CLAASPing ARADI: Automated Analysis of the ARADI Block Cipher

Emanuele Bellini[0000−0002−2349−0247], Mattia Formenti[0009−0001−0069−6146], David Gérault[0000−0001−8583−0668], Juan Grados[0000−0002−3863−3714], Anna Hambitzer[0000−0002−5357−832X], Yun Ju Huang[0000−0002−0820−1005], Paul Huynh[0000−0002−6965−3427], Mohamed Rachidi[0009−0008−5279−4902], Raghvendra Rohit[0000−0002−5272−1016], and Sharwan K. Tiwari[0000−0002−9487−0669]

Technology Innovation Institute, Cryptography Research Center, Abu Dhabi, UAE
{name.lastname}@tii.ae
https://www.tii.ae/cryptography

Abstract. In early August 2024, three NSA researchers – Patricia Greene, Mark Motley, and Bryan Weeks – published the technical specifications for a new low-latency block cipher, ARADI, along with its corresponding authenticated encryption mode, LLAMA, which is specifically designed for memory encryption applications. Their manuscript offered minimal security analysis of the design, only briefly discussing the differential, linear and algebraic properties of cipher's underlying components. In this work, we present a set of distinguishers for the round reduced ARADI block cipher, discovered using the automated cryptanalysis tool CLAASP. More precisely, using CLAASP, we evaluate the resistance of ARADI against avalanche, statistical and continuous diffusion tests, differential and linear distinguishers, impossible differentials, algebraic attacks, and neural distinguishers. Accordingly, we give distinguishers that reach up to 9 out of 16 rounds of ARADI. We hope these preliminary findings will encourage further in-depth cryptanalysis of the cipher to enhance confidence in its security.

Keywords: ARADI, CLAASP, low-latency ciphers, differential cryptanalysis, linear cryptanalysis, statistical tests, neural distinguishers

Table of Contents

CLAASPing ARADI: Automated Analysis of the ARADI Block Cipher 1
Emanuele Bellini, Mattia Formenti, David Gérault, Juan Grados, Anna Hambitzer, Yun Ju Huang, Paul Huynh, Mohamed Rachidi, Raghvendra Rohit, and Sharwan K. Tiwari

1 Introduction 3
1.1 Our Contributions 3
1.2 Organization of the Paper 4
2 Preliminaries 4
2.1 Specification of ARADI 5
2.2 CLAASP Overview 5
2.3 Implementation of ARADI in CLAASP 6
3 Avalanche Tests 7
3.1 Description 7
3.2 Findings 8
4 Statistical Tests 10
4.1 Description 10
4.2 Findings 11
5 Continuous Diffusion Tests 12
5.1 Description 12
5.2 Findings 12
6 Differential and Linear Analysis 14
6.1 Description 14
6.2 Findings 15
7 Impossible Differential Cryptanalysis 17
7.1 Description 17
7.2 Findings 18
8 Algebraic Analysis 19
8.1 Description 19
8.2 Findings 20
9 Neural Distinguishers 21
9.1 Description 21
9.2 Findings 21
10 Conclusions 23
A Avalanche Entropy Details 26

1 Introduction

In recent years, the rapid evolution of computing architectures has driven the need for new cryptographic designs that can meet the unique demands of these emerging environments. Many of these architectures, particularly those found in constrained devices such as IoT sensors, embedded systems, and real-time applications, require cryptographic algorithms that are not only secure but also efficient in terms of speed, power consumption, and resource utilization. This has led to a surge in the development of cryptographic algorithms tailored for such contexts.

One of the prominent industrial use-case is the design of ciphers for pointer and memory encryption, where *low-latency* meaning low *critical path of the circuit* is a major requirement. In 2021, researchers from the Intel Labs proposed a new memory safety mechanism, called Cryptographic Capability Computing (C3) [36], highlighting again the need of low-latency ciphers. Though the idea of low-latency ciphers is not something new, the trade-offs between security and latency has led to the proposal of several such ciphers till date. Notable examples include PRINCE [17], MANTIS [8], QARMA [2], PRINCEv2 [18], SPEEDY [35], ORTHROS [4], QARMAv2 [3], SCARF [20], BipBip [9], Sonic and SuperSonic [10].

In early August 2024, three NSA researchers—Patricia Greene, Mark Motley, and Bryan Weeks, introduced a new low-latency block cipher named ARADI [30], along with its corresponding authenticated encryption mode, LLAMA. This new cryptographic primitive is specifically designed to meet the stringent requirements of memory encryption applications, where both security and performance are critical. The ARADI cipher, due to its low-latency characteristics, holds promise for environments where rapid data processing and encryption are essential, such as in real-time systems, high-performance computing, and various embedded systems.

The ARADI cipher was detailed in a manuscript that primarily focused on its technical specifications and design principles. However, the manuscript provided only a minimal security analysis, briefly touching on the differential, linear, and algebraic properties of the cipher's underlying components. The limited scope of this initial security evaluation has left open questions regarding the robustness of ARADI against a broader spectrum of cryptanalytic attacks. This gap in the analysis necessitates further investigation to assess and establish the security guarantees of ARADI, especially given its potential application in critical systems.

To facilitate such a comprehensive security evaluation, automated cryptanalysis tools can be of immense value. One such tool is the CLAASP [11] library, a cutting-edge framework that enables the systematic exploration of cryptographic algorithms under various attack models. CLAASP provides a platform for conducting a wide range of cryptanalytic techniques, including tests for avalanche effects, statistical and continuous diffusion, differential and linear distinguishers, impossible differentials, algebraic properties, and neural distinguishers. These techniques are essential both as quick preliminary security assessment of a symmetric cipher, and as building blocks of key recovery attacks. Automated tools also offer a framework to benchmark different techniques and compare the effectiveness of these techniques when applied to a wide range of ciphers [15].

In this paper, we leverage the capabilities of CLAASP to perform an in-depth analysis of the ARADI block cipher. Our focus is on discovering and evaluating distinguishers for round-reduced versions of ARADI. Specifically, we assess the resistance of ARADI against a variety of cryptanalytic tests using CLAASP, aiming to provide a clearer picture of the cipher's security. Through our analysis, we identify distinguishers that extend up to 9 out of the 16 rounds of ARADI, raising important considerations for the cipher's overall security. These findings serve as a preliminary step in the ongoing evaluation of ARADI and underscore the need for further cryptanalysis to bolster confidence in the cipher's security.

Our work not only contributes to the understanding of ARADI's security properties but also highlights the utility of the CLAASP library in modern cryptanalysis. By applying CLAASP's comprehensive suite of cryptanalytic tools, we aim to encourage further research into the security of ARADI, ultimately contributing to the development of more secure cryptographic standards for memory encryption and beyond.

1.1 Our Contributions

In this paper, we give the first third-party cryptanalysis results on the ARADI block cipher. We implement two variants of ARADI, referred to as the *bitsliced* and *S-box* variants, in CLAASP. We

then use the CLAASP generic cryptanalysis tools to provide a comprehensive preliminary security analysis of ARADI. Our contributions are summarized as follows.

Avalanche tests We study the propagation of 1-bit input differences and show that it takes 5 rounds for such input differences to diffuse to the entire state. Moreover, we also show that it takes 5 rounds for ARADI to achieve the full bit avalanche effect in both plaintext and key variables.

Statistical tests We apply the NIST statistical tests on ARADI to analyze its randomness behavior. We show that ARADI passes these tests after 5 rounds.

Continuous diffusion tests We investigate the behavior of ARADI using the Continuous Avalanche Factor (CAF) (see [21] and subsection 5.1), and show that CAF value reaches 0.725 at round 7 under specific parameters. After 8 rounds, this value becomes 0.955 indicating that ARADI has strong diffusion for 8 of its 16 rounds under the same conditions.

Differential and linear trails We report optimal differential for 10 rounds and optimal linear trails for 8 rounds in the single-key setting. We also find 8-round differential with probability 2^{-109} and 9-round differential with probability $2^{-125.5}$. Moreover, we report a 9-round linear trail with squared correlation $2^{-124.82}$.

Impossible differentials We find a class of 230,400 impossible differentials for 8 rounds. These are derived from 7-round impossible differentials having Hamming weight of both input and output difference as 1.

Algebraic analysis We perform the algebraic analysis using CLAASP's algebraic tests module, which models the cipher symbolically and tries to solve the Boolean polynomial system using Gröbner bases. We also use the division property to find integral distinguishers for ARADI up to 7 rounds. By fixing certain variables, we observe that the algebraic degree grows more slowly, reaching 60 after 5 rounds, which leads to an integral distinguisher with a data complexity of 2^{61}.

Neural distinguishers We report 5-round neural distinguishers in single-key setting with an accuracy of 0.5954. In the related-key setting, we obtain 6-round neural distinguishers with an accuracy of 0.5631.

1.2 Organization of the Paper

The rest of the paper is organized as follows. In section 2, we give the specification of the ARADI block cipher, a brief introduction to CLAASP, and discuss the block cipher's implementation in this framework. In section 3, section 4, and section 5 we present the analysis of ARADI considering avalanche tests, statistical tests and continuous diffusion tests, respectively. In section 6, we discuss the differential and linear trails of ARADI, and report the best found trails. In section 7 we provide the impossible differential trails while in section 8 we investigate the algebraic degree growth of ARADI. We then present the neural distinguishers in section 9.

In each of these sections, we first give a high-level overview of the test or distinguisher, discuss our findings and then provide the corresponding CLAASP scripts to reproduce the results. We conclude the paper in section 10 with future directions.

2 Preliminaries

In this section, we introduce the ARADI block cipher, provide some background of the CLAASP tool, and describe how the block cipher has been implemented in this tool.

2.1 Specification of ARADI

ARADI is a low-latency block cipher proposed by Greene et al. [30] from the US National Security Agency. The block size and key size are 128 and 256 bits, respectively. The ARADI round function is based on the substitution permutation network design paradigm and consists of three operations, namely the S-box layer π, the i-th linear map Λ_i, and the i-th round key addition τ_{k_i}. The cipher consists of 16 rounds and is given by

$$\tau_{k_{16}} \circ \bigcirc_{i=0}^{15} (\Lambda_{i \bmod 4} \circ \pi \circ \tau_{k_i}), \tag{1}$$

where the composition is read from right to left.

