Cryptanalysis of Round-Reduced ASCON powered by ML

Dushica Jankovikj
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, Republic of North Macedonia
dushica.jankovikj@students.finki.ukim.mk

Hristina Mihajloska Trpceska
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, Republic of North Macedonia
hristina.mihajloska@finki.ukim.mk

Vesna Dimitrova
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, Republic of North Macedonia
vesna.dimitrova@finki.ukim.mk

***Abstract*— Our research focuses on attacking Ascon, a lightweight block cipher presented as a candidate in the NIST Lightweight Cryptography Standardization Process. This block cipher provides authenticated encryption with associated data functionalities. We propose a cryptanalysis model based on deep learning (DL), where the goal is to predict plaintext bits given knowledge of the ciphertext and other publicly known cipher input parameters. Our experiments show that such known-plaintext attacks can be successfully executed on a round reduced version of the cipher stripped of the finalization phase. This, in turn, validates the theoretical results. Cryptographic algorithms are complex for the purpose of security and cannot be easily broken by an ML model in their regular form (not reduced). We explore multiple dataset generation techniques, model design, and training hyperparameters.**

Keywords—lightweight cryptography, cryptanalysis, known plaintext attack, machine learning, deep learning

I. INTRODUCTION

Cryptanalysis studies the inner workings of ciphers, the ciphertexts they produce, and cryptosystems in general. The goal is understanding the inner workings of information systems, uncovering hidden aspects of their operation, and then discovering or developing techniques toward breaking them or exposing their weaknesses. This discipline is not inherently malicious, it is often used to discover the use cases when a breach in a security system can be achieved so it can be prevented in the future by improving the system. Cipher standardization includes extensive security evaluations where any attacks the cipher might fall weak to are discussed. Attackers performing cryptanalysis hope to obtain access to encrypted data and to infer the raw data value without knowledge of the actual cryptographic key.

Cryptanalysis of block ciphers is not a new idea [1]. This has been persistently studied, and as a result many cryptanalytic techniques have been proposed. Most papers list differential, linear and integral cryptanalysis, attacks which exploit the algebraic degree, meet-in-the middle attack. Legacy cryptanalytic technologies are known to require a great deal of resources, whether that be time, memory, known plaintexts.

Due to the fact that machine learning (ML) and cryptanalysis share many of the same concepts and concerns, they have been described as "sister fields" [2]. This relationship originates from the similarity in the goal of both disciplines: to learn some unknown function given pairs of input and output values (in the cryptanalysis scenario, these values can be ciphertext and plaintext pairs). We propose a ML based approach to cryptanalysis, more precisely, within the deep learning (DL) realm.

Encryption methods are considered lightweight if they are characterized by small memory requirements and computational complexity. Their use is suitable in constrained devices (low memory, power or computation resources) where performance of heavier cryptographic standards is not acceptable. Due to the lower complexity of operation as a requirement, these types of ciphers seem eligible for cryptanalysis using machine learning.

II. ASCON

A. Notation

This section defines some basic notation which will be used in the following sections. Here we define that $\oplus$ stands for XOR, right and left bit rotations will be denoted as $\gg$ and $\ll$ respectively, while a||b represents the concatenation of two bit strings a and b.

B. Ascon

Ascon is a family of ciphers equipped to handle authenticated encryption with associated data and hashing [3]. Our focus falls on the encryption functionality of the cipher.

Ascon uses a duplex-sponge-based mode of operation for authenticated encryption. At the core of operation stands a lightweight permutation, which is used for all family members. The permutation is used to apply substitution-permutation network (SPN) based transformations iteratively using multiple rounds. Input parameters are key, tag and nonce where different modes support different bit lengths of these parameters. Namely Ascon branches out into two versions, Ascon128 and Ascon96, both of which have differences in the parameters and the achieved security levels. The recommended choice is said to be Ascon128 and it is the focus of the analysis in this paper. This mode operates on a plaintext block size of 64 bits.

