a method for approximating the tanh activation function within an RNN over data encrypted with the CKKS cryptosystem. In our IMDb sentiment-analysis experiments, the ap-

proach reduced the computational burden of encrypted inference in our implementation while keeping the data encrypted throughout server-side processing. Under the repeated-evaluation protocol, it achieved a mean accuracy of 70.8% $\pm$ 1.2%, together with a precision of 71.3% $\pm$ 1.5%, a recall of 69.6% $\pm$ 1.4%, and an F1-score of 70.4% $\pm$ 1.3%.

The main lesson is that, in this setting, efficiency must be treated as part of the security and deployment problem rather than as a purely secondary objective. A modest loss in predictive performance can be justified if it yields a meaningful reduction in multiplicative depth, memory consumption, or server-side complexity, because those factors determine whether HE-based inference is practical at all. The contribution of this chapter is therefore a design that makes this trade-off explicit and usable in a realistic sentiment-analysis pipeline.

Secure neural networks may ultimately require end-to-end security measures that cover pre-processing, training, and inference. Some forms of privacy-preserving pre-processing may be possible using techniques such as private set intersection, Bloom filters, or related primitives for secure search, updating, and deletion over protected data. Although FHE is a powerful tool, practical solutions will likely need to combine it with other cryptographic primitives such as SMPC, federated learning (FL), or differential privacy (DP) to support a broader range of capabilities efficiently.

Chapter 6

Conclusions and Future Work

6.1 Overview and Summary of Contributions

This thesis investigated problems at the intersection of cryptanalysis, machine learning, and privacy-preserving computation. The contributions follow a common theme: using machine learning as a practical tool for evaluating symmetric primitives, while using cryptographic techniques to enable machine-learning inference under privacy constraints. The thesis developed and evaluated three lines of work: (i) reducing the computational cost of machine-learning-assisted differential attacks, (ii) extending rotational-XOR (RX) cryptanalysis of Simon-like block ciphers using deep-learning-based distinguishers and RX-characteristic analysis, and (iii) designing a privacy-preserving sentiment-analysis pipeline based on homomorphic encryption and attention mechanisms.

Across these three lines of work, efficiency is treated as a requirement of practical usefulness rather than as an isolated optimisation target. In Chapter 3, efficiency matters because a distinguisher that is too costly to train or use in downstream attack procedures is of limited cryptanalytic value, even if its raw accuracy is strong. In Chapter 4, efficiency matters because RX analysis is used as a search and evaluation framework over a large parameter space, so moderate distinguisher quality can still be valuable when it enables broader exploration. In Chapter 5, efficiency matters because encrypted inference is constrained by multiplicative depth, ciphertext growth, latency, and communication cost, making deployability a central concern. For this reason, the relevant question throughout the thesis is not whether efficiency improves at any cost, but whether a given reduction in cost is justified by maintaining enough

accuracy, distinguishing strength, or predictive utility for the task at hand.

Reducing the cost of ML-assisted differential cryptanalysis (Chapter 3). To reduce the time and memory requirements of machine-learning-assisted differential attacks, we proposed a Partial Differential (PD) machine-learning distinguisher that operates on a selected subset of output-difference bits rather than the full block state. We evaluated the approach on SPECK32/64 and showed that selecting high-scoring bits can preserve most of the distinguishing power while substantially lowering computational requirements. In particular, the PD-ML distinguisher achieved an accuracy of 68.8% using 8 selected bits, compared to 72% using all 32 bits over six rounds of SPECK32/64. The resulting reduction in input dimensionality also decreased the time complexity of the key-averaging procedure for training a 7-round distinguisher by a factor of 25, with only a 3% absolute reduction in accuracy. In this setting, the trade-off is meaningful because the objective is not maximum classifier accuracy in isolation, but a distinguisher that remains strong enough to support cryptanalytic procedures while becoming materially cheaper to train and integrate into attacks.

Rotational-XOR cryptanalysis of Simon-like ciphers (Chapter 4). We extended RX cryptanalysis to AND-RX block ciphers, focusing on the Simon and Simeck families. We derived closed-form expressions for the expected probability of RX-difference propagation through AND-RX rounds and developed an SMT-based model to search for RX characteristics. The results include RX characteristics for multiple Simeck variants, yielding distinguishers for up to 20, 27, and 35 rounds with probabilities of 2^{-26}, 2^{-42}, and 2^{-54}, respectively. For Simon32, the best characteristic covered 11 rounds with probability 2^{-24}. The analysis also highlighted practical differences between Simon and Simeck under RX differences, including the influence of key schedules and rotation parameters on characteristic propagation. Here, efficiency matters because the method is used as a search and evaluation tool over a large RX parameter space: even when a distinguisher is not optimal on raw accuracy alone, it is still valuable if it enables wider exploration of cipher variants, rotation settings, and full-key behaviour at practical cost.