We now give a brief overview of the individual operations. We first note that each operation works on a 128-bit state arranged into 4 32-bit words w, x, y, z and the complete state is read as $w\|x\|y\|z$ or (w, x, y, z).

The S-box layer π. The input state (w, x, y, z) is transformed by π as follows:

$$\begin{aligned} x &\leftarrow x \oplus (w \,\&\, y), \\ z &\leftarrow z \oplus (x \,\&\, y), \\ y &\leftarrow y \oplus (w \,\&\, z), \\ w &\leftarrow w \oplus (x \,\&\, z). \end{aligned} \tag{2}$$

The linear layer Λ_i. At round i, the input state (w, x, y, z) is transformed by Λ_i as follows:

$$\Lambda_i(w, x, y, z) \rightarrow (L_i(w), L_i(x), L_i(y), L_i(z)), \tag{3}$$

where L_i is an involutory linear map on 32-bit words.

Let the 32-bit input to L_i is composed of two 16-bit words u and l. Then L_i is given by

$$(u, l) \rightarrow (u \oplus S_{16}^{a_i}(u) \oplus S_{16}^{c_i}(l),\ l \oplus S_{16}^{a_i}(l) \oplus S_{16}^{b_i}(u)), \tag{4}$$

where the operation $S_{16}^{m}(\cdot)$ denotes the left circular shift of a 16-bit word by m positions. The shift offsets of the linear layer are given in Table 1.

Table 1. Shift offsets of the ARADI linear layer.

$i \bmod 4$	a_i	b_i	c_i
0	11	8	14
1	10	9	11
2	9	4	14
3	8	9	7

The key addition layer. This operation XORs a 128-bit round key to the state. The round keys are generated by the dedicated key scheduling algorithm ([30][Section 3.2]). We omit the details of key schedule as the analyses in this paper are independent of it.

2.2 CLAASP Overview

CLAASP, introduced in [11] stands for *Cryptographic Library for the Automated Analysis of Symmetric Primitives*. This library is an open-source tool designed to simplify and automate the analysis and design of symmetric ciphers. Built on top of Sagemath and Python, CLAASP is modular, extendable, and user-friendly, offering a comprehensive environment for cryptographers. It supports

a wide range of symmetric ciphers (more than 70 at the time of writing), including block ciphers, cryptographic permutations, and hash functions, and accommodates various design types like Feistel, Substitution-Permutation Networks (SPN), Addition-Rotation-Xor (ARX) constructions, and Linear and Nonlinear Feedback Shift Register-based ciphers (LFSR and NFSR).

The core of CLAASP revolves around representing ciphers as directed acyclic graphs, where components like S-boxes, linear layers, and constants are connected via input/output edges. From this representation, CLAASP can automatically generate code, perform statistical and avalanche tests, and analyze algebraic properties and neural distinguishers. It can also generate SAT, SMT, CP, and MILP models for use with various solvers, making it a highly automated and efficient tool for cipher analysis. Despite its broad functionality and automation, CLAASP remains competitive in terms of efficiency with more specialized tools.

Two tools based on a similar concept, but with a narrower scope, are TAGADA [37] and CASCADA [42]. TAGADA was initially designed to tackle Minizinc [41] models for the search for differential properties on word-based SPN ciphers, such as the AES, using the two-step strategy. On the other hand, CASCADA was designed to exploit SMT models for ARX ciphers, implemented through the theory of bit-vectors [5]. A more detailed comparison on these two general frameworks can be found in the CLAASP original paper [11], while the latest updates can be followed from their corresponding repositories: https://gitlab.com/tagada-framework/tagada for TAGADA and https://github.com/ranea/CASCADA for CASCADA.

At the time of writing and running the tests CLAASP v2.6.0 has been released. The source code is available at the following GitHub repository:

https://github.com/Crypto-TII/claasp.

The library can be installed directly on the user's machine or via its Docker image. CLAASP is also available to the public, with no installation effort, through the CLAASP-WEB platform at:

https://claasp.tii.ae/

While all the scripts in this document could be run in the free version of CLAASP-WEB as they are, most of them will take time to execute, due to the limited resources of the application. For better performances, we run most of the scripts on dedicated machines.

2.3 Implementation of ARADI in CLAASP

To exploit the several cryptanalytic tests already implemented in CLAASP, the user must provide a CLAASP implementation of the cipher under scrutiny. The way a cipher is represented is not unique, and as a consequence, different implementations of the cipher are possible, possibly serving different purposes depending on the tests.

We provided two implementation variants of ARADI in CLAASP. The first is the **bitsliced** variant, where each operation is carried out on 32-bit words. The second is **S-box** based, where we use the 4-bit S-box of ARADI. Both implementations are publicly available at:

- **Bit-sliced**: https://github.com/Crypto-TII/claasp/blob/main/claasp/ciphers/block_ciphers/aradi_block_cipher.py
- **S-box-based**: https://github.com/Crypto-TII/claasp/blob/main/claasp/ciphers/block_ciphers/aradi_block_cipher_sbox.py

The reason to have two different implementation is that different representations are more suitable to certain tests. For example, the bitsliced version yields a faster evaluation function, useful for the avalanche, statistical, continuous diffusion, and neural tests, which require to evaluate many instantiations of the cipher in parallel. On the other hand, the bitsliced version produces less accurate probabilities when modeling linear and differential propagations through the S-box layer, due to the input-output dependencies of the word-based components. These dependencies can somehow be ignored in the S-box version, since the multiple word-based components constituting the S-box are grouped together and looked at as a single block.

Benchmarks The benchmarks in Table 2 were obtained with the script Listing 1.1.

```
import numpy as np
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher
from timeit import default_timer as timer
import pickle

aradisb=AradiBlockCipherSBox()
aradi=AradiBlockCipher()

for run in range(10):
    times = []
    for pow in range(4, 8):
        X = np.random.randint(256, size=(16, 10**pow), dtype=np.uint8)
        K = np.random.randint(256, size=(32, 10**pow), dtype=np.uint8)
        t0 = timer()
        C0 = aradi.evaluate_vectorized([X, K])
        t1 = timer()-t0

        t0 = timer()
        C1 = aradisb.evaluate_vectorized([X, K])
        t2 = timer()-t0

        times.append((pow, t1, t2))
        print(times[-1])
    with open(f'bench_aradi_{run}', 'wb') as fp:
        pickle.dump(times, fp)
```

Listing 1.1. Bitsliced and S-box based implementations benchmark script.

Table 2. Benchmark of the ARADI Bitsliced and S-box based implementations for different sample sizes, with mean and standard deviation over 10 runs. The measurements were taken on a machine with 2x Intel(R) Xeon(R) Platinum 8490H Processors (total of 240 CPUs), 8x NVIDIA H100 80GB HBM3 GPUs, and 4TB of RAM.

# **Samples**	**Bitsliced**[seconds]	**S-box based**[seconds]
2^4	1.7 ± 0.2	2.4 ± 0.1
2^5	22.2 ± 1.3	28.6 ± 2.1
2^6	192.0 ± 3.4	239.3 ± 2.8
2^7	2220.5 ± 38.8	3074.7 ± 9.3

3 Avalanche Tests

In this section we provide a description and our findings about the avalanche properties of ARADI.

3.1 Description

A test to measure the avalanche properties of a symmetric iterated cipher is presented in [23] (see also [13] for its higher-order version). This test evaluates the cipher with respect to three different metrics that are generalizations of the *full diffusion*, the *avalanche*, and the *strict avalanche* criteria. The goal of the test is to compare how these criteria evolves with respect to the computational cost of the round function. Note that the common behaviour of an iterated cipher is to not meet the criterion for the first few rounds and then to meet it for all the remaining ones.

The avalanche probability vector The test is performed by computing the so called Avalanche Probability Vector (APV) $P_{\Delta F}$ of a cryptographic primitive F for an input difference Δ. The i-th component of the APV is the probability that bit i of the output of F flips due to the input difference Δ, or, equivalently, the probability that bit i of $F(x)+F(x+\Delta)$ equals 1. After M samples, the expected standard deviation of the elements of $P_{\Delta F}$ is $1/\sqrt{M}$. So for high precision, M must be chosen large enough. In [23], experiments $M = 250,000$ was used, but, as also mentioned in [13], much lower values already provide a good estimate of the probability. In this work, we observe the behaviour of the tests for $M = 1000$. Also, in this work, we do not compute the 3 metrics mentioned in [23], but we limit ourselves to the computation of the APV and its variants. The Avalanche Entropy Vector, for example, contains the corresponding bit-entropy instead of the probability. The pseudocode to compute the APV is provided in Algorithm 1.

Algorithm 1 Avalanche probability vector of order d

Require: a transformation F over $GF(2)^b$, a vector space $\mathcal{V}$ of length b and dimension d generated by a basis of single-bit vectors, and number of samples M.
Ensure: p, the avalanche probability vector of order d.
1: Initialize a b-bit vector p of probabilities p_i to all zeroes.
2: **for** M randomly generated states x **do**
3: Compute $B = \sum_{v \in \mathcal{V}} F(x+v)$
4: **for** all state bit positions i **do**
5: $p_i = p_i + B_i/M$
6: **end for**
7: **end for**

3.2 Findings

The Listing 1.2 generates the plot in Figure 1, which represents the average (over all 1-bit input differences) avalanche entropy vector of all 16 rounds. The greener the color the closer to 1 is the bit entropy. The redder, the closer to 0. The values of this graph are zoomed in section A. There it is possible to see that round 4 still contains several 0.998 values, indicating that "full diffusion" has not yet been reached. After round 5, included, all values have reached maximum entropy. This means that, on average, 25% of the 16 rounds are needed to propagate a 1-bit difference to the entire internal state of the cipher.