The cipher operates using a 320 bit internal state which gets updated and transformed by the sponge function. The permutation gets applied in a=12 rounds (for initialization and finalization phases) and b=8 rounds (for encryption process). Encryption happens when the input plaintext block is XOR-ed with the first 64 bits of the state. The part of the state that "meets" the plaintext block (the first 64 bits of the state) are called the rate r bits, while the rest are called the capacity bits. Cipher operations can be divided in four phases:

1. Initialization: The bits of the internal state are populated using the key K (whose number of bits are represented by k) and nonce N. More precisely, the internal state is filled by the vector IV||K||N where IV is some initialization vector and calculated as: k||r||a||b||0160-k. After the internal state registry of bits is no longer empty, initialization can begin. The 320 bit internal state is initialized using 12 rounds of the Ascon permutation p.
2. Associated Data processing: updates the state using associated data blocks Ai.

3. Plaintext Processing: injects plaintext blocks Pi into the state and extracts ciphertext blocks Ci. Ci can be calculated as Pi ⊕ Sr, where Sr presents the r bits from an internal state. After this operation is performed, the internal bits in Sr accept the value of Ci themselves.
4. Finalization: injects the key K into the capacity bits of the state (Sc). After this, finalization is carried out similarly to initialization - in 12 permutation rounds. Upon completion, an authentication tag T is produced with authenticates both the associated data and the encrypted message.

As mentioned, the internal state S of the cipher consists of 320 bits. It is divided into 5 "words" of 64 bits each. We will denote them S1, S2, S3, S4, S5; where S1 is equivalent to Sr. The Ascon permutation module consists of three transformations that affect the state as follows:

1. Addition of Round Constants: XOR-s a round specific 1-byte constant to the state word S3. The round constants exist within the cipher stored in a lookup table [3].
2. Nonlinear Substitution Layer: applies a 5-bit S-box 64 times in parallel in a bit-sliced fashion (vertically, across words) [4]. The S-box mapping rules are presented in the official cipher specification [3].
3. Linear Diffusion Layer: provides diffusion within each state word by XOR -ing different rotated copies of each word (horizontally, within each word). The transformations for each state word are shown below.

$$S1 = S1 \oplus (S1 \gg 19) \oplus (S1 \gg 28)$$
$$S2 = S2 \oplus (S2 \gg 61) \oplus (S2 \gg 39)$$
$$S3 = S3 \oplus (S3 \gg 1) \oplus (S3 \gg 6)$$
$$S4 = S4 \oplus (S4 \gg 10) \oplus (S4 \gg 17)$$
$$S5 = S5 \oplus (S5 \gg 7) \oplus (S5 \gg 41)$$

The permutation in Ascon is used during the initialization, when it is applied in 12 rounds. During encryption, it is applied in 8 rounds in between each plaintext being processed.

C. Round reducing ciphers

In the world of ciphers, rounds refer to the number of times some internal cipher function is applied to the data. For Ascon, rounds are used when the cipher's permutation is applied to the cipher state in different stages of the cipher operation. As mentioned, initialization applies the permutation 12 times (variable a), while the encryption uses the same permutation 8 times (variable b). Multiple applications of some function that mix up and shift the data bits ensure better confusion of the bits - security through complexity of the transformation. Namely, by mixing the data repeatedly (multiple rounds), it gets harder to simply return the data to its internal state. For Ascon, multiple iterations of the SPN transformations lead to a state where the transformation becomes layered, and the result is hard to express in terms of a single formula where all the modifications applied to a specific bit through the rounds can be tracked.

Cipher specifications are created with a thick background of experimentation and proof that defines the specific number of rounds used for the cipher operation. The number of rounds is chosen with security, but also with performance in mind. For example, Ascon with more than 8 rounds in the encryption permutation module might be more secure, but not secure to the point where the performance hindering is worth it. This is a lightweight cipher after all, so performance optimization is key.

Our cryptanalysis efforts focus on a round reduced Ascon, but only in the encryption phase. This means, the initialization carries out using all the rounds it is meant to, but for the encryption permutation module we experimented with using 1, up to 8 rounds.

