A Multi-head CNN-based Side-channel Attack on the Energy-efficient DIZY Stream Cipher

George Attia[1], Martin Schmid[2], and Elif Bilge Kavun[1,3]

[1] Barkhausen Institut, Schweriner Str. 1, 01067 Dresden, Germany
[2] University of Passau, Innstr. 43, 94032 Passau, Germany
[3] Dresden University of Technology, 01069 Dresden, Germany
[1]{george.attia,elif.kavun}@barkhauseninstitut.org
[2]martin.schmid@uni-passau.de
[3]elif_bilge.kavun@tu-dresden.de

Abstract. Side-channel attacks exploit physical leakages, such as power consumption, to undermine the security of cryptographic ciphers. This paper explores deep learning-based side-channel attacks on a hardware implementation of a lightweight stream cipher, namely DIZY. We assess the effectiveness of convolutional neural networks in exploiting power consumption leakage to recover the secret key, focusing on a leakage model that targets the outputs of the cipher's substitution boxes at the resynchronization phase. Two network architectures are compared: one using separate networks for each substitution box group and a multi-head model with a shared backbone. The multi-head architecture demonstrates enhanced performance, achieving faster convergence and higher accuracy in key recovery. We further evaluate the impact of dataset size on model performance, demonstrating that the multi-head architecture maintains superior convergence even with reduced training data. The models were trained and tested on power traces from fixed- and variable-key datasets captured on a field-programmable gate array. The results show partial recovery of the upper 64 key bits with recovery of up to 11 out of 32 two-bit groups, and requiring as few as 514 traces on average. This is the first deep learning side-channel attack on a small internal state stream cipher such as DIZY, highlighting side-channel vulnerabilities in energy-efficient cryptographic systems.

Keywords: Stream cipher · Side-channel analysis · Deep learning · Convolutional neural networks.

1 Introduction

In symmetric cryptography, the majority of side-channel analysis research efforts have focused on block ciphers, with the Advanced Encryption Standard (AES) [16] serving as a benchmark to evaluate the effectiveness of various Side-Channel Attacks (SCAs), as also highlighted in previous studies [14,17]. This focus stems from the widespread adoption of AES as the standard for symmetric encryption,

which facilitates standardized comparisons across attack methodologies. However, stream ciphers, which operate by generating a continuous keystream to encrypt data, have received less attention in the SCA literature. This relative neglect can be attributed to the diverse designs of stream ciphers, including their internal state update mechanisms and initialization procedures [14].

The study of side-channel vulnerabilities in lightweight stream ciphers is particularly crucial in the context of distributed computing environments. These ciphers are optimized for resource-constrained devices, such as those in the Internet of Things (IoT) ecosystem. A lightweight algorithm, in this context, refers to a cryptographic primitive designed to minimize computational, memory, and power requirements while maintaining adequate security levels for constrained environments [5]. Unlike traditional computer systems, IoT devices are often deployed in physically accessible locations, making them susceptible to adversaries who can perform direct measurements. Therefore, understanding and mitigating SCA risks in lightweight stream ciphers becomes essential to protect sensitive data in these ubiquitous and vulnerable platforms. This work addresses this gap by exploring Deep Learning-based Side-Channel Attacks (DL-SCAs) on a recently proposed lightweight stream cipher, namely DIZY [11], which is based on a Substitution-Permutation Network (SPN) design that can also be applied to other cryptographic constructions such as block ciphers, hash functions, and sponge ciphers. This highlights both its vulnerabilities and the broader implications for security systems in resource-constrained environments.

It is important to highlight that attacking DIZY presents unique challenges compared to standard benchmarks such as AES. First, the specific parallel hardware architecture of DIZY executes 32 Substitution-boxes (S-boxes) simultaneously in a single clock cycle, compared to 16 in a standard parallel AES-128 implementation. This high degree of parallelism significantly increases algorithmic noise, reducing the Signal-to-Noise Ratio (SNR) for any single target S-box. Second, unlike AES where an 8-bit S-box output maps directly to an 8-bit key byte, DIZY's 5-bit S-box output corresponds to only a 2-bit key segment mixed with state bits. This necessitates a probabilistic approach to map the predicted S-box output back to the secret key.

The contributions of this work can be summarized as follows:

- A leakage model that exploits power consumption patterns from (S-box) operations in the cipher's first round of key initialization using a fixed Initialization Vector (IV) and multiple random initial states.
- A multi-head Convolutional Neural Network (CNN) architecture designed to predict the entire 160-bit internal state (divided into 32 five-bit groups), enabling simultaneous recovery of the upper 64 key bits.
- A comparative analysis evaluating the impact of different CNN architectures (multi-head compared with separate networks) on attack performance. This includes an evaluation of the impact of training dataset size on the attack's convergence rate, performed on both fixed- and variable-key power trace datasets captured from a Field Programmable Gate Array (FPGA).

2 DIZY Cipher Family Overview

In this section, we describe DIZY, which is a stream cipher family developed by Gül and Kara in 2023 [11]. It is a lightweight cryptographic algorithm designed for resource-constrained devices. The authors of DIZY define it as a Small Internal State Stream (SISS) cipher, which is a cipher with an internal state size smaller than twice its key size.

DIZY is designed as two variants: DIZY-80, with an 80-bit key and 120-bit internal state, and DIZY-128, with a 128-bit key and 160-bit internal state. Since the DIZY-128 variant offers stronger security due to its larger secret key and internal state sizes, this work only implements an SCA on the stronger DIZY-128 variant. Therefore, the description of the DIZY-80 variant is omitted from this section. However, note that our attack also works on the DIZY-80 variant.

DIZY is a round-based cipher that performs 15 unkeyed rounds as its state update function. Each round consists of four consecutive operations as follows:

- **AddRoundConstant:** Adds a 4-bit constant to each 5-bit group of the input state. The input state is divided into 5-bit groups, and the constant is XORed with the least significant 4 bits of each 5-bit group. There are 15 unique constants, one for each of DIZY's rounds. The constants are produced by a Linear Feedback Shift Register (LFSR) with a characteristic polynomial $x^4 + x + 1$ and an initial state (1, 0, 0, 0).
- **S-box:** Applies a substitution operation to each 5-bit group of the cipher's internal state.
- **MultMatrix:** Applies a linear transformation to the state to achieve diffusion. The 160-bit state is divided into four 40-bit groups, each of which is transformed using a predefined binary matrix $\boldsymbol{M}$.
- **Permute:** Divides the input state into 8 sub-blocks and permutes it as (0, 4, 1, 5, 2, 6, 3, 7).