Privacy-preserving sentiment analysis with homomorphic encryption (Chapter 5). We proposed a privacy-preserving sentiment-analysis method

based on the CKKS homomorphic encryption scheme applied to a pre-trained deep learning model. The method splits inference into a client-side attention mechanism and a server-side prediction head: the client encrypts intermediate representations before transmission, while the server evaluates a simplified recurrent layer and linear activation on ciphertexts. The client then decrypts the output and applies a sigmoid function to obtain the final sentiment score. We evaluated the approach on the IMDb dataset and achieved an accuracy of 70.73%, demonstrating a practical balance between privacy protection and computational overhead. In this chapter, efficiency is a deployment constraint rather than a secondary preference: a slight reduction in predictive performance is justified if it yields an encrypted-inference pipeline with manageable depth, memory use, latency, and communication cost.

Together, these contributions provide practical methods and empirical evidence for (i) improving the efficiency of ML-assisted cryptanalytic workflows, (ii) strengthening our understanding of RX behaviour in Simon-like designs, and (iii) enabling encrypted inference for natural-language sentiment analysis. More importantly, they show that the central trade-off in this thesis is not efficiency versus quality in the abstract, but practical usefulness under constraint: a method is valuable when the reduction in cost is matched by retaining enough cryptanalytic or predictive utility to remain meaningful in its intended setting.

6.2 Limitations

The results in this thesis should be interpreted in light of several limitations.

First, the experimental evaluations focus on a specific set of primitives, datasets, and model configurations. For example, the PD-ML distinguisher experiments target SPECK32/64 and are based on the data-generation and training setup described in Chapter 3. Extending the approach to additional primitives and attack settings would strengthen conclusions about generality and help identify which cipher structures benefit most from bit selection.

Second, the RX-cryptanalysis experiments and searches were constrained by available computational resources. Larger-scale exploration (for example, broader searches over parameters or longer-round settings) could reveal additional characteristics or enable stronger empirical validation of the derived probability estimates.

Third, the privacy-preserving sentiment-analysis pipeline demonstrates feasi-

bility, but the overhead of homomorphic encryption remains substantial. The design choices made to control multiplicative depth and ciphertext growth necessarily trade expressiveness for efficiency; further optimisation is needed to improve latency and scalability for deployment at larger model sizes or higher-throughput settings.

Finally, some parts of the work assume standard security and systems conditions (for example, correct key management and benign execution environments). While these assumptions are typical for the threat models considered, additional study under more adversarial or deployment-realistic models would strengthen the robustness of end-to-end security claims.

6.3 Future Work

The thesis suggests several concrete directions for future research. In particular, future work should continue to make explicit what counts as an acceptable efficiency–performance trade-off in each setting, so that gains in speed, memory, or search cost are always tied to practical cryptanalytic or deployment value.

A natural extension of the PD-ML approach is to evaluate bit-selection and partial distinguishers across a wider range of symmetric primitives and attack settings, including different block sizes, round functions, and differential trails. More advanced feature-selection methods and architectures (for example, learned masking or sparsity-inducing training objectives) may further reduce cost while maintaining accuracy.

For RX cryptanalysis, future work can refine and scale the deep-learning-based distinguisher framework and integrate it more tightly with characteristic search. One challenging direction is key recovery from RX distinguishers: because RX attacks are related-key in nature, bridging the gap from distinguishing to key recovery requires careful handling of key schedules and modelling assumptions. Systematic study of how key schedules and rotation constants affect RX propagation may also inform the design of AND-RX ciphers with improved resistance to RX attacks.

For privacy-preserving sentiment analysis, improving the efficiency of encrypted inference remains central. Hybrid designs that combine homomorphic encryption with complementary techniques such as secure multi-party computation or differential privacy may yield better trade-offs between latency, bandwidth, and security. Extending the approach to other text tasks and datasets,

and evaluating performance under deployment constraints, would further clarify practical applicability.

More broadly, future work should continue to stress-test the proposed methods under richer threat models and realistic operational settings. As machine learning and cryptography both evolve rapidly, iterative evaluation and careful threat modelling will remain essential for developing secure and trustworthy systems.