D. Simplified Ascon

Aside from using a round reduced Ascon, which is common in cryptanalysis attempts, we additionally strip the cipher of the initialization phase. By using an empty associated data input, that section can also be considered avoided. The figure below shows the sections that aren't relevant to our work as grayed out. Beginning with an initialized state, the first plaintext block P1 is injected in the first state word. This modified the state itself but outputs C1 as a result as well. Then the state is processed using the encryption permutation pb (where b is reduced). Next, P2 is XOR -ed to the first state word to obtain C2.

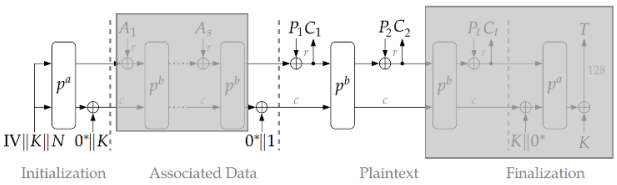

Fig. 1. Ascon encryption (simplified version omits greyed out processes)

We are focused on discovering the relationship between P2 and C2, thus producing plaintext recovery attacks. Additionally, if knowledge of P1 and C1 is assumed, knowing the first state word after initialization is also assumed.

E. Traditional cryptanalysis of Ascon

The security analysis of Ascon [5] proposed by its creators gives a detailed inspection of the possible weaknesses when rounds are being reduced. In their work they apply cube-like, differential, and linear cryptanalysis to evaluate the security. The focus falls on practical key-recovery attacks on round-reduced versions of Ascon-128. The initialization phase was round reduced to 5 out of 12 rounds. Theoretical key recovery attacks were proven as feasible for up to 6 rounds of initialization. From the aspect of forgery, a practical forgery attack was presented for 3 rounds of the finalization, while a theoretical forgery attack was presented for 4 rounds finalization and zero-sum distinguishers for the full 12-round Ascon permutation.

They present attempts of linear cryptanalysis of Ascon which can be considered first in the related research efforts and prove the bounds on the minimum number of active s-boxes for the Ascon permutation.

III. MACHINE LEARNING FRAMEWORK

A. Machine learning concepts

Machine learning (ML) is a term that can be used to describe algorithms that can perform intelligent predictions based on a data set. The dataset part is necessary since ML seeks to automatically learn meaningful relationships and patterns from examples and observations.

The algorithms do multiple passes over the data which enables them to iteratively learn and uncover hidden relationships and complex patterns in the data points.

In literature, the algorithms are often referred to as models. Choosing the appropriate model for a learning task depends on the complexity of the function, and the dimensionality and volume of data.

Artificial neural networks represent a family of algorithms whose internal processing units are mathematical functions. These models take inspiration from the neurons in the human brain, and so, the processing units are, each connection transmitting signals like synapses in the brain. The strength of these signals depends on an internal weight for each model, which is updated and adjusted through the learning process. This affects subsequent processing in the network since the connected neurons only process signals if the signal strength exceeds a certain threshold. An activation function defines this threshold.

Neurons aren't chaotically distributed throughout the model; instead, they are organized into layers. The neurons within a layer do not communicate; instead, they are linked only to neurons in the previous and following layer. Depending on the type of neurons, the information (signal) flow can be one-directional (data flows from one layer to the next), bi-directional (data flows in different directions). Data enters the model through the input layer and exists in the form of predictions through the output layer. The layers in between are hidden and are tasked with learning a non-linear mapping between the input and output layers.

Deep learning (DL) is a machine learning technique which uses deep neural networks (DNN) that contain more than one hidden layer. These networks follow deeply nested architectures and can consist of advanced neurons with multiple activation functions and more complex internal functions. The simplest forms of ML can estimate and learn simple linear functions, while complex neural networks can be used to estimate complex functions. DL uses multiple layers to progressively extract higher level learnable features from the input data and as a result is capable of learning nonconvex and nonlinear functions [6].

B. Machine learning in cryptanalysis

The closeness of cryptanalysis and machine learning has been theoretically established for three decades [3], but even so, the research community has been scarce of practical efforts which exploit more possibilities this similarity offers. For the most part, practical work in ML driven side channel analysis (SCA) tasks has shown as very valuable. The earliest efforts utilized support vector machines (SVM) [7], but recently DL enhanced SCA has gained a lot of traction.