Compared to other ciphers, e.g. the finalists of the NIST LW standardization process, range from the 10% to the 35% (with the exception of TinyJambu, which had a very high percentage, between 42% and 52%, depending on the key size, which was eventually broken. For more details, see [14, Fig. 7]). This makes ARADI an average conservative cipher from the avalanche effect perspective.

CLAASP scripts In Listing 1.3 we report the code to generate Figure 1, which visualizes the average avalanche weight per bit.

```
from claasp.cipher_modules.avalanche_tests import AvalancheTests
from claasp.cipher_modules.report import Report
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher
cipher = AradiBlockCipher()
test = AvalancheTests(cipher).avalanche_tests(number_of_samples=1000)
report = Report(test)
report.show(test_name="avalanche_entropy_vectors")
```

Listing 1.2. Avalanche entropy script (14m,47.6s in CLAASP-WEB)

```
plot = AvalancheTests(cipher).generate_3D_plot(number_of_samples=1000,
    criterion="avalanche_weight_vectors")
```

Listing 1.3. Avalanche weight script (13m,58.1s in CLAASP-WEB)

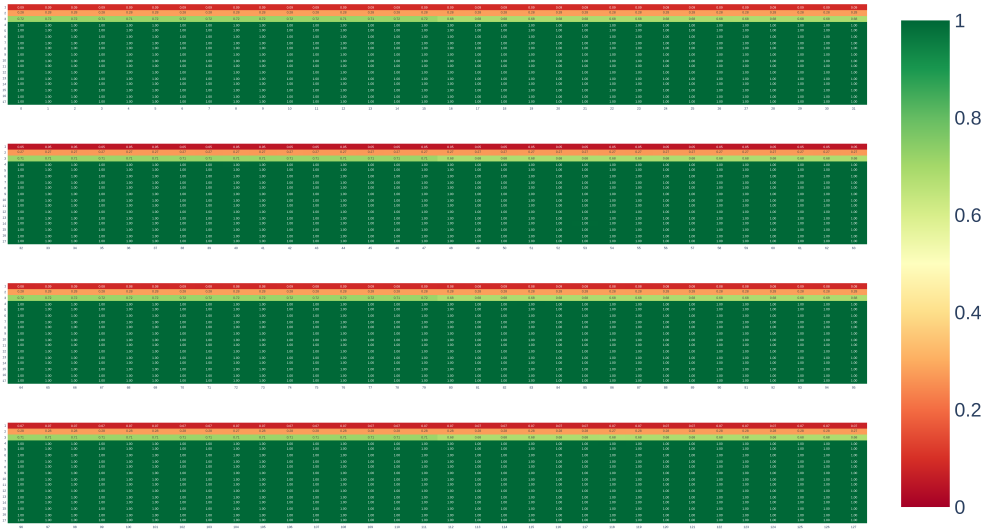

Fig. 1. Avalanche Entropy. Each rectangle represents 32 bits of the ARADI state

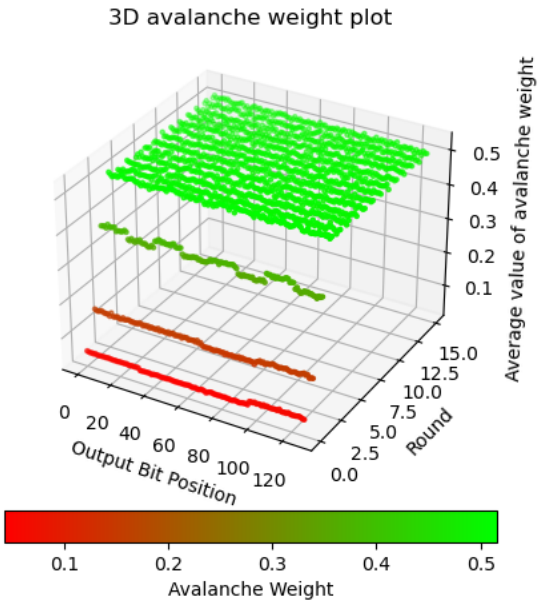

Fig. 2. Avalanche weight

4 Statistical Tests

In this section we provide a description and our findings about the statistical properties of ARADI observed as a black box.

4.1 Description

During the selection process for the Advanced Encryption Standard (AES), statistical tests, which are well-known tools for verifying randomness, were used to evaluate the applicability of the proposed ciphers as random number generators [43,6]. The same techniques can be used as a quick preliminary assessment of the cipher's resistance to linear and differential cryptanalysis. One of the most popular tool to perform statistical tests is the NIST Statistical Test Suite (NIST STS)[44]. Other well-known test suites are Marsaglia's DIEHARD tests [39] and Brown's DIEHARDER tests [19].

To conduct statistical tests on a cipher, we first encrypt a specific data format using the cipher with varying numbers of rounds to generate a series of bit-streams, referred to as a dataset. The randomness of this dataset is then assessed using statistical tools, which serve as randomness distinguishers. If the dataset passes the tests at a certain round r, it is considered indistinguishable from a random dataset, indicating that the cipher remains secure after r rounds.

CLAASP was already used in the past to analyze the finalists of the NIST lightweight ciphers standardization process [14,15]. CLAASP allows the user to perform the previous analysis using either the NIST STS or the DIEHARDER tools. Here, we apply the same analysis to ARADI using NIST STS. While DIEHARDER includes a much larger number of statistical tests, in our experience their execution is much slower, and often the results do not improve over NIST STS.

Dataset setting Nine datasets were proposed to evaluate the randomness of ciphers during the AES selection process [43,6]. Based on the results from the AES analysis and the cryptanalysis of NIST lightweight ciphers conducted in [14], the most effective datasets are the *avalanche*, *low-density*, and *high-density* datasets. In this paper, we utilize these same datasets to assess the randomness of the ARADI cipher.

To achieve a significance level of $\alpha = 0.01$, which corresponds to a false positive rate of 1 in 100 tests, each dataset must be processed with more than 100 sequences. We adhere to the parameter settings outlined in [43].

CLAASP scripts We performed the NIST STS tools included in CLAASP [15] over 3 days on a 4-core 3.8GHz Intel CPU with 256GB RAM. The script to run the experiments is presented in Listing 1.4. The `index = [0, 1]` are used to select the analysis with respect to the plaintext and the key, respectively.

```
from claasp.cipher_modules.statistical_tests.nist_statistical_tests import
    NISTStatisticalTests
from claasp.cipher_modules.report import Report
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher
cipher = AradiBlockCipher()
test = NISTStatisticalTests(aradi)
index = [0,1]
dataset = ["avalanche", "low_density", "high_density"]
for d in dataset:
    for i in index:
        test_results = test.nist_statistical_tests(d, input_index=i)
        report = Report(test_results)
        report.show()
```

Listing 1.4. CLAASP script to perform the statistical tests with NIST STS over 6 dataset types.

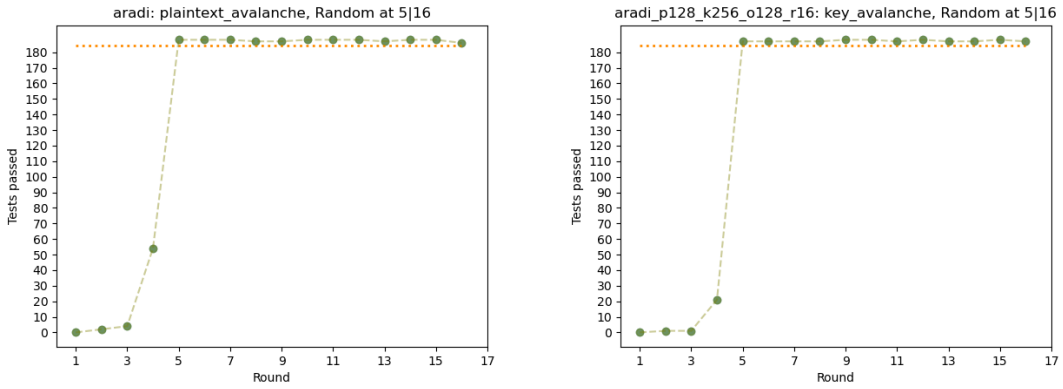

Fig. 3. NIST statistical tests for plaintext and key avalanche datasets of ARADI

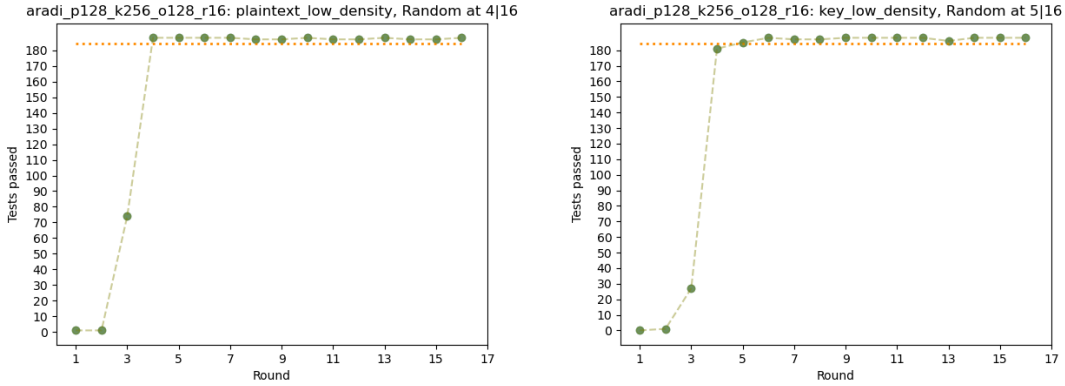

Fig. 4. NIST statistical tests for low-density plaintext and key datasets of ARADI

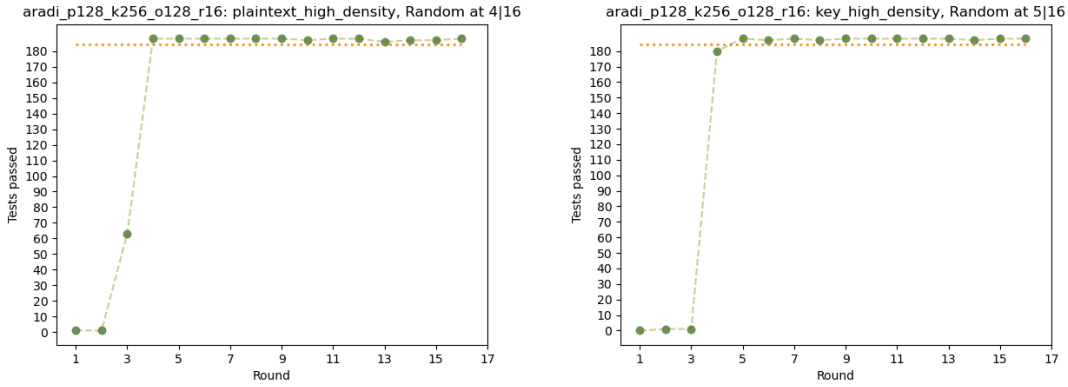

Fig. 5. NIST statistical tests for high-density plaintext and key datasets of ARADI

4.2 Findings

From Figure 3, Figure 4 and Figure 5 we notice that NIST STS can not distinguish ARADI's output from a random sequence after round 5 (out of 16) rounds. These results are in line with other ciphers such as ASCON [14].