References

[20] Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, April 2020. Microsoft Research, Redmond, WA.

[AA16] Martín Abadi and David G Andersen. Learning to protect communications with adversarial neural cryptography. *arXiv preprint arXiv:1610.06918*, 2016.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Norx: parallel and scalable aead. In *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II 19*, pages 19–36. Springer, 2014.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[AL13] Martin R Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In *Selected Areas in Cryptography: 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers 19*, pages 1–15. Springer, 2013.

[AL16] Tomer Ashur and Yunwen Liu. Rotational cryptanalysis in the presence of constants. *IACR Transactions on Symmetric Cryptology*, pages 57–70, 2016.

[Bar09] Gregory Bard. *Algebraic cryptanalysis*. Springer Science & Business Media, 2009.

[Bas11] Sandipan Basu. International data encryption algorithm (idea)–a typical illustration. *Journal of global research in Computer Science*, 2(7):116–118, 2011.

[BB22] Anubhab Baksi and Anubhab Baksi. Machine learning-assisted differential distinguishers for lightweight ciphers. *Classical and*

Physical Security of Symmetric Key Cryptographic Algorithms, pages 141–162, 2022.

[BBD+23] Anubhab Baksi, Jakub Breier, Vishnu Asutosh Dasu, Xiaolu Hou, Hyunji Kim, and Hwajeong Seo. New results on machine learning-based distinguishers. *IEEE Access*, 2023.

[BBDH21] Anubhab Baksi, Jakub Breier, Vishnu Asutosh Dasu, and Xiaolu Hou. Machine learning attacks on speck. *Security and Implementation of Lightweight Cryptography (SILC)*, pages 1–6, 2021.

[BBG+25] Emanuele Bellini, Rocco Brunelli, David Gerault, Anna Hambitzer, and Marco Pedicini. Generic partial decryption as feature engineering for neural distinguishers. In *Progress in Cryptology – LATINCRYPT 2025*, Lecture Notes in Computer Science, pages 373–398. 2025.

[BBP22] Norica Băcuieti, Lejla Batina, and Stjepan Picek. Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher. In *International Conference on Applied Cryptography and Network Security*, pages 809–829. Springer, 2022.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Miss in the middle attacks on idea and khufu. In *Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings 6*, pages 124–138. Springer, 1999.

[BÇMT18] Lawrence Bassham, Çağdaş Çalık, Kerry McKay, and Meltem Sönmez Turan. Submission requirements and evaluation criteria for the lightweight cryptography standardization process. *US National Institute of Standards and Technology*, 2018.

[Ber14] Daniel J. Bernstein. Caesar call for submissions. `http://competitions.cr.yp.to/caesar-call.html`, 01 2014. Accessed: date-of-access.

[BGHR22] Emanuele Bellini, David Gerault, Anna Hambitzer, and Matteo Rossi. A cipher-agnostic neural training pipeline with automated finding of good input differences. *Cryptology ePrint Archive*, 2022.

[BGL+21] Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Conditional differential-neural cryptanalysis. *IACR Cryptol. ePrint Arch.*, 2021:719, 2021.

[BGPT21] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. In

Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40, pages 805–835. Springer, 2021.

[Bih95] Eli Biham. On matsui's linear cryptanalysis. In *Advances in Cryptology—EUROCRYPT'94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings 13*, pages 341–355. Springer, 1995.

[Bis19] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[BK20] Seunggeun Baek and Kwangjo Kim. Recent advances of neural attacks against block ciphers. In *2020 Symposium on Cryptography and Information Security (SCIS 2020)*. IEICE Technical Committee on Information Security, 2020.

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, pages 450–466. Springer, 2007.

[BOP06] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151, 2006.

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 384–397. Springer, 2002.

[BR21] Emanuele Bellini and Matteo Rossi. Performance comparison between deep learning-based and conventional cryptographic distinguishers. In *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 3*, pages 681–701. Springer, 2021.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4:3–72, 1991.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. *cryptology eprint archive*, 2013.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck lightweight block ciphers. In *Proceedings of the 52nd annual design automation conference*, pages 1–6, 2015.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[BVLC16] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in arx: application to block cipher speck. In *International Conference on Fast Software Encryption*, pages 289–310. Springer, 2016.