DIZY's initialization is performed during the first 30 rounds of execution. First, the key is divided into two parts (upper 64 bits and lower 64 bits). The upper and lower key parts are added to the internal state before and after the first round, respectively. Afterwards, the remaining 14 rounds are executed. Similarly, the IV is divided into two parts and added to the internal state before and after the 16th round. Afterwards, 14 more rounds are executed, which complete the initialization phase.

During key/IV addition, the 64-bit Key/IV parts are split into 2-bit groups. Each 2-bit group is XORed with the most significant bits of each corresponding 5-bit group of the internal state of the cipher. The remaining 3 bits are passed on without changes.

After initialization, the cipher generates the pseudo-random output stream. The round function is performed 15 times as the state update function. After each state update, the 32 least significant $3i$-th state bits with $i = 0, ..., 31$ are output as pseudo-random bits. According to the cipher's authors, a maximum of 2^{32} 32-bit blocks can be output before the key/IV needs to be resynchronized.

3 Related Work

Most of the SCAs on hardware-optimized lightweight stream ciphers are attacks on ciphers developed under the eStream project, such as TRIVIUM [6], GRAIN [12], and MICKEY [3].

Several previous works performed SCAs on such ciphers using statistical methods like Differential Power Analysis (DPA) and Correlation Power Analysis (CPA) [10], [15], and [19]. More recent studies used advanced and augmented SCA methods. Chakraborty *et al.* [8] utilized a combined Fault Injection Attack (FIA) and DPA that targeted Grain. Kazmi *et al.* [13] suggested an Algebraic Side-Channel Attack (ASCA) on Trivium and Grain. Chakraborty *et al.* [9] attacked MICKEY-128 with a template attack that uses Least Squares Support Vector Machines (LS-SVM) for trace classification based on Hamming distance classes.

In the domain of DL-SCA, Zhang *et al.* [22] introduced a multilabel classification framework. Their approach decomposes the 8 bits of a single target byte (for example, an AES S-box output) into separate binary classification tasks within a single network to mitigate class imbalance. Our work diverges from this by addressing the parallelism inherent to the DIZY architecture. Rather than decomposing the bits of a single intermediate value, our proposed multi-head architecture utilizes a shared backbone to simultaneously learn representations for 32 distinct, parallel S-box operations, each predicted by a dedicated classification head. This structure enables the concurrent recovery of 32 independent key groups (spanning 64 bits) in a single attack pass.

Most recently, Schmid *et al.* [21] presented a non-profiled DPA on DIZY. They targeted the resynchronization phase (specifically rounds 16 and 17) to recover the internal state, requiring approximately 4000 power traces. While their approach achieves full state recovery, it necessitates a subsequent algebraic cryptanalysis step using SAT solvers to reverse the state and extract the original secret key. In contrast, our work employs a profiled Deep Learning approach targeting the very first round of initialization. This allows for the direct recovery of secret key bits from the S-box leakage without the mathematical complexity of solver-based key recovery. Furthermore, our multi-head model demonstrates convergence on specific key groups with significantly fewer traces (averaging 514 to 775 traces), highlighting the efficiency of profiled attacks in low-SNR environments.

All the mentioned works target the initialization phase since the secret key becomes entirely mixed within the internal state during the pseudo-random phase. In addition, all these works use traces with multiple (or chosen) IVs. To our knowledge, there are no existing DL-SCAs on any lightweight SISS ciphers such as Sprout [2], LILLE [4], and Fruit-80 [1]. This work on the DIZY cipher differs by introducing a DL-SCA targeted for a lightweight stream cipher, utilizing CNNs to analyze power traces from its hardware FPGA implementation, and focusing on a leakage model that targets the initialization phase without relying on multiple IVs.

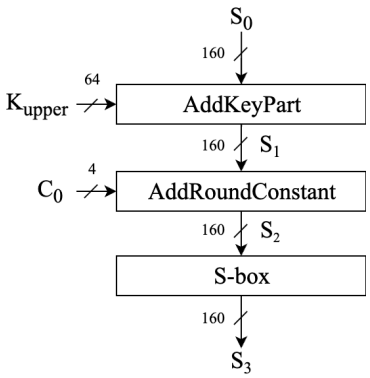


Fig. 1: Visual representation of DIZY's proposed leakage model

4 Methodology and Implementation

This section details the methodology and implementation for performing a DL-SCA on DIZY. Our implementation[4] is a Machine Learning (ML) pipeline that streamlines the DL-SCA process, which includes data acquisition, preprocessing, model training, model evaluation, and secret key recovery through side-channel analysis.

4.1 Leakage Model

S-boxes have been shown to be a primary source of side-channel leakage due to their inherent nonlinear operations [7, 18]. The DIZY stream cipher, which employs an SPN-based structure, utilizes 5-bit S-boxes in its design, making it susceptible to such attacks. However, extracting the key from these S-boxes is non-trivial. While the S-box outputs a 5-bit value, this value only embodies a 2-bit segment of the secret key. This contrasts with block ciphers such as AES, where the S-box output typically relates to a full byte of the key. We formulate the SCA as an ML classification problem, where a neural network is trained to predict the S-box outputs of the cipher's first round during key initialization directly from the captured power traces. Given these internal states, the secret key can be derived. Fig. 1 shows a high-level overview of the proposed leakage model where we predict the internal state S_3 output from the S-box.

We formulate the problem mathematically by first stating the notation used as:

> **Initial State:** Let S_0 be the 160-bit initial state, split into 32 groups of 5 bits each: $S_0 = (S_{0,0}, S_{0,1}, \ldots, S_{0,31})$. Here, $S_{0,0}$ is the most significant group (bits 155 to 159), and $S_{0,31}$ is the least significant group (bits 0 to 4).

[4] `https://gitlab.com/georgeattia/dizy-dl-sca`

Key: Let K be the 128-bit secret key. The upper 64 bits, called K_{upper}, are divided into 32 groups of 2 bits each: $K_0, K_1, \ldots, K_{31}$, where K_0 is the most significant 2 bits (bits 126 to 127 of K), and K_{31} is the least significant 2 bits of K_{upper} (bits 64 to 65).
Round Constant: Let C_0 be the 4-bit constant used in the first round, generated by an LFSR.