Ghor's work in [2] introduces a ML based cryptanalysis strategy which later inspires further work in the domain. His work uses DL mechanisms to produce a neural based distinguisher tasked to perform key recovery attacks on the lightweight block cipher Speck [8].

The authors in [9] proposes a detailed analysis of the functionality of the new neural distinguisher who was thus far known to work as a black box. Studying the classified sets led to the conclusion that the neural distinguisher relies on the differential distribution on the ciphertext pairs. Moreover, they propose a ML based distinguisher which performs similarly using simple standard machine learning tools.

The other authors, [6] proposes a DL based cryptanalysis model whose task is to predict the key of block ciphers given known (plaintext, ciphertext) pairs. Their experiments show that the DL based approach successfully recovers the key bits in a reduced key space scenario. When key space restriction is not applied, the attacks are not successful unless the ciphers are round reduced. Their work is focused on lightweight block ciphers: simplified DES, Simon, and Speck.

Perushevska et al. in [10] apply DL for the cryptanalysis of the DES algorithm. The model uses (ciphertext, plaintext) pairs during training to learn to predict the plaintext. DES works in 16 rounds; their work analyses the full implementation and a reduced-round implementation with only one round. Also, in [11] they implement a deep neural network to perform a known-plaintext attack on AES. The goal is to recover the bits of the plaintext. Their work encompasses a wide range of experiments using different key sizes and modes of operation on AES. The results show that the proposed approach can restore the bits in the whole data set with a probability of more than 98%, and more than half of the plaintext bytes with a probability of 99%.

IV. METHODOLOGY

A. Prediction task

Our aim is to construct a model which would be able to predict the plaintext Pi given the ciphertext Ci. This effectively is a known plaintext attack where using pairs (Pi, Ci) we aim to train a DL model. The model is then tested in order to evaluate the number of plaintext bits it can correctly guess when provided with the ciphertext bits as input.

B. Dataset generation

A cipher algorithm is a sequence of operations which can roughly be explained as a very complex mathematical formula where the end goal is to conceal and obscure the input data. However, in order to conduct any data transformation, a few other elements are needed. These elements come in the form of raw input data which introduces randomness to the process since the inner workings of a cipher are not secret. A key and nonce are necessary to initialize the state of Ascon, this is a necessary step before any data concealing can even begin.

Encryption is already a complex nonlinear data transformation and as such it is not straightforward to learn. We keep this in mind when generating a dataset to train the model with. By carefully crafting the dataset and placing some restrictions and rules upon it, we ensure that aside from

the complexity of the algorithm, the input data is not entirely random and all over the place. That can be achieved by using a dataset where the inputs have small variability: any input i does not differ from any other input j by more than x bits (where x is some small number). This way, the model can gradually learn how small differences in the input manifest in the output. The same key was used for the creation of one dataset, since we want to simulate it as an ingrained part of the cipher itself, and not as a randomizing input parameter.

Dataset generation begins with the generation of one random key which will be used throughout the generation process. For the dataset creation, we created 1000 low variability ciphertexts and 1000 low variability nonces and then we combined each ciphertext with all the nonces from the nonces dataset. This resulted in a dataset with 1 million pairs of type (ciphertext, nonce). For each pair, the nonce and the predefined fixed key were used to initialize the cipher state. After initialization, associated data comes into play (mentioned in the introduction). However, the associated data affects the internal state, making it go through more rounds of permutation. We decided to eliminate this step (since it is optional) in order to simplify the processing as much as possible, so we didn't use associated data input (left it empty). This still does a change to the state, and 1 is XOR-ed to the last bit of the state. After the initialization, one is able to see the existing state of the cipher.

Ascon encrypts data by XOR-ing the data block (64 bits) to the first word of the state (also 64 bits, so no need for padding): Pi $\oplus$ S1. Knowing the plaintext-ciphertext pair produced by this XOR reveals the first word of the state itself.