[CCH10] Claude Carlet, Yves Crama, and Peter L Hammer. Boolean functions for cryptography and error-correcting codes., 2010.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[Cho15] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[CLC12] Jung-Wei Chou, Shou-De Lin, and Chen-Mou Cheng. On the effectiveness of using state-of-the-art machine learning techniques to launch cryptographic distinguishing attacks. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, pages 105–110, 2012.

[Col] Google Colaboratory. Google colaboratory. `https://colab.`

research.google.com/.

[CSYY23] Yi Chen, Yantian Shen, Hongbo Yu, and Sitong Yuan. A new neural distinguisher considering features derived from multiple ciphertext pairs. *The Computer Journal*, 66(6):1419–1433, 2023.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[CY21] Yi Chen and Hongbo Yu. Bridging machine learning and cryptanalysis via edlct. *Cryptology ePrint Archive*, 2021.

[CZX+23] Siwei Chen, Mingming Zhu, Zejun Xiang, Runqing Xu, Xiangyong Zeng, and Shasha Zhang. Rotational-xor differential rectangle cryptanalysis on simon-like ciphers. In *Topics in Cryptology – CT-RSA 2023*, 2023. Also available as Cryptology ePrint Archive, Report 2023/178.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2. *Submission to the CAESAR Competition*, 5(6):7, 2016.

[DH76] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112, 1976.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[DS98] Norman R Draper and Harry Smith. *Applied regression analysis*, volume 326. John Wiley & Sons, 1998.

[DSD01] Morris Dworkin, NATIONAL INST OF STANDARDS, and TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV. Recommendation for block cipher modes of operation. methods and techniques. 2001.

[EGP23] Amirhossein Ebrahimi, David Gerault, and Paolo Palmieri. Deep learning-based rotational-xor distinguishers for and-rx block ciphers: Evaluations on simeck and simon. In *International Conference on Selected Areas in Cryptography*, pages 429–450. Springer,

2023.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[ERP22] Amirhossein Ebrahimi, Francesco Regazzoni, and Paolo Palmieri. Reducing the cost of machine learning differential attacks using bit selection and a partial ml-distinguisher. In *International Symposium on Foundations and Practice of Security*, pages 123–141. Springer, 2022.

[ES03] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[FAL25] Marcos Florencio, Luiz Alencar, and Bianca Lima. An end-to-end homomorphically encrypted neural network. arXiv:2502.16176, 2025.

[FHL+20] Qi Feng, Debiao He, Zhe Liu, Huaqun Wang, and Kim-Kwang Raymond Choo. Securenlp: A system for multi-party privacy-preserving natural language processing. *IEEE Transactions on Information Forensics and Security*, 15:3709–3721, 2020.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. Milp-based automatic search algorithms for differential and linear trails for speck. In *Intl. Conf. on Fast Software Encryption*, pages 268–288. Springer, 2016.

[GBDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.

[GD95] Patrick Gallagher and Acting Director. Secure hash standard (shs). *FIPS PUB*, 180:183, 1995.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GHHP24] David Gérault, Anna Hambitzer, Moritz Huppert, and Stjepan

Picek. Survey: 6 years of neural differential cryptanalysis. Cryptology ePrint Archive, Report 2024/1300, 2024. Last updated 2025-08-25.

[GHZ+18] Aidan N Gomez, Sicong Huang, Ivan Zhang, Bryan M Li, Muhammad Osama, and Lukasz Kaiser. Unsupervised cipher cracking using discrete gans. *arXiv preprint arXiv:1801.04883*, 2018.

[GLN22] Aron Gohr, Gregor Leander, and Patrick Neumann. An assessment of differential-neural distinguishers. *Cryptology ePrint Archive*, 2022.

[GMW19] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[GO21] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39*, pages 150–179. Springer, 2019.

[Gre17] Sam Greydanus. Learning the enigma with recurrent neural networks. *arXiv preprint arXiv:1708.07576*, 2017.

[GY23] Chen Gao and Jia Yu. Securerc: a system for privacy-preserving relation classification using secure multi-party computation. *Computers & Security*, 128:103142, 2023.

[Hab21] Ivan Habernal. When differential privacy meets nlp: The devil is in the detail. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1522–1528, 2021.

[HK70] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[HL15] Rakel Haakegaard and Joanna Lang. The elliptic curve diffie-hellman (ecdh). *Online at https://koclab. cs. ucsb. edu/teaching/ecc/project/2015Projects/Haakegaard+ Lang. pdf*, 2015.

[Hou07] Russ Housley. Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS). RFC 5084, November 2007.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014*, volume 8616 of *LNCS*, pages 554–571. Springer, 2014.