For each group $g = 0$ to 31, the first round applies the following steps:

1. **Key Addition:** The 2-bit key group K_g is XORed with the two most significant bits of $S_{0,g}$. This is written as:

$$S_{1,g} = S_{0,g} \oplus (K_g \ll 3)$$

2. **Constant Addition:** The 4-bit constant C_0 is XORed with the four least significant bits of $S_{1,g}$:

$$S_{2,g} = S_{1,g} \oplus (C_0 \wedge 0b01111)$$

3. **S-box:** A 5-bit to 5-bit S-box is applied:

$$S_{3,g} = \text{S-box}(S_{2,g})$$

An ML model predicts the S-box output $S_{3,g}$ for each group g, giving probabilities:

$$P_{g,v} = P(S_{3,g} = v) \quad \text{for} \quad v = 0, 1, \ldots, 31$$

where $P_{g,v}$ is the probability that $S_{3,g}$ equals v.

To obtain the 2-bit key part $K_g(v)$ for each group g from a predicted S-box output value v, we reverse the S-box by applying its inverse to v, reverse the constant addition by XORing the result with the known 4-bit round constant C_0, and compute the 2-bit key part as the XOR of the two most significant bits of this pre-constant-addition value with the two most significant bits of the initial state group $S_{0,g}$.

For each v, we get a possible $K_g(v)$ as described above. The probability that $K_g = k$ (where $k = 0, 1, 2, 3$) is:

$$P(K_g = k) = \sum_{v:K_g(v)=k} P_{g,v}$$

This process calculates the probability of each possible value of K_g based on the predictions of the model.

Given a set of power traces that will be analyzed to find their secret key, we can combine their predicted key probabilities. In order to avoid numerical underflow of probability aggregation, we use log probabilities. For a set of traces $t = 1, 2, \ldots, T$, the combined log probability for $K_g = k$ is:

$$\log P(K_g = k) = \sum_{t=1}^{T} \log P_t(K_g = k)$$

where $P_t(K_g = k)$ is the probability calculated from trace t.

To evaluate the attack's efficiency, we use a Key Rank metric derived from these accumulated log probabilities. For a specific 2-bit key group K_g and the true secret key value k^*, the rank is defined as the position of k^* within the sorted list of all possible key candidates when ordered by their likelihood. Specifically, we evaluate the rank across varying trace set sizes N. For a given size N, we randomly sample a subset of traces $\mathcal{T}_N$ and compute the accumulated log-likelihood for every candidate $k \in \{0, 1, 2, 3\}$:

$$\mathcal{L}_N(k) = \sum_{t \in \mathcal{T}_N} \log P_t(K_g = k)$$

The candidates are then sorted in descending order based on their scores $\mathcal{L}_N(k)$. The rank r is determined as the 1-based index of the correct key k^* in this sorted list; a rank of 1 indicates that the correct key is successfully identified as the most likely candidate. To ensure statistical robustness and mitigate variance arising from specific trace selections, this process is repeated $E = 5$ times for each size N using different random subsets. The final reported metric is the average rank:

$$\text{Average Rank}_N = \frac{1}{E} \sum_{i=1}^{E} r_i$$

This ranking metric serves as the primary benchmark for evaluating the convergence behavior of the trained models. By analyzing the Average Rank as a function of N, we can assess which model architecture is more data-efficient. Specifically, identifying which model allows the correct key bits to converge to a stable Rank 1 with the fewest number of power traces.

We note that the designed initial state of the cipher is 0. This leakage model assumes a controlled variable initial state and thus assumes a stronger adversary who can control the initial state either through fault injections or implementation errors. Additionally, this leakage model is designed to only recover the upper 64 bits of the key. However, it can be extended to recover the entire key. In this case, a neural network would be trained to predict the S-box outputs from the second round (after the lower part of the key is added). The probability vectors for both the first and the second rounds can then be used to reduce the guessing entropy of the lower part of the secret key.

4.2 Hardware Setup and Data Acquisition

The original designers of DIZY developed a parallel and a serial hardware implementation for each of the two variants of the cipher, focusing primarily on demonstrating its security proofs rather than optimizing for hardware efficiency [11,20]. Subsequent work by Schmid *et al.* [20] built upon these implementations by introducing optimizations aimed at improving energy efficiency. This work uses a round-based parallel implementation of DIZY-128 as synthesized by Schmid

et al. to perform the proposed SCA. This implementation performs one round of DIZY per clock cycle using parallel substitution and matrix multiplication operations that are performed at each round.

The power traces were acquired using the following hardware and software setup. The ChipWhisperer-Lite CW1173 board was used to measure power from a ChipWhisperer CW305 target board with an Xilinx Artix-7 FPGA (XC7A100T, FTG256 package) running DIZY. The bitstream was generated using Xilinx Vivado 2024.1. The target board operated at 5 MHz, while the capture board operated at 20 MHz, enabling four samples per target clock cycle. Note that both the target and capture chip clocks were synchronized to ensure consistent trace alignment across recordings.

Three datasets were captured to perform the SCA:

1. **Fixed-key Dataset:** 250,000 power traces. It uses a single, fixed secret key, a fixed IV, and a variable random initial state. This dataset is used as a baseline to assess the success of basic SCA, where both profiling and attack power traces use the same secret key.
2. **Variable-key Dataset:** 500,000 power traces. It uses 10 different secret keys, each associated with 50,000 traces. Each subset has a fixed IV. All power traces have variable and random initial states.
3. **Analysis Dataset with Variable-key:** 500,000 power traces. It uses 10 secret keys, each associated with 50,000 traces (different from the keys used for the profiling dataset). This dataset is used to provide more accurate measurements for performing side-channel analysis correctly.

The recorded power traces capture both the key and IV initialization phases. The recording stops before the stream generation phase begins. A collected power trace sample can be plotted for inspection, as shown in Fig. 2. The figure clearly shows 30 peaks of energy consumption. Each peak represents one full round function of the cipher's operation during one clock cycle. The first 15 peaks correspond to the key initialization, while the subsequent 15 peaks represent the IV initialization.