In our dataset generation we wish to focus on the encryptions produced when the cipher state has gone through encryption permutation rounds. This is possible by skipping the first plaintext block (since it only meets with the initialized state that doesn't get affected by the encryption permutation) and focusing on the second block of the plaintext. We set the first block of the plaintext to all 0's. Knowing this, it can be assumed that the first word of the state is revealed as well, since we controlled the first input block. This is a subject to a later discussion.

Now, moving on to a phase where the encryption permutation module has been executed and using b encryption rounds, the permutation was applied to the state. Ascon applies the permutation in b=8 rounds. We experimented with round reduced versions which use from 1 up to 8 rounds, and for each round option we generated the next cipher state. This is important since XOR-ing the state to the generated ciphertext in the pairs provides the plaintext in the equation. This might seem like a backwards way to generate (plaintext, ciphertext) pairs, but since the ciphertext is the input to the model, it has to be the starting point (generated first) so the variability across the dataset can be controlled and reduced.

The resulting dataset contains 1 million records, each with values: (ciphertext, nonce, plaintext, state). Since the nonces are repeated for the dataset, this can be considered as a simulation of a nonce misuse scenario. This is not a disruption of the rules for train test datasets - no input train data is repeated in the test data. Additionally, even if the number of unique ciphertexts in the dataset is 1000, the number of unique (ciphertext, nonce) input pairs is 1.000.000.

C. *Feature selection*

Hinting plaintext bits. The problem can be simplified further, by tasking the model to predict only a subset of the plaintext bits, while the other bits of the plaintext are revealed to it and are used as input when training. This captures the scenarios when chunks of plain data can be sniffed and uncovered, and only some parts remain unknown. In our work, we have experimented with revealing the first x bits of the plaintext and predicting the remaining bits, where x varies from 0 to 32.

Previous state. As hinted earlier, the state of the first word of the cipher after initialization is a sequence of 64 bits which might be revealed to the model. Just as the section above, this leads to branching in the experimentation department. Namely, one can use the state as additional information to the ciphertext and nonce in the task of predicting the plaintext. This state can be considered as completely unknown, partially revealed (a subset of the bits used as input to the ML model) or completely known.

D. *Applying machine learning*

The design of the models is responsible for the model performance. Specifying the hyperparameters of a model refers to defining the number of layers and neurons, activation function.

The number of layers, the number of nodes and the type of neurons in each hidden layer are hyperparameters that control the topology of the network. In our work we used a deep neural network with fully connected (dense) layers whose inside neurons connect to every neuron in the preceding layer. At each node, the input data (signal) gets multiplied by the weights in a node. The resulting value is then transformed using an activation function to produce the output (or activation) of the node.

In order to use stochastic gradient descent with backpropagation of errors when training DNN's, the activation function must be chosen so that complex relationships between input and output data can be learnt all while avoiding oversaturation. The ReLU activation function outputs the input if it is positive, otherwise outputs zero. Neurons in all layers except the output layers were set to use this activation function.

The sigmoid activation function is nonlinear and transforms the input data into a value between 0.0 and 0.1. Inputs larger than 1.0 are mapped to the upper bound (1.0), while extremely small values converge to 0.0. This is the default function to use when a binary answer is required. This activation is applied in the output layer of the neural network; the final sigmoid activation function performs the binary classification, and it dictates the prediction for each output bit. The cutoff is done by a threshold in a way that if siqmoid(x) is greater than the threshold then the result is 1 (the default value for the threshold is 0.5).

1) *Optimizer*

During the compilation of the model, the Adam optimizer, and binary cross entropy loss were used. Adam is an optimization algorithm created as an extension to stochastic

gradient descent whose task is to iteratively update network weights based on input data. The algorithm works with the gradient and the squared gradient variables to calculate an exponential moving average [12].

2) Loss

Loss functions measure the deviation between a model's estimation of a value and the actual value. The loss function maps model predictions to their associated costs. Classification tasks aim to produce a discrete class prediction as output. Binary classification tasks are even simpler since only two classes exist. Our research is in the realm of binary classification, in the sense that we are predicting the values of bits in the plaintext which.

Entropy can be defined as a measure of randomness of the data during processing. It is a way to express the uncertainty associated with a given probability distribution. Difference between the randomness of two random variables is binary cross entropy. As the predicted probability diverges increasingly from the expected label the cross-entropy loss increases.