5 Continuous Diffusion Tests

In this section we provide a description and our findings about the continuous diffusion properties of ARADI.

5.1 Description

In [21], Coutinho *et al.* present a framework for constructing functions over the real numbers to model some properties of Boolean functions. By making specific independence assumptions, these continuous functions capture the probability (or correlation) of differential (linear) propagation between input and output differences (masks). Leveraging this framework, they generalize several cryptographic operations, enabling the development of models over the real numbers of entire cryptographic algorithms to study the propagation of differences (masks) between inputs and outputs.

Based on these models of cryptographic algorithms, they define three metrics: the Continuous Avalanche Factor (CAF), the Continuous Neutrality Measure (CNM), and the Diffusion Factor (DF). The CAF serves as the continuous analogue of the avalanche factor [24], which measures the proportion of output bits that change, on average, when the Hamming distance between input pairs is 1; for a random permutation, this proportion is expected to be 0.5.

In the continuous framework, the Hamming distance is replaced by the L2 norm, or Euclidean Distance (ED), to evaluate the CAF. The CAF measures the average variation in the output of a continuous algorithm model when the probability of a given input bit being 1 is perturbed by a small real value, denoted as λ. Specifically, it assesses how the ED between outputs $y_0 = f(x_0)$ and $y_1 = f(x_1)$, for $x_0, x_1 \in \mathbb{B} = \{x \in \mathbb{R} \colon -1 \leq x \leq 1\}$, changes on average when the ED between x_0 and x_1 is smaller than λ. For pseudo-random permutations, it is expected that even small perturbations in λ will lead to larger average EDs in the propagation of these variations. For further details on the continuous neutrality measure and diffusion factor, we refer the reader to [21].

5.2 Findings

We present the CAF evaluation of ARADI, subject to $\lambda = 0.001$, in Table 3. In Table 3, we also provide the comparison with other ciphers such as Speck128-256 [7], AES-128, ASCON320 [25] permutation, the low-latency block ciphers QARMAv2 [3] and SPEEDY [35]. ARADI demonstrates slower diffusion of differences compared to AES-128, despite both having the same block size. However, it achieves better diffusion than Speck128-256, which also shares the same block size as ARADI. Its CAF values begin to show notable non-zero increases around round 6, reaching a value of 0.725 by round 7. This indicates that ARADI requires 7 rounds to begin propagating input differences significantly under this metric. By round 8, it achieves a CAF of 0.955, indicating that by this point, it has reached a strong diffusion under this metric.

Figure 6 shows the comparison of ARADI for different values of λ used to compute the CAF value with 4000 random samples[1]. The results indicate that as λ decreases, the number of rounds required to reach the maximum CAF increases. This behavior is expected, as smaller perturbations in input (represented by smaller λ values) lead to slower propagation of differences through the cipher, requiring more rounds to fully diffuse the input changes and reach a maximum CAF. The script used to generate Figure 6 is shown in Listing 1.5. This script generates a list of JSON lines, each corresponding to the results for a specific lambda value. The lines were sent to Elasticsearch [26], and the figure was created using Kibana [27]. Alternatively, as for other tests, each entry of the **results** variable of the script could have been passed to the `Report` class.

The experiments were conducted using the continuous diffusion analysis module of CLAASP [11] on a Ubuntu 22.04.1 machine equipped with 256 AMD core processors and 1TB of memory. When comparing Table 3 to Table 2 in [21], we observed slight variations in the CAF values. This difference is due to our use of the Python Decimal package to handle small numbers, while the

[1] We repeated the experiment 6 times with 4,000 random samples. The maximum CAF standard deviation was 0.014 at round 7, which is low compared to the theoretical maximum of 0.5 and acceptable for this sample size given the expected small fluctuations of this metric.

implementation of Table 2 in [21] employed the Relic library [1]. For instance, for five rounds of AES-128, we obtained a value of 0.777, whereas [21] reports 0.734.

Table 3. Continuous avalanche factor comparison for AES-128, ASCON320, QARMAv2, Speck128-256, ARADI, and SPEEDY using $\lambda = 0.001$. The script used to generate this table took around 8m.

Rounds	AES-128	ASCON320	QARMAv2	Speck128-256	ARADI	SPEEDY
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0.172	0	0	0
4	0	0	0.884	0	0	0
5	0.777	0.008	0.999	0	0.001	0.505
6	0.971	0.761	-	0	0.126	0.959
7	0.999	0.962	-	0.001	0.725	0.999
8	-	0.998	-	0.055	0.955	-
9	-	0.999	-	0.286	0.997	-
10	-		-	0.597	-	-
11	-	-	-	0.804	-	-
12	-	-	-	0.929	-	-
13	-	-	-	0.976	-	-
14	-	-	-	0.995	-	-

CLAASP scripts The script in Listing 1.5 was used to generate Figure 6.

```
import json
from claasp.cipher_modules.continuous_diffusion_analysis import
    ContinuousDiffusionAnalysis
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher

number_of_rounds = 12
cipher = AradiBlockCipher(number_of_rounds=number_of_rounds)
cda = ContinuousDiffusionAnalysis(cipher)
lambda_values = [0.1, 0.01, 0.02, 0.001, 0.02]
results = []
for lambda_value in lambda_values:
    cda = ContinuousDiffusionAnalysis(cipher)
    test_args = {
        'is_continuous_neutrality_measure': False,
        'is_diffusion_factor': False,
        'continuous_avalanche_factor_number_of_samples': 4000,
        'threshold_for_avalanche_factor': lambda_value
    }
    result = cda.continuous_diffusion_tests(**test_args)
    result['input_parameters']['cipher'] =
        str(result['input_parameters']['cipher'])

    round_output_results =
        result['test_results']["plaintext"]["round_output"]
    plaintext_values =
        round_output_results["continuous_avalanche_factor"][0]["values"]
    round_number_caf_values = [
        {"round_number": idx + 1, "caf_value": val} for idx, val in
            enumerate(plaintext_values)
    ]

    result.update({
        "round_number_caf_values": round_number_caf_values,
```

```
            "number_samples": number_samples,
            "lambda_value": lambda_value,
            "cipher": "_".join(cipher.id.split("_")[:-1])
        })

        results.append(result)

with open("caf_results.txt", "a") as json_file:
    for res in results:
        json.dump(res, json_file)
        json_file.write("\n")
```

Listing 1.5. Continuous avalanche factor script created with CLAASP for generating data for multiple rounds and λ values. The data was processed and visualized using Elasticsearch and Kibana.

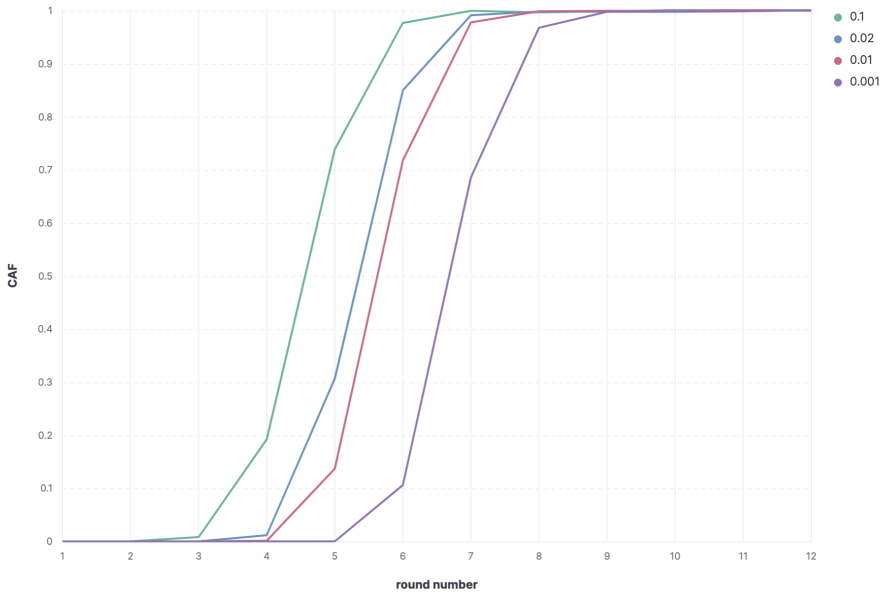

Fig. 6. ARADI: Number of rounds against CAF for several λ values. The script shown in Listing 1.5 which was used to generate the data for this figure, completed its execution in 5 minutes.

6 Differential and Linear Analysis

In this section we provide a description and our findings about the differential and linear properties of ARADI.