4.3 Data Preprocessing

During the data acquisition phase, each raw power trace X_i was recorded alongside its corresponding secret key K and random initial state S_0. We applied the leakage model defined in Section 4.1 to these known parameters to compute the S-box outputs for the first round of every trace. Specifically, the 160-bit intermediate state S_3 was calculated by analytically executing the first round's Key Addition, Constant Addition, and S-box operations. This state was then decomposed into 32 distinct 5-bit groups $(S_{3,0}, \ldots, S_{3,31})$. These computed values serve as the target labels $y_{true} \in \{0, \ldots, 31\}$ for the corresponding classification heads, thereby establishing a direct link between the physical power measurements and the algorithmic leakage defined in the proposed leakage model.

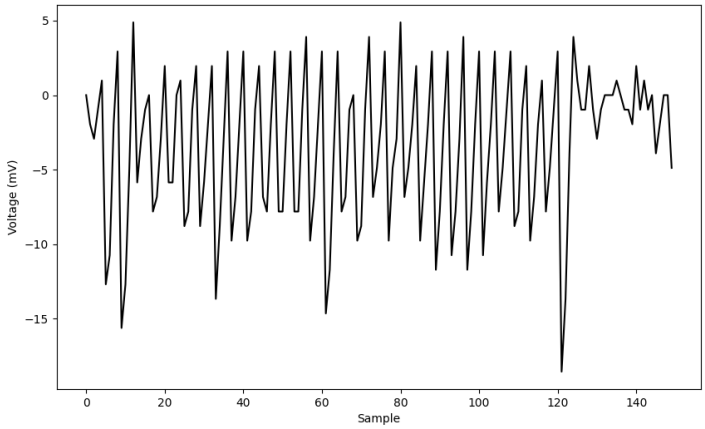


Fig. 2: Sample power trace collected from DIZY hardware

Following label calculation, the power traces collected were normalized to improve training stability, accelerate convergence, and improve model generalization. The normalization process standardizes the power traces using the training set statistics. For a training set $\mathbf{X}_{\text{train}} \in \mathbb{R}^{N \times T}$, where N is the number of traces and $T = 150$ is the number of sample points per trace, the normalized datasets are computed as follows:

The mean μ_j for each sample point $j = 1, 2, \ldots, T$ is computed as

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} X_{i,j}$$

The standard deviation σ_j for each sample point j is computed as

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (X_{i,j} - \mu_j)^2}$$

This yields the vectors $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_T]$ and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_T]$. Each trace $\mathbf{X}_i = [X_{i,1}, \ldots, X_{i,T}]$ across all sets (train, validation, test) is normalized as

$$\hat{X}_{i,j} = \frac{X_{i,j} - \mu_j}{\sigma_j} \quad \text{for} \quad j = 1, 2, \ldots, T$$

The result is $\hat{\mathbf{X}}_i$, with sample points standardized to a mean of 0 and a standard deviation of 1 based on the training set.

In addition, a uniform distribution of classification label values was empirically checked and verified. Since the leakage model uses uniformly random and

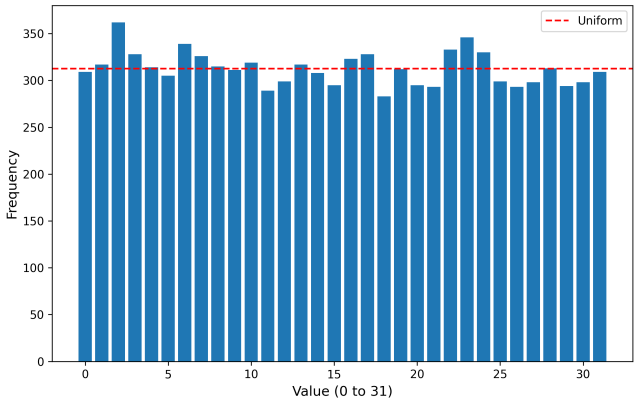


Fig. 3: Distribution of first round s-box output group 0 (sample size: 10000)

variable initial states, this also resulted in a uniform distribution of values for each S-box output. Fig. 3 shows an example distribution of one of the S-box outputs of the first round using a sample of 10,000 power traces. The distribution is near uniform and converges to perfect uniformity as the number of power trace sample increases. All other S-box instances exhibited the same value distribution.

Finally, we note that no manual signal processing techniques, such as temporal windowing or points-of-interest selection, were applied to physically isolate the specific clock cycle of the first round or the operations of individual S-boxes. The input to the neural network consists of the full power trace ($T = 150$ samples) capturing the simultaneous execution of all 32 parallel S-boxes across the entire initialization phase. Consequently, the task of isolating the leakage of a specific target S-box group from the aggregate power consumption and algorithmic noise is performed implicitly by the CNN during the supervised training process.

4.4 CNN Architectures

Two network architectures are compared: one using separate networks for each substitution box group and a multi-head model with a shared backbone. The separate networks model serves as a baseline for comparison with the multi-head model. It comprises 32 subnetworks, each predicts the output of one of the 32 S-box groups in the 160-bit internal state. Each subnetwork processes the power trace independently to classify the 5-bit S-box output into one of 32 possible labels. The model consists of one convolutional layer followed by two dense layers. The sum of parameters in each subnetwork is 12,948 parameters, and the total number of parameters for the entire model is 32 times the sum of the parameters in a single subnetwork $32 * 12,948 = 414,336$.

In contrast, the multi-head model uses a shared backbone for feature extraction, followed by multiple classification heads, to predict all 32 S-box outputs

Table 1: Layer specifications of the multi-head model

Layer Type	Parameters	Output Shape	Parameter Count
Conv1D	filters=10, kernel_size=3, strides=1, padding=no, act.=ReLU	(148, 10)	40
MaxPooling1D	pool_size=2, strides=2	(74, 10)	0
Flatten		(740)	0
Dense	units=64, act.=ReLU	(64)	47,424
Dense	units=64, act.=ReLU	(64)	4,160
Dense	units=64, act.=ReLU	(64)	4,160
Dense	units=64, act.=ReLU	(64)	4,160
Dropout	$p = 0.5$	(64)	0
Head-FC (x32)	units=48, act.=ReLU	(48)	3,120 (each)
Head (x32)	units=32, act.=Softmax	(32)	1,568 (each)

Table 2: Summary of trained models

ID	Architecture	Dataset Configuration	Traces (Train/Val/Test)	Keys (Train/Val/Test)
SepS	SeparateNetworks	Fixed-Key, Small	90K/10K/50K	Same Key
MultS	MultiHead	Fixed-key, Small	90K/10K/50K	Same Key
SepL	SeparateNetworks	Fixed-key, Large	180K/20K/50K	Same Key
MultL	MultiHead	Fixed-key, Large	180K/20K/50K	Same Key
SepVar	SeparateNetworks	Variable-key	400K/50K/50K	8/1/1
MultVar	MultiHead	Variable-key	400K/50K/50K	8/1/1

simultaneously. The model architecture is detailed in Table 1 and visualized in Fig. 4. It has 59,944 parameters in the backbone and 4,688 per head, totaling $59,944 + 32 * 4,688 = 209,960$ parameters. This is significantly lower than the 414,336 parameters in the separate networks model, highlighting its parameter efficiency.