3) Train/test partitions

The models were trained on 80/20 train test datasets derived from the generated dataset (1 million records). Data is randomly picked from the dataset to generate the train and test datasets, and they are used equally for each model. This means, all models see the same training data, and get evaluated for the same test data.

4) Training Hyperparameters

Machine learning models have hyperparameters which control the learning process. These parameters can in turn affect the time to train and test the model, as well as the model's performance as well. Performance variations of the models are likely caused by a subset of all the hyperparameters, and we will be focusing on two of them below.

The batch size controls the accuracy of the estimate of the error gradient during training. Using larger batch sizes allows for speedups fueled by parallel computation using a GPU. A batch refers to the number of training samples used in an iteration

An epoch encompasses a single complete pass of all the training data through the machine learning algorithm. The model learns by updating the weight of nodes at the end of each epoch. Defining the number of epochs means controlling the number of times the model gets the opportunity to learn from the data.

If the batch size is defined big enough to include the entire training data, then the number of epochs will be equal to the number of iterations. This is usually not the case in practice since most models require more than one epoch to familiarize themselves with the data and learn. Generally, if d is the size of the dataset, while the number of epochs is e and batch size is b, the number of iterations *i* can be expressed as $i = d*e/b$

Choosing the hyperparameter values requires a deep understanding of the data supported by a lot of experimentation. The following section provides our insights about these hyperparameters.

V. EXPERIMENTS AND RESULTS

A. Setup

The neural networks in our research were implemented using the TensorFlow 2, Keras module. Initially we experimented with different neural network architectures, but in the end we settled for a fully connected neural network with 5 hidden layers between the input and output layers. The number of nodes is discussed below:

- Input layer: 1024 neurons.
- Hidden layers: The number of neurons in the layers ordered from input to output: 2048, 2048, 2014, 512, 256 fully connected neurons.
- Output layer: number of bits the model aims to learn and predict.

We experimented with networks with more layers and different neuron configurations, but that didn't seem to affect performance in a noticeable way. However, this is not eliminated as a direction for work in the future.

As mentioned, we used ReLU activation for the input and hidden layers, and sigmoid activation for the output layer. The Adam optimizer was used with the binary crossentrophy loss function.

Our experiments focused on estimating the model's performance when different options are used for the input data:

- Plaintext hinting: the number of bits of the plaintext revealed to the model on input, while the rest is used as target when classifying
- State word reveal: whether the state of the cipher from the previous step (right after initialization) is available as input.

The number of epochs significantly affected performance consistency; variable results were obtained from multiple runs using 50 epochs for training. Better stability is achieved when 500 epochs are used. We used 5000 for batch size.

B. Performance metric

The classification results are expressed in terms of "percentage of correctly guessed bits". This is an aggregate metric across all test prediction results. More precisely, the metric percentage of correctly guessed bits is calculated for each test sample and then averaged for all. We also include a minimum and maximum guessed bits within one test sample metric. The goal is to catch the instances with low accuracy even if their number is low and will not affect the global metric in a noticeable way.

C. Results

The experiments displayed in the tables below used 150 epochs for training using a batch size of 5000.

When hinting the first j plaintext bits to the model and using them as input we use the notation P[:j]. We experimented with different options for plaintext hinting:

- Revealing the first half of the plaintext P and using it as input during training, while the other half was the target for predictions.

- Revealing only the first 20 bits and predicting the remaining 44 bits.
- Revealing no plaintext bits and predicting the entire plaintext block.

State hinting: The experiments where the input is enriched with the state of the cipher immediately after initialization are denoted as CNS0. The ones where only the ciphertext bits and the nonce bits are used as input for the model are denoted CN.

In the table below, the column Rounds denotes the number of encryption rounds used when generating the dataset, while the column Accuracy displays the percentage of correctly guessed bits metric.