6.1 Description

Differential cryptanalysis Differential cryptanalysis is one of the main statistical cryptanalysis technique against symmetric ciphers [16]. Consider a function f defined over $\mathbb{F}_2^n$. A differential on f is a pair $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, with $\alpha \neq 0$, and its probability is defined as

$$\mathsf{DP}_f(\alpha, \beta) = 2^{-n} \cdot |\{x \in \{0,1\}^n \mid f(x) \oplus f(x \oplus \alpha) = \beta\}|.$$

For a differential (α, β), we define its weight as $\mathbf{w}_d(\alpha, \beta) = -\log_2(\mathsf{DP}_f(\alpha, \beta))$. For an r-round iterative cipher given by $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$, an r-round differential trail is a sequence $(\alpha_0, \alpha_1, \cdots, \alpha_r)$ where each (α_i, α_{i+1}) is a differential on f_i (with $\mathsf{DP}_{f_i}(\alpha_i, \alpha_{i+1}) > 0$) and its

probability (assuming independence of rounds) is defined as $\prod_{i=0}^{r-1} \mathsf{DP}_{f_i}(\alpha_i, \alpha_{i+1})$. The weight of the trail is defined as $\sum_{i=0}^{r-1} \mathbf{w}_d(\alpha_i, \alpha_{i+1})$ where $\mathbf{w}_d(\alpha_i, \alpha_{i+1}) = -\log_2(\mathsf{DP}_{f_i}(\alpha_i, \alpha_{i+1}))$. In our search of differential trails, we are interested in trails having low weights.

Linear cryptanalysis The linear cryptanalysis method introduced by Matsui [40] is another statistical cryptanalysis technique against symmetric ciphers. Consider a function f defined over $\mathbb{F}_2^n$. The main idea behind linear cryptanalysis is to find a pair of input and output masks $(\gamma, \delta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ of f having high bias ϵ, given by

$$\epsilon(\gamma, \delta) = \frac{\#\left\{x \in \mathbb{F}_2^n \mid \gamma^{\mathrm{T}} x + \delta^{\mathrm{T}} f(x) = 0\right\}}{2^n} - \frac{1}{2}.$$

The correlation $\mathrm{C}(\gamma, \delta)$ equals $2 \cdot \epsilon(\gamma, \delta)$. Since the correlation is either positive or negative, we are usually interested in input and output masks for which the squared correlation $\mathrm{C}^2(\gamma, \delta)$ is high.

Now, similar to differentials, we define the correlation weight as $\mathbf{w}_l(\gamma, \delta) = -\log_2(\mathrm{C}^2(\gamma, \delta))$. For an r-round iterative cipher $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$, an r-round linear trail is a sequence of masks $(\gamma_0, \gamma_1, \cdots, \gamma_r)$ where each (γ_i, γ_{i+1}) is an input and output mask on f_i (with $\mathrm{C}^2_{f_i}(\gamma_i, \gamma_{i+1}) > 0$) and its squared correlation (assuming independence of rounds) is defined as $\prod_{i=0}^{r-1} \mathrm{C}^2_{f_i}(\gamma_i, \gamma_{i+1})$. The correlation weight of the trail is defined as $\sum_{i=0}^{r-1} \mathbf{w}_l(\gamma_i, \gamma_{i+1})$ where $\mathbf{w}_l(\gamma_i, \gamma_{i+1}) = -\log_2(\mathrm{C}^2_{f_i}(\gamma_i, \gamma_{i+1}))$.

In case of both differential and linear trails, we say an r-round trail with weight $\mathbf{w}$ is *optimal* if a r-round trail with weight less than $\mathbf{w}$ does not exist.

6.2 Findings

We report the weights of best found differential and linear trails of ARADI in Table 4. The differential and linear trails in the table are optimal up to 10 rounds and 8 rounds, respectively.

Table 4. Weights of best found differential and linear trails of ARADI in single key setting, using ParKissat-RS. Optimal weights are highlighted in bold. Experiments were run on a machine with 2× Intel(R) Xeon(R) Platinum 8490H Processors (total of 240 CPUs), 8× NVIDIA H100 80GB HBM3 GPUs, and 4TB of RAM.

Round	Differential		Linear	
	Weight	Time (sec.)	Weight	Time (sec.)
1	**2**	3.10	**1**	2.08
2	**8**	9.53	**4**	5.31
3	**14**	16.38	**7**	8.72
4	**32**	64.66	**16**	31.13
5	**50**	238.73	**25**	84.05
6	**84**	5,359.51	**41**	1,600.19
7	**106**	54,630.40	**53**	15,052.11
8	**122**	$>$ 3 days	**62**	49,460.84
9	**137**	$>$ 3 days	71	$>$ 3 days
10	**143**	$>$ 3 days	-	-

An example of 9-round differential trail with weight 137 is given in Table 5. By fixing the input and output difference of this trail, we enumerated all trails satisfying this fixed pair of difference (using Listing 1.8) and multiplied all trails differential probabilities to obtain the total probability of the differential. In other words, the overall probability of the input difference

$$\begin{aligned}
\Delta w^0 &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Delta x^0 &= 1\ldots\ldots\ldots\ldots\ldots 1\ldots\ldots\ldots 1. \\
\Delta y^0 &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Delta z^0 &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots
\end{aligned}$$

going to the output difference

$$\begin{aligned}\Delta w^9 &= \texttt{...1....1.................1....}\\ \Delta x^9 &= \texttt{.....................................}\\ \Delta y^9 &= \texttt{.....................................}\\ \Delta z^9 &= \texttt{.....................................}\end{aligned}$$

after 9 rounds is approximately $2^{-125.5}$.

Table 5. A 9-round differential trail with weight 137.

Round i	Δw^i	Δx^i	Δy^i	Δz^i	Weight
0	..... 1.....1.....	.....	.....	.....	6
1	.....	.....1.....	.....	.....	2
2	.....	...1.....1.....1.....	.....	.....	6
3	...1..1...1.....1.....1...1	1..1..1...1.....11.....1..11	.....	.....	18
4	1.111.11..11.11..1.11.1...111111	1.111..1..11.11..1.1111...111111	.....	.....	42
5	11.11...111.....11.1.1...1...1.1	11.11.1.11.1...11.11.....1...1.1	.....1.....	.....	37
6	..11.....111....1.....1....1	..11.....11....1.....1..1.....	.....	.....	18
7	...1.....1.....1.....	.....1.....	.....	.....	6
8	...1.....	.....	.....	.....	2
9	...1....1.....1....	.....	.....	.....	

The 8-round differential with probability 2^{-109} is given as follows.

$$\begin{aligned}\Delta w^0 &= \texttt{.....................................}\\ \Delta x^0 &= \texttt{.........1....1..1.............}\\ \Delta y^0 &= \texttt{.....................................}\\ \Delta z^0 &= \texttt{.........1....1..1.............}\\ \Delta w^8 &= \texttt{1....1.11....11......111....111.}\\ \Delta x^8 &= \texttt{.....................................}\\ \Delta y^8 &= \texttt{1..1.1.11....11...1..111..1.111.}\\ \Delta z^8 &= \texttt{.....................................}\end{aligned}$$

An example of a 9-round linear trail is presented in Table 6. Similarly to what we did for the 9-round differential trail, we fixed the input and output masks for the 9-round linear trail. We then enumerated all trails that satisfy this fixed mask pair and multiplied all the trail correlations to compute the total correlation. Specifically, we found 120 solutions with a correlation of 2^{-71} and at least 520 solutions with a correlation of 2^{-72}. The total correlation, taking into account the linear hull effect, is $2^{-62.41}$. Due to computational resource limitations, we did not continue further; otherwise, this correlation could potentially be even greater than $2^{-62.41}$.

Table 6. A 9-round linear trail with correlation 2^{-71}.

Round i	Δw^i	Δx^i	Δy^i	Δz^i	Weight
0	.....	.....	....1.....11.....1.....1..	.....	5
1	.....	.....	....1.1....1...1.....1.....	.....	5
2	..1.....1.....1.....1.....	..1.....1.....1.....1.....	1.1.111..1.1..11.1.1..1..11....1	..1.....1.....1.....1.....	15
3	1..1.....1..1..1.1...1111....1.	1..1.....1..1..1.1...1111....1.	1..1..111.1..11.111.1.1111..111.	1..1.1....11.1..1.1...1111....1.	21
4	11....1..1.....1..1.....1.	11....1..1.....1..1.....1.	11....1..1.....1..1.....1.	11....1..1.1.....1..1.....1.1.	9
5	.....1.....	.....1.....	.....1.....	1.....1.....1.....	3
6	.....	.....	.....	.....1.....	1
7	.....	.....	.....	.....1.....1....1.....	3
8	.....	.....11.....11.....1.....1..	.....	.....11....111....1.1....11..	9
9	.1....1.....1..1..1.1.....	..1.11.1111.1111....11.11..1...1	.1....1.....1..1..1.1.....	..1.11.1111.1111....11.11..1...1	

CLAASP scripts The following scripts (Listing 1.6 - Listing 1.8) can be used to reproduce the results mentioned in the previous section. Notice that trails up to 5 rounds can be obtained in seconds while it may take a bit longer to reproduce results for 6 or more rounds.

```
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
from
    claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model
    import SatXorDifferentialModel
cipher = AradiBlockCipherSBox(number_of_rounds=4)
model = SatXorDifferentialModel(cipher)
trail = model.find_lowest_weight_xor_differential_trail()
```

Listing 1.6. CLAASP optimal differential trail search of 4 rounds ARADI

```
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
from claasp.cipher_modules.models.sat.sat_models.sat_xor_linear_model
    import SatXorLinearModel
cipher = AradiBlockCipherSBox(number_of_rounds=4)
model = SatXorLinearModel(cipher)
trail = model.find_lowest_weight_xor_linear_trail()
```

Listing 1.7. CLAASP optimal linear trail search of 4 rounds ARADI

```
from claasp.cipher_modules.models.utils import set_fixed_variables,
    integer_to_bit_list
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
cipher = AradiBlockCipherSBox(number_of_rounds=9)
from
    claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model
    import SatXorDifferentialModel
key = set_fixed_variables(component_id='key', constraint_type='equal',
    bit_positions=range(256), bit_values= [0]*256)
pt = set_fixed_variables(component_id='plaintext',
    constraint_type='equal', bit_positions= range(128),
    bit_values=integer_to_bit_list(0x00000000400010020000000000000000,
    128, 'big'))
ct = set_fixed_variables(component_id='intermediate_output_7_61',
    constraint_type='equal', bit_positions= range(128),
    bit_values=integer_to_bit_list(0x10800080000000000000000000000000,
    128, 'big'))
sat = SatXorDifferentialModel(cipher)
trail = sat.find_all_xor_differential_trails_with_weight_at_most(137, 137,
    fixed_values=[pt, ct,  key])
```

Listing 1.8. CLAASP differential trails enumeration of 9 rounds ARADI

7 Impossible Differential Cryptanalysis

In this section we provide a description and our findings about the impossible differentials of ARADI.