4.5 Model Training

Six models were trained to evaluate both CNN architectures and the impact of dataset size. For the fixed-key scenario, two configurations were used: a "Small" dataset consisting of 150,000 traces (90,000 training, 10,000 validation, 50,000 testing) and a "Large" dataset consisting of 250,000 traces (180,000 training, 20,000 validation, 50,000 testing). The variable-key models used the variable-key dataset, with 400,000 training traces from 8 keys (50,000 each), 50,000 validation traces from 1 key, and 50,000 test traces from 1 key. This configuration tests the models' generalization across key variability while focusing on architectural differences. Table 2 summarizes these models.

All models shared a standardized training setup to ensure consistency. Cross-entropy loss was used for multi-class classification across the 32 S-box output

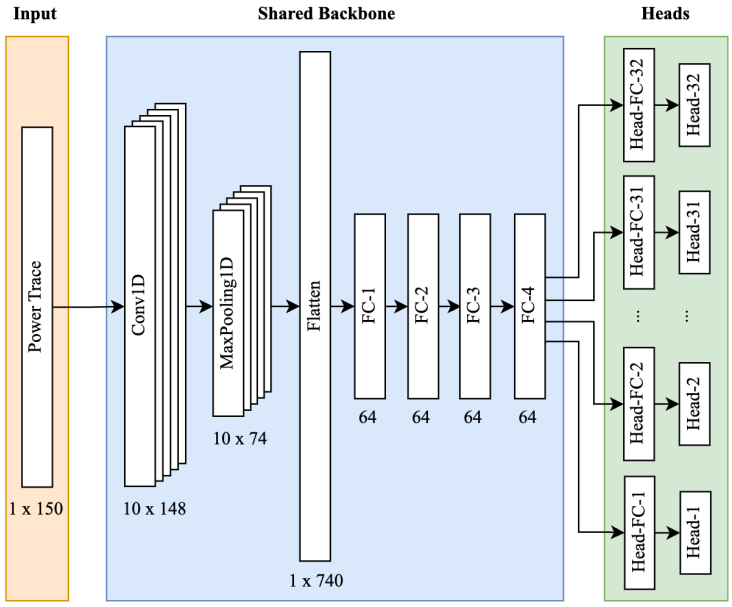


Fig. 4: Visual representation of the multi-head CNN model

groups, suitable for predicting probabilistic outputs. Adam optimizer was used with a learning rate of $1e-4$. Regularization techniques included dropout at a rate of 0.5 to suppress overfitting, and early stopping based on validation loss with a patience of 10 epochs. Additional hyperparameters were consistent: batch size of 128, up to 100 epochs to allow sufficient convergence.

For the multi-head models (MultFix and MultVar), a fine-tuning phase followed initial training to enhance per-head specialization. The shared backbone was frozen to preserve the learned features, while each of the 32 heads was unfrozen and trained individually. This used the Adam optimizer with a reduced learning rate of $1e-5$ and early stopping based on each head's validation loss. This step refines the predictions for individual S-box groups. The separate networks models (SepFix and SepVar) did not require fine-tuning, as their subnetworks were optimized independently from the beginning.

5 Evaluation and Discussion

This section evaluates the performance of the proposed DL-SCA on DIZY. Subsection 5.1 evaluates the multi-head models' performance differences due to changes in the training dataset size. In Subsection 5.2, a comparison between the variable-key models (SepVar and MultVar) is drawn. A direct comparison between the large dataset fixed-key models is omitted from this section for brevity. Section 5.3 presents the side-channel analysis results obtained from the analysis

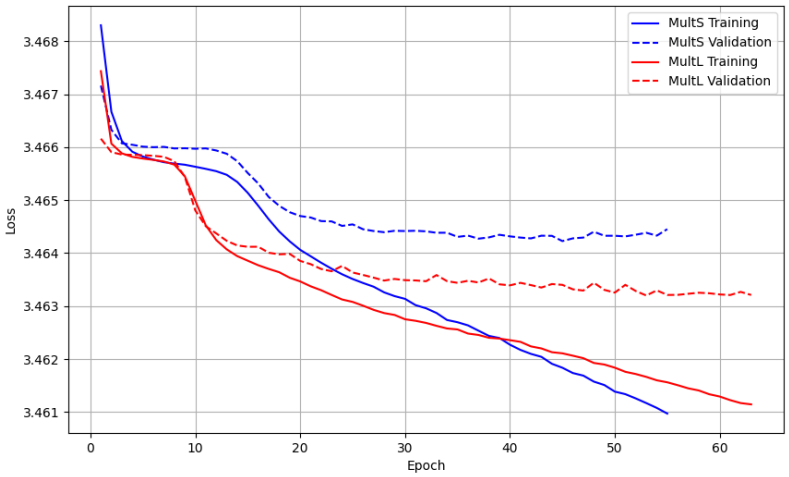


Fig. 5: Comparison of loss over epochs between the MultS and MultL models

power trace dataset. Finally, Subsection 5.4 discusses aggregated performance results across trained models.

5.1 Dataset Size Evaluation

The MultS model was trained on a subset of the available fixed-key training data to test the performance changes due to changes in dataset size. It was trained on 90,000 power traces, while the MultL model was trained on 180,000 power traces.

The MultS model converged with a validation loss of 3.4645, and the MultL model converged with a validation loss of 3.4632. Fig. 5 compares the training and validation losses over training epochs. It can be noted that the MultL model has a sharp decline in loss in an earlier epoch. This can be justified due to the change in the training dataset size. In other words, the MultL model converges faster since it has twice the dataset size when compared with the MultS model. It can also be noted that the MultL model has less pronounced overfitting. This matches the expectation since the increment of dataset size works as a form of regularization, and the model capacity matches the size of the available data.