[HTG17] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *CoRR*, abs/1711.05189, 2017.

[HWW22] Fanliang Hu, Huanyu Wang, and Junnian Wang. Multi-leak deep-learning side-channel analysis. *IEEE Access*, 10:22610–22621, 2022.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JL17] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (eddsa). Technical report, 2017.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[JNPS21] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. The deoxys aead family. *Journal of Cryptology*, 34(3):31, 2021.

[JS15] Poonam Jindal and Brahmjit Singh. Rc4 encryption-a literature survey. *Procedia Computer Science*, 46:697–705, 2015.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. Technical report, 1997.

[KEI+22] Hayato Kimura, Keita Emura, Takanori Isobe, Ryoma Ito, Kazuto Ogawa, and Toshihiro Ohigashi. Output prediction attacks on block ciphers using deep learning. In *International Conference on Applied Cryptography and Network Security*, pages 248–276.

Springer, 2022.

[KMS02] Alexander Klimov, Anton Mityagin, and Adi Shamir. Analysis of neural cryptography. In *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*, pages 288–298. Springer, 2002.

[KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational cryptanalysis of arx. In *Fast Software Encryption: 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers 17*, pages 333–346. Springer, 2010.

[KNR14] Dmitry Khovratovich, Ivica Nikolić, and Christian Rechberger. Rotational rebound attacks on reduced skein. *Journal of cryptology*, 27:452–479, 2014.

[Knu] Lars R Knudsen. Truncated and higher order differentials. In *International Workshop on Fast Software Encryption*.

[Knu95] Lars R Knudsen. Truncated and higher order differentials. In *Fast Software Encryption: Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings 2*, pages 196–211. Springer, 1995.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In *Fast Software Encryption: 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers 9*, pages 112–127. Springer, 2002.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233, 1994.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCX+25] Chengcai Liu, Siwei Chen, Zejun Xiang, Shasha Zhang, and Xiangyong Zeng. Enhancing deep learning-based rotational-xor attacks on lightweight block ciphers simon32/64 and simeck32/64. arXiv:2511.06336, 2025.

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 619–631. ACM, 2017.

[LLA+22] Jinyu Lu, Yunwen Liu, Tomer Ashur, Bing Sun, and Chao Li. Improved rotational-xor cryptanalysis of simon-like block ciphers. *IET Information Security*, 16(4):282–300, 2022.

[LLS+22] Jinyu Lu, Guoqiang Liu, Bing Sun, Chao Li, and Li Liu. Improved (related-key) differential-based neural distinguishers for simon and simeck block ciphers. *Cryptology ePrint Archive*, 2022.

[LLW17] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal differential trails in simon-like ciphers. *IACR Transactions on Symmetric Cryptology*, pages 358–379, 2017.

[LMSV07] Elena C Laskari, Gerasimos C Meletiou, Yannis C Stamatiou, and Michael N Vrahatis. Cryptography and cryptanalysis through computational intelligence. In *Computational Intelligence in Information Assurance and Security*, pages 1–49. Springer, 2007.

[Loh11] Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[M+67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Advances in Cryptology—EUROCRYPT'93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings 12*, pages 386–397. Springer, 1994.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic.

Coding Thv, 4244:114–116, 1978.

[MDP+11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[MGP24] Amirhossein Ebrahimi Moghaddam, Buvana Ganesh, and Paolo Palmieri. Privacy-preserving sentiment analysis using homomorphic encryption and attention mechanisms. In Martin Andreoni, editor, *Applied Cryptography and Network Security Workshops - ACNS 2024 Satellite Workshops, AIBlock, AIHWS, AIoTS, SCI, AAC, SiMLA, LLE, and CIMSS, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II*, volume 14587 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2024.

[Mil85] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[MRTZ18] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[Nis92] Corporate Nist. The digital signature standard. *Communications of the ACM*, 35(7):36–40, 1992.

[OL22] Yu Ou and Lang Li. Side-channel analysis attacks based on deep learning network. *Frontiers of Computer Science*, 16:1–11, 2022.

[PAM+21] Jestine Paul, Meenatchi Sundaram Muthu Selva Annamalai, William Ming, Ahmad Al Badawi, Bharadwaj Veeravalli, and Khin Mi Mi Aung. Privacy-preserving collective learning with homomorphic encryption. *IEEE Access*, 9:132084–132096, 2021.