7.1 Description

Impossible differential cryptanalysis, introduced by Knudsen in [32], is the study of differentials with probability 0.

The search for impossible differentials $\delta \nrightarrow \gamma$ has mostly moved from custom search algorithms, such as the $\mathcal{U}$ and UID methods, to automated techniques based on SAT, MILP or CP solvers. Among these automated techniques, two different approaches have been proposed. One, based on miss-in-the-middle [45], looks for a contradiction between the deterministic forwards (resp. backwards) propagation of δ and γ. The second approach, initiated in [22], enumerates low Hamming weight candidate δ, γ, and looks for a (non-truncated) differential trail with input difference δ and output difference γ. In CLAASP, this second approach is implemented.

In this section, δ^r denotes the difference at the start of round r. We denote by $\delta^r_{i_0 \dots i_k}$ a 128-bit difference vector where only bits $i_0 \dots i_k$ are *active* (non-zero). Similarly, we denote by $\Delta^r_{n_0 \dots n_k}$ a 32-bit truncated difference vector where only nibbles $n_0 \dots n_k$ are active, where n_i is the concatenation of bits $w^r_i, x^r_i, y^r_i, z^r_i$. Finally, with an asterisk, $\Delta^{r*}_{n_0 \dots n_k}$ we indicate that the active nibbles $n_0 \dots n_k$ are all equal.

7.2 Findings

We find a class of 230,400 impossible differentials for 8 rounds of ARADI, derived from 230,400 7-round simple impossible differentials found using CLAASP.

The 7-round impossible differentials are obtained using a straightforward application of the technique from [22], i.e., by enumerating all Hamming weight 1 input and output differences δ^0_i and δ^7_j, and checking satisfiability. We find that all the corresponding $128 \cdot 128 = 16384$ possible combinations correspond to impossible differentials.

We then slightly modify the CLAASP search to consider active nibble patterns $\Delta^0_{n_i}$, $\Delta^7_{n_j}$, and find that all 7-round transitions from 1 active nibble to 1 active nibble are impossible differentials, i.e.,

$$\forall_{i,j \in [0,31]^2} \Delta^0_{n_i} \nrightarrow \Delta^7_{n_j}$$

For 8-rounds, we do not find any impossible differentials using this simple approach. On the other hand, we observe that, by construction of the linear layer Λ_i of ARADI, $\Delta^7_{n_i}$ is deterministically reached by propagating backwards the round 8 difference $\Delta^{8*}_{n_0,n_1,n_2} = \Lambda_7(\Delta^7_{n_i})$.

Therefore, for each 7-round impossible differential of the form $\Delta^0_{n_i} \nrightarrow \Delta^7_{n_j}$, a set of 15 impossible differentials can be built by iterating over the value of the 3 active (and equal) nibbles of round 8, totalling $15 \cdot 32 \cdot 15 \cdot 32 = 230400$ impossible differentials for 8 rounds.

CLAASP scripts The following scripts reproduce the experiments mentioned in the previous subsection. Each unsatisfiable instance (pair of input and output differences) is solved in less than a second, and while the following script is sequential, solving the instances concurrently is possible, because they are independent.

The first experiment, iterating over Hamming weight 1 input differences, can be run with the Listing 1.9 script.

```
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
cipher = AradiBlockCipherSBox(number_of_rounds=4)
id_list = cipher.impossible_differential_search("cp", solver='Chuffed')
```

Listing 1.9. CLAASP Hamming weight 1 impossible differential enumeration

The Listing 1.9 script iterates through all Hamming weight 1 input and output difference pairs for 7 rounds, outputs solving statistics each time a new impossible differential is identified, and returns the list `id_list` of pairs δ^0_i, δ^7_j such that $\delta^0_i \nrightarrow \delta^7_j$ (represented as pairs of integers).

In the second experiment, we enumerate through all 7 rounds nibble activity patterns with a single active nibble.

```
import numpy as np
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
from claasp.cipher_modules.models.utils import set_fixed_variables,
    integer_to_bit_list
cipher = AradiBlockCipherSBox(number_of_rounds=7)
model = cipher.get_model('cp', 'xor_differential')
search_function = model.find_one_xor_differential_trail
last_component_id = cipher.get_all_components()[-1].id
id_list = []
inputs_dictionary = cipher.inputs_size_to_dict()
plain_bits = inputs_dictionary['plaintext']
key_bits = inputs_dictionary['key']
for input_word_position in range(32):
```

```
zero_bits_in_input = [i for i in range(128) if (i%32) !=
    input_word_position]
non_zero_bits_in_input = [32*i + input_word_position for i in range(4)]
for output_word_position in range(32):
    zero_bits_in_output = [i for i in range(128) if (i%32) !=
        output_word_position]
    non_zero_bits_in_output = [32*i + output_word_position for i in
        range(4)]
    fixed_values = []
    fixed_values.append(set_fixed_variables('key', 'equal',
        list(range(key_bits)),
                            integer_to_bit_list(0, key_bits, 'big')))
    fixed_values.append(set_fixed_variables('plaintext', 'equal',
        zero_bits_in_input,
                            integer_to_bit_list(0,
                                plain_bits-4,'big')))
    fixed_values.append(set_fixed_variables(last_component_id,
        'equal', zero_bits_in_output,
                            integer_to_bit_list(0,
                                plain_bits-4,'big')))
    fixed_values.append(set_fixed_variables('plaintext', 'not_equal',
        non_zero_bits_in_input,
                            integer_to_bit_list(0,
                                plain_bits-4,'big')))
    fixed_values.append(set_fixed_variables(last_component_id,
        'not_equal', non_zero_bits_in_output,
                            integer_to_bit_list(0,
                                plain_bits-4,'big')))
    solution = search_function(fixed_values, solver_name='Chuffed')
    if solution['status'] == "UNSATISFIABLE":
        id_list.append((input_word_position, output_word_position))
```

Listing 1.10. CLAASP single active nibble impossible differential enumeration

The Listing 1.10 script returns the list `id_list` of pairs of integers (i, j) such that $\Delta_i^0 \nrightarrow \Delta_j^7$.

8 Algebraic Analysis

In this section we provide a description and our findings about the algebraic properties of ARADI.

8.1 Description

Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2^n$ be an n-bit block cipher with m-bit secret key. Then the algebraic normal form (ANF) of a output bit of E can be written as

$$E(x,k) = \sum_{u \in \mathbb{F}_2^n,\, v \in \mathbb{F}_2^m} a_{u,v} x^u k^v, \tag{5}$$

where $a_{u,v} \in \{0,1\}$, $x^u = \prod_i x_i^{u_i}$ and $k^v = \prod_i k_i^{v_i}$. In case of a fixed key k, we consider the mapping $E_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$ where $E_k(x) = E(x,k)$. Thus, the ANF of E_k is given by

$$E_k(x) = \sum_{u \in \mathbb{F}_2^n} \Big(\sum_{v \in \mathbb{F}_2^m} a_{u,v} k^v \Big) x^u = \sum_{u \in \mathbb{F}_2^n} f_u(k) x^u, \tag{6}$$

where $f_u(k) = \sum_{v \in \mathbb{F}_2^m} a_{u,v} k^v$. The algebraic degrees of E and E_k are different and given as follows.

$$\deg(E) = \max_{u \in \mathbb{F}_2^n,\, v \in \mathbb{F}_2^m} \{\mathrm{Hw}(u) + \mathrm{Hw}(v) \mid a_{u,v} \neq 0\} \tag{7}$$

$$\deg(E_k) = \max_{u \in \mathbb{F}_2^n} \{\mathrm{Hw}(u) \mid f_u(k) \neq 0\}. \tag{8}$$

From the design perspective, the degree of E_k should be high; otherwise lower degree leads to higher-order differential or integral distinguishers [34,33].

We use two approaches to evaluate the algebraic behavior of ARADI.

– CLAASP's algebraic testing module, which symbolically models the cipher for a specified number of rounds. It then attempts to solve the resulting system of polynomials using Gröbner bases. If the system is solvable within the allocated time, it indicates that the cipher is not algebraically secure for that number of rounds.
– Division property [46,47,31] methods to evaluate the upper bounds of $\deg(E_k)$ and to find vectors u satisfying $f_u(k) = 0$ for all k. Given a number of rounds of a chosen cipher and a chosen output bit, CLAASP's division property module [2] (see Listing 1.12) produces a model that can either obtain the ANF of this chosen output bit, or find the degree of this ANF, or check the presence or absence of a specified monomial.

8.2 Findings

With CLAASP's algebraic test module as given in Listing 1.11, we find that two rounds of ARADI are solvable within 10 seconds, indicating the cipher is not secure against algebraic attacks up to this round. Due to the known limitations of Groebner basis solvers, we did not further increase the time threshold above 10 seconds. Therefore, the only conclusion we can currently draw is that on the tested architecture, with the solvers available in SageMath, solving the CLAASP-generated system for 3 rounds or more requires more than 10 seconds. The experiment was conducted on an Apple M1 Max with 32 GB of RAM.

Using the path search algorithm from [46], the number of chosen plaintexts required to construct integral distinguishers for the ARADI block cipher is presented in Table 7. Note that the numbers in the table give the upper bound on the algebraic degree (in plaintext variables) which may not be tight.

Table 7. Number of chosen plaintexts required to construct r-round integral distinguishers for ARADI.