In terms of post-training evaluation, the MultL model yielded an aggregate accuracy of 3.57% and an F1-score of 0.0293. The MultS model yielded an accuracy of 3.49% and F1-score of 0.0304 . The MultL model has an accuracy improvement of 2.29% and F1-score drop of 3.62% over the MultS model. Fig. 6 shows a comparison of the aggregate training and validation accuracy over epochs for the two models. It can also be noted that the same mentioned patterns of convergence and overfitting are also present in this accuracy figure.

Although there were relatively small differences in ML evaluation metrics between the MultS and MultL models, the convergence analysis shows more

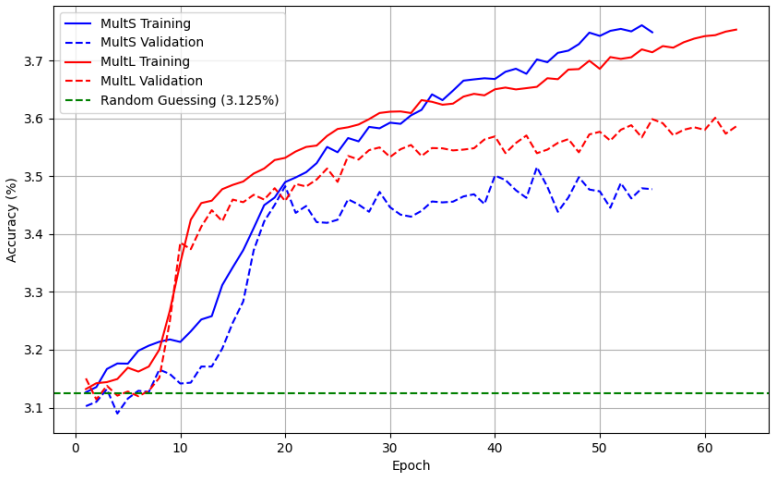
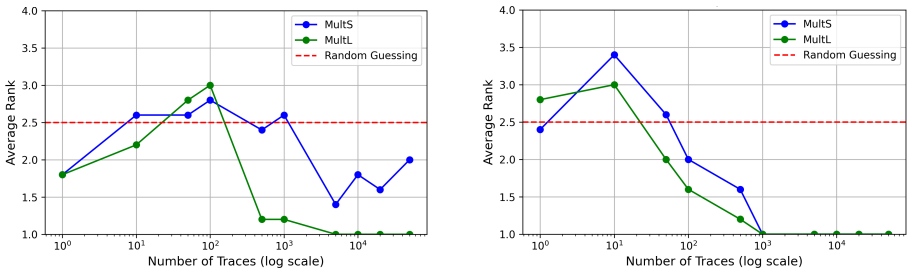


Fig. 6: Comparison of accuracy over epochs between the MultS and MultL models



(a) Group 11 (b) Group 27

Fig. 7: Comparison of convergence analysis between MultS and MultL models

substantial differences. The MultL model can generally recover more 2-bit key groups with fewer power traces. Using the test dataset, 11 out of 32 groups were correctly recovered using the MultL model, while the MultS model only recovered 6 out of 32 groups. The MultL model requires an average of 514 traces to recover the correct key bits, and the MultS model requires an average of 592 traces.

Fig. 7 shows the key convergence analysis of two example 2-bit key groups. In Fig. 7a, the MultS model failed to recover the key bits. In contrast, the MultL model had a stable ranking of 1 for the correct key bits. Both models could recover the key bits for group 27 as shown in Fig. 7b. However, the MultL model had a lower ranking of the correct key bits when calculating the average rank with different draw sizes of power traces. In other words, the MultL model was ahead of the MultS model in recovering key bits.

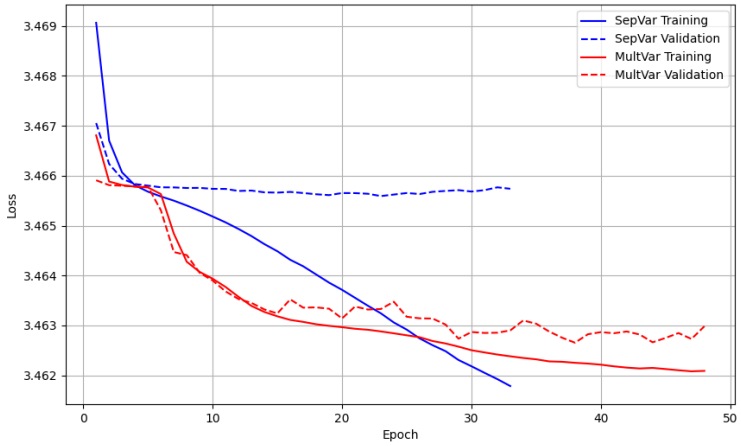


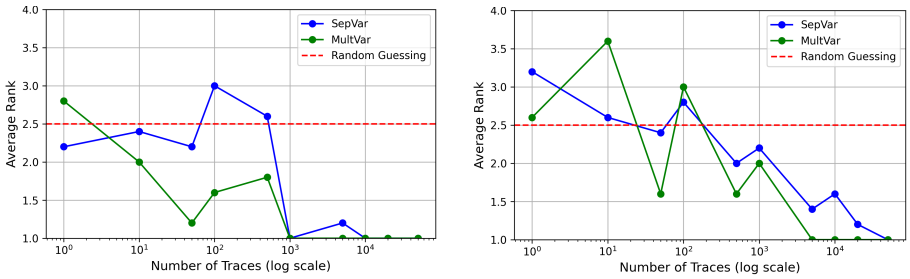
Fig. 8: Comparison of loss over epochs between the SepVar and MultVar models

5.2 Variable-key Models Evaluation

The SepVar and MultVar models were trained to test the performance differences between the proposed CNN architectures using a variable-key scenario. The MultVar model converged with a validation loss of 3.4630, and the SepVar model converged with a validation loss of 3.4657. Fig. 8 compares the training and validation losses of both models over training epochs. It can be noted that the MultVar model exhibits much better learning characteristics in comparison with the SepVar model. The MultVar converges to a lower loss, with minimal overfitting when compared with the SepVar model. In contrast, the SepVar model plateaued with a higher validation loss after the first 5 epochs of training, with a decreasing training loss with each additional epoch. We have also experimented with other regularization techniques and reduced the model capacity to reduce overfitting. However, that hindered learning and resulted in a higher loss in comparison with the current SepVar model. The comparatively better learning characteristics of the MultVar model can be associated with assigning 32 labels to each power trace, which inherently works as a form of data augmentation and emphasizes learning the correlation between power consumption and leakage of the cipher's secret key.