TABLE I. DL MODEL PERFORMANCE SUMMARY

Rounds	Plaintext hinting	Accuracy
1	P[:32]	**0.9947**
2	P[:32]	**0.9858**
3	P[:32]	**0.9989**
4	P[:32]	**0.9989**
5	P[:32]	**0.99997**
6	P[:32]	0.7736
7	P[:32]	0.5149
8	P[:32]	0.5339
1	P[:20]	**0.9879**
2	P[:20]	**0.9756**
3	P[:20]	**0.9747**
4	P[:20]	**0.9818**
5	P[:20]	**0.8907**
6	P[:20]	0.8532
7	P[:20]	0.5108
8	P[:20]	0.5339

The experiment results in the table above were obtained using full state hinting (all the bits of the previous state are revealed: CTNS0)

When using no plaintext hints and no state hints, the performance of the models drops quickly: the precision for predicting the plaintext bits for b=1 is somewhere around 0.7.

Our experimental results confirm that as the number of rounds increases, the harder it becomes to learn the inherent plaintext-ciphertext bit relationships. This aligns with the theoretical results, and hope lies in the future that deeper analysis and more advanced DL techniques will exceed the theoretical results.

VI. CONCLUSION

Our work marks an experimental approach to cryptanalysis of Ascon using DL models. We believe this to be the first ML-based cryptanalysis of the named cipher. While lightweight ciphers are not new to DL-based attacks, the previous attempts are focused on key recovery. We successfully predict the plaintext (with known plaintext attack) with an accuracy of 0.998 percent.

A significant drawback is that our efforts attack a very niche-specific dataset where the randomness and variability are brought to a minimum. The generated dataset has a nonce misuse situation, which does the job of reducing randomness but is not a likely real-world scenario. The cipher itself is stripped of the finalization phase, which introduces a lot of security and ensures data integrity.

In practice, the cipher assumes and enforces randomness by applying nonces correctly and functioning as a unit where all the phases are being executed. Such functions can be very complex, and ML would not be able to find meaningful relationships between the inputs and the outputs. In a non-restricted scenario, DL-based cryptanalysis fails to attack the block ciphers.

We plan to extend our work in the future by randomizing the datasets more and strengthening the model performance using techniques more focused on the model design and training. Hopefully, this inspires the application of more ML-based attacks in the domain of cryptanalysis in the future.

ACKNOWLEDGEMENT

This research was partially supported by Faculty of Computer Science and Engineering at “Ss Cyril and Methodius” University in Skopje.

REFERENCES

[1] A. Gohr, “Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning,” Bundesamt f ̈ur Sicherheit in der Informationstechnik (BSI), Germany, 2019
[2] R. L. Rivest, “Cryptography and Machine Learning,” Laboratory for Computer Science Massachusetts Institute of Technology Cambridge, MA 02139,
[3] C. Dobraunig, M. Eichlseder, F Mendel, and M. Schläffer, “Ascon v1.2,” May 2021
[4] P. Grabher, J. Großschädl, and D. Page, “Light-Weight Instruction Set Extensions for Bit-Sliced Cryptography”, In: Oswald, E., Rohatgi, P. (eds) Cryptographic Hardware and Embedded Systems – CHES 2008, 2008
[5] C. Dobraunig, M. Eichlseder, F Mendel, and M. Schläffer, “Cryptanalysis of Ascon,” March 2021
[6] U. M. Khokhar, “Deep Learning-Based Cryptanalysis of Lightweight Block Ciphers”, July 2020
[7] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, “Machine learning in side-channel analysis: a first study”, October 2011
[8] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, “The Simon and Speck Families of Lightweight Block Ciphers”, June 2013
[9] A. Benamira, D. Gerault, T. Peyrin, and Q. Q. Tan, “A Deeper Look at Machine Learning-Based Cryptanalysis”,
[10] M. Gj. Perusheska, H M. Trpceska, V. Dimitrova, “Deep Learning-Based Cryptanalysis of Different AES Modes of Operation”, March 2022
[11] S. Andonov, J. Dobreva, L. Lumburovska, S. Pavlov, V. Dimitrova, and A. Popovska-Mitrovikj “Application of Machine Learning in DES Cryptanalysis”, September 2020
[12] D. P. Kingma, J. Ba, “Adam: A Method for Stochastic Optimization”, 2014