Round r	2	3	4	5	6	7
$\log_2$(data)	4	12	28	84	113	124

The number of plaintexts in Table 7 can be reduced by exploiting the ANF of the ARADI S-box. More precisely, by setting $y = 0$, $z = 0$, and taking the remaining 64 variables in w, x as active, we can partially linearize the first round. Consequently, the algebraic degrees in the next rounds will not grow at a large rate. The degree evolution for up to 5 rounds is shown in Table 8. Notice that the maximum degree is 60 after 5 rounds. This means taking any 61 out of 64 variables from w and x as active variables will give a distinguisher with complexity 2^{61}.

Table 8. Reducing algebraic degree growth.

Round	Degree of word			
	w	x	y	z
0	1	1	0	0
1	1	1	2	1
2	4	3	4	3
3	11	8	11	8
4	30	22	30	22
5	64	60	64	60

One of the purpose of the CLAASP's division property module is to verify experimentally the degree bounds shown in Table 8. By running the script given in Listing 1.12 for ARADI, one can verify that the degrees for 1 and 2 rounds are indeed tight. Ideally, we would like to show with the help of CLAASP that for the next rounds, the actual degrees are less than the bounds in Table 8.

[2] This module is not yet in the main branch, one can find it in the branch: https://github.com/Crypto-TII/claasp/tree/feat/division_property

Currently, the model given by CLAASP is too huge to obtain results for 3 or more rounds in a reasonable time. We are currently working on simplifying the model. The experiments for 2 rounds were conducted on a 2x AMD EPYC 7763 64-core processor and the results were obtained in less than 6 seconds.

```
from claasp.ciphers.block_ciphers.aradi_block_cipher_sbox import
    AradiBlockCipherSBox
from claasp.cipher_modules.algebraic_tests import AlgebraicTests
aradi = AradiBlockCipherSBox(number_of_rounds=3)
test = AlgebraicTests(aradi)
result=test.algebraic_tests(timeout_in_seconds=10)
```

Listing 1.11. CLAASP algebraic test script for 3 rounds ARADI

```
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher
cipher = AradiBlockCipher(number_of_rounds=2)
from claasp.cipher_modules.division_trail_search import *
model = MilpDivisionTrailModel(cipher)
model.find_degree_of_specific_anf(0)
```

Listing 1.12. CLAASP division property script for finding the degree of bit 0 after 2 rounds

9 Neural Distinguishers

In this section we provide a description and our findings about ARADI's resistance against neural distinguishers.

9.1 Description

Neural differential cryptanalysis, which applies deep learning to modern block cipher differential cryptanalysis, was introduced by A. Gohr in his seminal paper at CRYPTO 2019 [29]. In neural differential cryptanalysis, a deep neural network $\mathcal{ND}$ is trained to distinguish between ciphertext pairs C_0, C_1 derived from plaintext pairs P_0, P_1 with a fixed input difference (δ) and those originating from a random input difference (rand). Gohr's work challenged conventional wisdom in two ways: *i)* the neural distinguisher for SPECK32, reduced to 8 rounds, exhibited higher accuracy compared to pure differential distinguishers, and *ii)* the resulting neural differential distinguishers enabled a new state-of-the-art key recovery complexity for 11 rounds of SPECK32. Gohr's success has motivated extensive follow-up research in neural differential cryptanalysis over the past five years, leading to a recent systematization of knowledge in [28]. However, the application to new cryptanalytic primitives has remained challenging, with the obtained distinguishers being highly specialized and not easily applicable to other primitives.

The most generic neural distinguisher pipeline to date – AutoND – was presented at FSE'24 [12]. In AutoND, an evolutionary algorithm first automatically searches for optimal input differences for neural distinguishers and rates them using a *bias score*, which quantifies the average bias of output difference bits through sampling. Next, the best input differences are used to train a neural network known as DBitNet, which is generic in that it does not contain cipher-specific elements in its architecture. AutoND operates in two scenarios: the 'single-key' and 'related-key' scenario. In the single-key scenario, each ciphertext in a sample pair C_0, C_1 is encrypted with the same key, while in the related-key scenario, the key for C_1 is related to the key for C_0 by a fixed key input difference.

9.2 Findings

ϵ-close input differences Table 9 shows the total number of input differences found by the evolutionary optimizer of the AutoND pipeline for ARADI compared to SIMON and SPECK (with similar block size). In SIMON, the rotational equivalence of many found input differences results in a high number of input differences with virtually the same bias score. SPECK lacks such a property, and at most one or two input differences δ can be clearly identified for further investigation in the pipeline.

Table 9. The total number of differences returned by our optimizer for each cipher, and the number of ϵ-close solutions for $\epsilon \in \{0.01, 0.1, 0.25\}$, where ϵ-close denotes differences for which the score differ at most by a factor ϵ to the optimal score. ARADI $^{\text{RK}}$ shows the optimizer results in the related-key scenario.

Primitive	Total	0.01-close	0.1-close	0.25-close	Source
SIMON128	266	64	64	64	[12]
SPECK128	156	1	1	1	[12]
ARADI	80	13	58	79	*This Work*
ARADI $^{\text{RK}}$	110	1	15	48	*This Work*

The diffusion properties of the linear layer make it so that one active S-box in the input difference goes to 3 active S-boxes at round 2, and 9 at round 3, with probability 1. Therefore, such input differences lead to heavily biased (towards 0) difference bit positions at round 3; these are correctly found by the optimizer, which identifies biases up to round 3. In round 4, individual bit biases are still present, but too low to trigger the default detection threshold of AutoND, which favours heavier biases in lower rounds to high biases in higher rounds.

Neural distinguishers The AutoND pipeline trains neural distinguishers using the best input differences identified by the optimizer in Table 9. The number and composition of training and validation samples are chosen identical to [29,12][3].

Table 10. Results of the AutoND training pipeline for the best input difference found for ARADI (Table 9). $\mathcal{ND}_{\text{DBitNet}}$ is the validation accuracies of the DBitNet neural distinguisher in the respective round of ARADI. Validation accuracies below 0.5050 (ten standard deviations away from 0.50) are considered compatible with a random guess and are greyed out. The neural distinguisher training stops in this round.

Primitive	Difference	Round	$\mathcal{ND}_{\text{DBitNet}}$
ARADI	`0x10000000000000000000`	4	1.0
		5	0.5954
		6	0.5010
ARADI $^{\text{RK}}$	(`0x0`,	4	1.0
	`0x2000000000000000000`...[4])	5	0.9960
		6	0.5631
		7	0.5007

Table 10 presents the results of running the CLAASP AutoND module on ARADI in both the single-key and related-key scenarios. In single-key, the neural distinguisher reaches 100% accuracy for 4 rounds, and 60% for 5 rounds. These results complement the diffusion tests (section 3): at round 4, for our Hamming weight 1 input difference, relatively high bit biases are still present, and

[3] In detail that means that each sample consists of a concatenated ciphertext pair $C_0|C_1$ that can originate from a plaintext pair (P_0, P_1) related by the chosen input difference δ, or a random input difference. In total 10^7 training samples, and 10^6 validation samples are used. Half of the training, respectively validation samples originate from δ, the other half from random input differences.

[4] `0x20000000000000000000000000000000000000000000000000000000`

5 S-box positions (4, 10, 14, 19, 21) are active with probability 1. At round 5, simple biases such as the ones used in the diffusion tests are harder to detect; on the other hand, a cryptographer can easily invert the last linear layer to obtain the difference after the S-boxes of round 4, and observe the biases mentioned above. More specifically, each difference bit of the input to the linear layer is the XOR of 3 bits from the round 5 state. We posit that DBitNet, through its long and short range dependency capabilities, is able to identify the corresponding groups of 3 bits and perform a similar analysis. From the neural distinguisher perspective in the single-key scenario, ARADI behaves similarly to SPECK and SIMON, in that roughly one-third of the full rounds are covered by the neural distinguisher[5].

We note that ARADI is not designed for "...related-key security [as it] is not a significant concern for memory encryption as keys are generated and stored locally" [30], however, the neural distinguisher pipeline only runs one additional round in the related-key scenario. This behavior is similar to that of SIMON32 [38, Tab. 1], where the related-key neural distinguisher covers one more round than the single-key version. This contrasts with, for example, SIMECK [38, Tab. 1] or HIGHT [12], where the related-key neural distinguisher covers three and four additional rounds, respectively.

CLAASP scripts AutoND is integrated in CLAASP and the previous analysis can be obtained using Listing 1.13.

```
from claasp.ciphers.block_ciphers.aradi_block_cipher import
    AradiBlockCipher
from claasp.cipher_modules.neural_network_tests import NeuralNetworkTests
from claasp.cipher_modules.report import Report
cipher = AradiBlockCipher()
test = NeuralNetworkTests(aradi)
report = test.run_autond_pipeline()
```

Listing 1.13. CLAASP script to run the neural distinguisher AutoND pipeline. Runtime: On a A100 GPU the training of the neural distinguisher takes around 40 seconds per epoch with 40 epochs per round. The exact execution time will depend on the number of rounds the neural distinguisher covers, and the number of found ϵ-close differences.

10 Conclusions

In this paper, we presented the applications of the automated cryptanalysis tool CLAASP to analyze the recently proposed low-latency block cipher ARADI. We gave several distinguishers reaching up to 9 out of 16 rounds of ARADI showing the effectiveness of CLAASP in analyzing new symmetric ciphers in a relatively short time (about 3 weeks from the publication of ARADI's technical specifications). We believe the number of rounds of distinguishers could be improved further, and needless to say that key-recovery attacks on more than 9 rounds can be set-up easily using the presented distinguishers. Improving the distinguishers and dedicated key-recovery attacks on ARADI will be our future work.