In post-training evaluation, the MultVar yielded an aggregate accuracy of 3.62% and an F1-score of 0.0276. The SepVar model yielded an accuracy of 3.29% and an F1-score of 0.0305. The MultVar model has an accuracy improvement of 10.03% and F1-score drop of 9.5% over the SepVar model.

In terms of key recovery analysis, the MultVar also showed improved performance in comparison with the SepVar, where 10 out of the 32 2-bit key groups converged towards the correct key bit. However, the SepVar model only recovered 6 out of the 32 groups. Similarly, the MultVar required an average of 775 power traces to correctly recover the key bits, while the SepVar required an average of 992 traces.



(a) Group 22 (b) Group 29

Fig. 9: Comparison of convergence analysis between SepVar and MultVar models

The convergence analysis of two-example 2-bit key groups is shown in Fig. 9. In Fig. 9a, the MultVar model was able to correctly rank the correct key bits at rank 1 starting from a set of 1000 traces. The model remained stable at its ranking for the subsequent larger trace draws (5000, 10000, 20000, and 500000 traces). The SepVar model, on the other hand, ranked the correct key bits with a lower rank using subsets of 5,000 traces. Additionally, the MultVar model has generally lower ranks for the correct key bits with smaller subsets of power traces in comparison with the SepVar model. Similarly, Fig. 9b shows similar patterns as shown in Fig. 9a. These results generally suggest that the multi-head architecture is more capable of recovering the key bits when compared to the mainstream Separate Networks approach.

5.3 Analysis Dataset

During the evaluation of trained models, it has been noted that their performance varies depending on the test dataset used. The number of recovered 2-bit key groups was different and with different estimates for the number of power traces required for the model to rank the correct key bits with a rank of 1.

Based on these observations and to find a more accurate estimate of the models' performance, we captured the analysis dataset (as mentioned in Section 4.2). The analysis dataset has a total of 500,000 power traces with 10 different secret keys (50,000 traces per key).

Both the SepVar and MultVar models were tested using each key subset of the analysis dataset. Table 3 shows the convergence analysis results for each key subset of the data, along with their aggregate mean and standard deviation. While the MultVar model recovered a mean of 6.9 2-bit key groups with a standard deviation of 3.2128, the SepVar model only recovered a mean of 2.8 2-bit key groups with a lower standard deviation of 1.9889.

5.4 Discussion

This section gives a high-level overview of the specifications and performance results of the six trained models. It also discusses general aspects of the im-

Table 3: Key convergence analysis results for the analysis dataset

Dataset ID	SepVar - Conv. Grps.	MultVar - Conv. Grps.
0	1/32	3/32
1	1/32	2/32
2	4/32	10/32
3	5/32	11/32
4	2/32	8/32
5	2/32	4/32
6	1/32	5/32
7	5/32	9/32
8	1/32	7/32
9	6/32	10/32
μ	**2.8/32**	**6.9/32**
σ	**1.9889**	**3.2128**

Table 4: Overview of the specifications and performance of the trained models

Model ID	Converging Grps.	# of Power Traces	# of Parameters	Training Time (s)	Accuracy
SepS	2/32	3050	414,336	**1147**	0.0311
MultS	6/32	592	**209,960**	3617	0.0349
SepL	0/32	N/A	414,336	2080	0.0318
MultL	**11/32**	**514**	**209,960**	7492	0.0357
SepVar	6/32	992	414,336	7833	0.0329
MultVar	10/32	775	**209,960**	13730	**0.0362**

plemented DL-SCA on the DIZY cipher. Table 4 specifies the training time, number of parameters, number of converging 2-bit key groups, average number of required power traces for convergence, and model accuracy for each of the trained models. The results must be interpreted in the context of the cipher's high parallelism. With 32 S-boxes switching simultaneously, the classification task involves distinguishing the leakage of one 5-bit group amidst the noise of 31 others. Despite this low SNR environment, the multi-head model demonstrates successful recovery.

Each S-box of the DIZY-128 cipher outputs one of 32 possible values. Since the distribution of these values has been shown to be uniform (see Section 4.3), a random classifier that predicts the output of any S-box would have an accuracy of 3.125%. For all trained models, the overall accuracy is only slightly higher than that of a random classifier. This has been shown to be adequate, since the predicted probabilities of several power traces are aggregated. Given sufficient leakage patterns in the used traces, the actual key bits will have the highest likelihood as per the model, and thus will be ranked at position 1.

As noted before, we emphasize that the multi-head models offer improved performance when compared with their Separate Networks counterparts. Ad-

ditionally, the training and inference times for the multi-head architecture are shorter than those that use the Separate Networks architecture (assuming no fine-tuning phase for the multi-head models).

To our knowledge, multi-task deep learning networks are extensively used in many ML applications (for example, Natural Language Processing (NLP) applications). However, they have not been used in the context of DL-SCAs. The use case for these types of architectures seems logical to us, since the overall goal of an SCA is to recover the entire secret key rather than recovering a specific part of it. That said, benchmarks and research competitions that focus on recovering a specific part of the key have held back the potential for exploring multi-task deep learning architectures for SCAs.

6 Conclusion

This work performed a DL-SCA on the lightweight DIZY stream cipher through a leakage model that targets the cipher's S-boxes used in the first round of the initialization phase. This leakage model uses a variable initial state of the cipher and thus assumes a stronger adversary who can control the initial state. Additionally, the proposed leakage model targets only the upper part of the key (the upper 64 bits). However, we showed that this model can be extended to target the entire secret key.

Two CNN models were developed to test the effectiveness of CNNs in performing side-channel analysis. One model, the separate networks model, uses 32 independent neural networks, each of which predicts the output of its corresponding S-box. The other, multi-head model, uses a single neural network with 32 heads and a shared backbone to predict the 32 S-box outputs simultaneously.

Results have shown that the multi-head architecture offers better performance in terms of accuracy and faster convergence towards the correct key bits. Our evaluation of dataset sizes further indicates that increasing the training set size from 90,000 to 180,000 traces nearly doubles the number of recovered key groups (from 6 to 11), highlighting the importance of data volume in profiling attacks. They have also shown that the multi-head architecture offers shorter training and inference times when compared with the separate networks architecture.