[Pea01] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[PMC+22] Debranjan Pal, Upasana Mandal, Mainak Chaudhury, Abhijit Das, and Dipanwita Roy Chowdhury. A deep neural differential distinguisher for arx based block cipher. *Cryptology ePrint Archive*, 2022.

[PPS14] Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. Big bias hunting in amazonia: Large-scale computation and exploitation of rc4 biases. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 398–419. Springer, 2014.

[PT20] Robert Podschwadt and Daniel Takabi. Classification of encrypted word embeddings using recurrent neural networks. In *PrivateNLP@ WSDM*, pages 27–31, 2020.

[PT21] Robert Podschwadt and Daniel Takabi. Non-interactive privacy preserving recurrent neural network prediction with homomorphic encryption. In *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, pages 65–70. IEEE, 2021.

[Pub12] FIPS Pub. Secure hash standard (shs). *Fips pub*, 180(4), 2012.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[Riv91] Ronald L Rivest. Cryptography and machine learning. In *International Conference on the Theory and Application of Cryptology*, pages 427–439. Springer, 1991.

[Riv92] Ronald Rivest. The md5 message-digest algorithm. Technical report, 1992.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[S+99] Data Encryption Standard et al. Data encryption standard. *Federal Information Processing Standards Publication*, 112, 1999.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.

[Sch15] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley; 20th edition, 2015.

[SDP+22] Sinem Sav, Abdulrahman Diaa, Apostolos Pyrgelis, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Privacy-preserving federated recurrent neural networks. *CoRR*, abs/2207.13947, 2022.

[So20] Jaewoo So. Deep learning-based cryptanalysis of lightweight block ciphers. *Security and Communication Networks*, 2020:1–11, 2020.

[SP97] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[SRC25] Rui-Tao Su, Jiong-Jiong Ren, and Shao-Zhen Chen. Improved framework of related-key differential neural distinguisher and applications to the standard ciphers. Cryptology ePrint Archive, Paper 2025/537, 2025.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher clefia. In *Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers 14*, pages

181–195. Springer, 2007.

[SSL+24] Dongsu Shen, Yijian Song, Yuan Lu, Saiqin Long, and Shujuan Tian. Neural differential distinguishers for GIFT-128 and ASCON. *Journal of Information Security and Applications*, 82:103758, May 2024.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 287–314. Springer, 2015.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WP14] Hongjun Wu and Bart Preneel. Aegis: A fast authenticated encryption algorithm. In *Selected Areas in Cryptography–SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers 20*, pages 185–201. Springer, 2014.

[WWHL18] Xuzi Wang, Baofeng Wu, Lin Hou, and Dongdai Lin. Automatic search for related-key differential trails in simon-like block ciphers based on milp. In *Information Security: 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings 21*, pages 116–131. Springer, 2018.

[WWL+25] Lingling Wu, Xu An Wang, Jiasen Liu, Yunxuan Su, Zheng Tu, Wenhao Liu, Haibo Lei, Dianhua Tang, Yunfei Cao, and Jianping Zhang. Homomorphic encryption for machine learning applications with CKKS algorithms: A survey of developments and applications. *Computers, Materials & Continua*, 85(1):89–119, 2025.

[WWS24] Gao Wang, Gaoli Wang, and Siwei Sun. A new (related-key) neural distinguisher using two differences for differential cryptanalysis. *IET Information Security*, 2024(1):4097586, 2024.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.

[YK21] Tarun Yadav and Manoj Kumar. Differential-ml distinguisher: Ma-

chine learning based generic extension for differential cryptanalysis. In *International Conference on Cryptology and Information Security in Latin America*, pages 191–212. Springer, 2021.

[YW24] Xue Yuan and Qichun Wang. Improving differential-neural distinguisher for simeck family. Cryptology ePrint Archive, Paper 2024/2002, 2024.

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In *Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 307–329. Springer, 2015.

[ZLW+22] Kai Zhang, Xuejia Lai, Lei Wang, Jie Guan, Bin Hu, Senpeng Wang, and Tairong Shi. Rotational-xor differential cryptanalysis and an automatic framework for and-rx ciphers. *IEEE Transactions on Information Theory*, 2022.

[ZXF+20] Libang Zhang, Xinpeng Xing, Junfeng Fan, Zongyue Wang, and Suying Wang. Multilabel deep learning-based side-channel attack. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(6):1207–1216, 2020.

[ZXW21] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. GALA: greedy computation for linear algebra in privacy-preserved neural networks. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[ZYH+25] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In *Network and Distributed System Security Symposium (NDSS) 2025*, 2025.