Both fixed- and variable-key datasets were captured to train and evaluate the proposed models. Both settings have similar results in terms of the models' ability to recover the key bits. The upper key part was partially recovered, with the multi-head model successfully retrieving up to 11 out of 32 two-bit key groups across different datasets, achieving a mean recovery of 6.9 groups in the variable-key analysis. This demonstrates the effectiveness of the proposed DL-SCA in exploiting side-channel leakages in the DIZY cipher. Finally, a critical direction for future work involves the investigation of protected hardware implementations of DIZY. Specifically, applying this multi-head deep learning attack against designs incorporating countermeasures, such as Boolean masking

or shuffling, will provide deeper insights into the resilience of SISS ciphers in hardened, energy-constrained environments.

Acknowledgments. This study was funded in part by DFG Project No. 439892735 (SPP 2253).

References

1. Amin Ghafari, V., Hu, H.: Fruit-80: A Secure Ultra-Lightweight Stream Cipher for Constrained Environments. Entropy **20** (03 2018). https://doi.org/10.3390/e20030180
2. Armknecht, F., Mikhalev, V.: On Lightweight Stream Ciphers with Shorter Internal States. In: Leander, G. (ed.) Fast Software Encryption. pp. 451–470. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_22
3. Babbage, S., Dodd, M.: The MICKEY Stream Ciphers, p. 191–209. Springer-Verlag, Berlin, Heidelberg (2008), `https://doi.org/10.1007/978-3-540-68351-3_15`
4. Banik, S., Isobe, T., Morii, M.: On Design of Robust Lightweight Stream Cipher with Short Internal State. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E101.A**, 99–109 (01 2018). https://doi.org/10.1587/transfun.E101.A.99
5. Buchanan, W.J., Li, S., Asif, R.: Lightweight Cryptography Methods. Journal of Cyber Security Technology **1**(3-4), 187–201 (2017). https://doi.org/10.1080/23742917.2017.1384917
6. Cannière, C., Preneel, B.: Trivium, p. 244–266. Springer-Verlag, Berlin, Heidelberg (2008), `https://doi.org/10.1007/978-3-540-68351-3_18`
7. Carlet, C.: On Highly Nonlinear S-boxes and Their Inability to Thwart DPA Attacks. In: Proceedings of the 6th International Conference on Cryptology in India. p. 49–62. INDOCRYPT'05, Springer-Verlag, Berlin, Heidelberg (2005). https://doi.org/10.1007/11596219_5
8. Chakraborty, A., Mazumdar, B., Mukhopadhyay, D.: A Combined Power and Fault Analysis Attack on Protected Grain Family of Stream Ciphers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **PP**, 1–1 (02 2017). https://doi.org/10.1109/TCAD.2017.2666601
9. Chakraborty, A., Mukhopadhyay, D.: A Practical Template Attack on MICKEY-128 2.0 Using PSO Generated IVs and LS-SVM. In: 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID). pp. 529–534 (2016). https://doi.org/10.1109/VLSID.2016.66
10. Fischer, W., Gammel, B.M., Kniffler, O., Velten, J.: Differential Power Analysis of Stream Ciphers. In: Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology. p. 257–270. CT-RSA'07, Springer-Verlag, Berlin, Heidelberg (2007). https://doi.org/10.1007/11967668_17
11. Gül, Ç., Kara, O.: A New Construction Method for Keystream Generators. IEEE Transactions on Information Forensics and Security **18**, 3735–3744 (2023). https://doi.org/10.1109/TIFS.2023.3287412
12. Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. Int. J. Wire. Mob. Comput. **2**(1), 86–93 (May 2007). https://doi.org/10.1504/IJWMC.2007.013798

13. Kazmi, A.R., Afzal, M., Amjad, M.F., Abbas, H., Yang, X.: Algebraic Side Channel Attack on Trivium and Grain Ciphers. IEEE Access **5**, 23958–23968 (2017). https://doi.org/10.1109/ACCESS.2017.2766234
14. Kumar, S., Dasu, V., Baksi, A., Sarkar, S., Jap, D., Jakub, B., Bhasin, S.: Side Channel Attack On Stream Ciphers: A Three-Step Approach To State/Key Recovery. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**, 166–191 (02 2022). https://doi.org/10.46586/tches.v2022.i2.166-191
15. Liu, J., Gu, D., Guo, Z.: Correlation Power Analysis Against Stream Cipher MICKEY v2. In: Proceedings of the 2010 International Conference on Computational Intelligence and Security. p. 320–324. IEEE Computer Society, USA (2010). https://doi.org/10.1109/CIS.2010.75
16. National Institute of Standards and Technology: FIPS 197: Advanced Encryption Standard (AES). Federal Information Processing Standards Publication (2001), `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf`
17. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: SoK: Deep Learning-based Physical Side-channel Analysis. ACM Comput. Surv. **55**(11) (Feb 2023). https://doi.org/10.1145/3569577
18. Prouff, E.: DPA Attacks and S-boxes. In: Proceedings of the 12th International Conference on Fast Software Encryption. p. 424–441. FSE'05, Springer-Verlag, Berlin, Heidelberg (2005). https://doi.org/10.1007/11502760_29
19. Qu, B., Gu, D., Guo, Z., Liu, J.: Differential Power Analysis of Stream Ciphers with LFSRs. Computers & Mathematics with Applications **65**, 1291–1299 (05 2013). https://doi.org/10.1016/j.camwa.2012.02.024
20. Schmid, M., Arul, T., Kavun, E.B., Regazzoni, F., Kara, O.: Robust and Energy-efficient Hardware Architectures for DIZY Stream Cipher. In: 2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). pp. 461–465 (2024). https://doi.org/10.1109/APCCAS62602.2024.10808577
21. Schmid, M., Kavun, E.B.: Differential Power Analysis on Low-Energy Keystream Generating Hardware: Full-State Recovery in Dizy Implementation. In: 2025 IEEE 36th International Conference on Application-specific Systems, Architectures and Processors (ASAP). pp. 147–154 (2025). https://doi.org/10.1109/ASAP65064.2025.00032
22. Zhang, L., Xing, X., Fan, J., Wang, Z., Wang, S.: Multi-label Deep Learning based Side Channel Attack. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **PP**, 1–1 (10 2020). https://doi.org/10.1109/TCAD.2020.3033495