References

1. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an Efficient Library for Cryptography. https://github.com/relic-toolkit/relic
2. Avanzi, R.: The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. IACR Trans. Symmetric Cryptol. **2017**(1), 4–44 (2017). https://doi.org/10.13154/TOSC.V2017.I1.4-44, https://doi.org/10.13154/tosc.v2017.i1.4-44

[5] AutoND [12, Table 8] obtains neural distinguishers for the following fractions of the total rounds of SPECK, respectively SIMON: SPECK32 - 8 rounds with 51.1% accuracy ($8/22 = 0.36 \approx 1/3$); SPECK128 - 10 rounds with 59.1% accuracy ($10/34 \approx 1/3$); SIMON32- 11 rounds with 51.6% accuracy ($11/32 \approx 1/3$); SIMON128- 20 rounds with 50.7% accuracy ($20/72 \approx 1/3$); ARADI 128 completes at 16 rounds [30, Fig. 3.4]: ARADI 128 - 5 rounds with 59.5% accuracy ($5/16 \approx 1/3$).

3. Avanzi, R., Banik, S., Dunkelman, O., Eichlseder, M., Ghosh, S., Nageler, M., Regazzoni, F.: The qarmav2 family of tweakable block ciphers. IACR Trans. Symmetric Cryptol. **2023**(3), 25–73 (2023). https://doi.org/10.46586/TOSC.V2023.I3.25-73, https://doi.org/10.46586/tosc.v2023.i3.25-73
4. Banik, S., Isobe, T., Liu, F., Minematsu, K., Sakamoto, K.: Orthros: A low-latency PRF. IACR Cryptol. ePrint Arch. p. 390 (2021), https://eprint.iacr.org/2021/390
5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
6. Bassham, L., Soto, J.: NISTIR 6483: Randomness testing of the advanced encryption standard finalist candidates. NIST Internal or Interagency Reports (2000)
7. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch. p. 404 (2013), http://eprint.iacr.org/2013/404
8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016). https://doi.org/10.1007/978-3-662-53008-5_5, https://doi.org/10.1007/978-3-662-53008-5_5
9. Belkheyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: Bipbip: A low-latency tweakable block cipher with small dimensions. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 326–368 (2023). https://doi.org/10.46586/TCHES.V2023.I1.326-368, https://doi.org/10.46586/tches.v2023.i1.326-368
10. Belkheyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: Introducing two low-latency cipher families: Sonic and supersonic. IACR Cryptol. ePrint Arch. p. 878 (2023), https://eprint.iacr.org/2023/878
11. Bellini, E., Gérault, D., Grados, J., Huang, Y.J., Makarim, R.H., Rachidi, M., Tiwari, S.K.: CLAASP: A cryptographic library for the automated analysis of symmetric primitives. In: SAC. Lecture Notes in Computer Science, vol. 14201, pp. 387–408. Springer (2023)
12. Bellini, E., Gerault, D., Hambitzer, A., Rossi, M.: A cipher-agnostic neural training pipeline with automated finding of good input differences. IACR Transactions on Symmetric Cryptology **2023**(3), 184–212 (2023)
13. Bellini, E., Grados, J., Rachidi, M., Satpute, N., Daemen, J., Hirch, S.E.: Ace-hot: Accelerating an extreme amount of symmetric cipher evaluations for (high-order) avalanche tests. In: LATINCRYPT. Lecture Notes in Computer Science, vol. 14168, pp. 24–43. Springer (2023)
14. Bellini, E., Huang, Y.J., Rachidi, M.: Statistical tests for symmetric primitives - an application to NIST lightweight finalists. In: SecITC. Lecture Notes in Computer Science, vol. 13809, pp. 133–152. Springer (2022)
15. Bellini, E., Piccoli, A.D., Formenti, M., Gérault, D., Huynh, P., Pelizzola, S., Polese, S., Visconti, A.: Differential cryptanalysis with sat, smt, milp, and CP: A detailed comparison for bit-oriented primitives. In: CANS. Lecture Notes in Computer Science, vol. 14342, pp. 268–292. Springer (2023)
16. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990). https://doi.org/10.1007/3-540-38424-3_1, https://doi.org/10.1007/3-540-38424-3_1
17. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 208–225. Springer (2012). https://doi.org/10.1007/978-3-642-34961-4_14, https://doi.org/10.1007/978-3-642-34961-4_14
18. Bozilov, D., Eichlseder, M., Knezevic, M., Lambin, B., Leander, G., Moos, T., Nikov, V., Rasoolzadeh, S., Todo, Y., Wiemer, F.: PRINCEv2 - More Security for (almost) No Overhead. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12804, pp. 483–511. Springer (2020). https://doi.org/10.1007/978-3-030-81652-0_19, https://doi.org/10.1007/978-3-030-81652-0_19
19. Brown, R.G.: Dieharder: A Random Number Test Suite Version 3.31.1 (2021), available at https://webhome.phy.duke.edu/~rgb/General/dieharder.php

20. Canale, F., Güneysu, T., Leander, G., Thoma, J.P., Todo, Y., Ueno, R.: SCARF - A low-latency block cipher for secure cache-randomization. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 1937–1954. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/canale
21. Coutinho, M., de Sousa Júnior, R.T., Borges, F.: Continuous Diffusion Analysis. IEEE Access **8**, 123735–123745 (2020)
22. Cui, T., Chen, S., Jia, K., Fu, K., Wang, M.: New automatic search tool for impossible differentials and zero-correlation linear approximations. Cryptology ePrint Archive, Paper 2016/689 (2016), https://eprint.iacr.org/2016/689
23. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Trans. Symmetric Cryptol. **2018**(4), 1–38 (2018). https://doi.org/10.13154/tosc.v2018.i4.1-38, https://doi.org/10.13154/tosc.v2018.i4.1-38
24. Daum, M.: Cryptanalysis of Hash functions of the MD4-family. Ph.D. thesis, Ruhr University Bochum (2005), http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/DaumMagnus/
25. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2 submission to nist. Tech. rep., NIST (May 2021), https://ascon.iaik.tugraz.at/files/asconv12-nist.pdf
26. Elastic: Elasticsearch (2024), https://www.elastic.co/elasticsearch/, version 8.9
27. Elastic: Kibana (2024), https://www.elastic.co/kibana/, version 8.9
28. Gerault, D., Hambitzer, A., Huppert, M., Picek, S.: Sok: 5 years of neural differential cryptanalysis. Cryptology ePrint Archive (2024)
29. Gohr, A.: 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. pp. 150–179. Springer (2019)
30. Greene, P., Motley, M., Weeks, B.: Aradi and llama: Low-latency cryptography for memory encryption. Cryptology ePrint Archive (2024)
31. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 466–495. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_17, https://doi.org/10.1007/978-3-030-45721-1_17
32. Knudsen, L.: Deal-a 128-bit block cipher. complexity **258**(2), 216 (1998)
33. Knudsen, L.R., Wagner, D.A.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2365, pp. 112–127. Springer (2002). https://doi.org/10.1007/3-540-45661-9_9, https://doi.org/10.1007/3-540-45661-9_9
34. Lai, X.: Higher order derivatives and differential cryptanalysis. Communications and Cryptography: Two Sides of One Tapestry pp. 227–233 (1994)
35. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 510–545 (2021). https://doi.org/10.46586/TCHES.V2021.I4.510-545, https://doi.org/10.46586/tches.v2021.i4.510-545
36. LeMay, M., Rakshit, J., Deutsch, S., Durham, D.M., Ghosh, S., Nori, A., Gaur, J., Weiler, A., Sultana, S., Grewal, K., Subramoney, S.: Cryptographic capability computing. In: MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021. pp. 253–267. ACM (2021). https://doi.org/10.1145/3466752.3480076, https://doi.org/10.1145/3466752.3480076
37. Libralesso, L., Delobel, F., Lafourcade, P., Solnon, C.: Automatic Generation of Declarative Models For Differential Cryptanalysis. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
38. Lu, J., Liu, G., Sun, B., Li, C., Liu, L.: Improved (related-key) differential-based neural distinguishers for simon and simeck block ciphers. The Computer Journal **67**(2), 537–547 (2024)
39. Marsaglia, G.: The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness (1995), web archived at https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/
40. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings. Lecture Notes in Computer Science, vol. 658, pp. 81–91. Springer (1992). https://doi.org/10.1007/3-540-47555-9_7, https://doi.org/10.1007/3-540-47555-9_7

41. Nethercote, N., Stuckey, P.J., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming - CP 2007. Principles and Practice of Constraint Programming 2007, Springer (2007)
42. Ranea, A., Rijmen, V.: Characteristic automated search of cryptographic algorithms for distinguishing attacks (CASCADA). IET Inf. Secur. **16**(6), 470–481 (2022)
43. Soto, J.: NISTIR 6390: Randomness testing of the advanced encryption standard candidate algorithms. NIST Internal or Interagency Reports (1999)
44. Soto, J.: Statistical testing of random number generators. In: Proceedings of the 22nd National Information Systems Security Conference (NISSC). pp. 1–12. National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA (1999), https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1999/10/21/proceedings-of-the-22nd-nissc-1999/documents/papers/p24.pdf
45. Sun, L., Gérault, D., Wang, W., Wang, M.: On the usage of deterministic (related-key) truncated differentials and multidimensional linear approximations for SPN ciphers. IACR Trans. Symmetric Cryptol. **2020**(3), 262–287 (2020). https://doi.org/10.13154/TOSC.V2020.I3.262-287, https://doi.org/10.13154/tosc.v2020.i3.262-287
46. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_12, https://doi.org/10.1007/978-3-662-46800-5_12
47. Todo, Y., Morii, M.: Bit-based division property and application to simon family. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 357–377. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_18, https://doi.org/10.1007/978-3-662-52993-5_18

A Avalanche Entropy Details

In this section we report the average values of the avalanche entropy for each output bit, up to round 5, included. From round 6 onwards, the entropy reaches its maximum level, so we do not report it to keep the table more compact.

	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7	bit8	bit9	bit10	bit11	bit12	bit13	bit14	bit15
round1	0.089	0.089	0.089	0.089	0.090	0.089	0.089	0.089	0.090	0.090	0.090	0.089	0.090	0.089	0.089	0.090
round2	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.278	0.279	0.279	0.279	0.279	0.279
round3	0.715	0.715	0.716	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715	0.715
round4	0.998	0.998	0.998	0.998	0.998	0.998	0.998	0.998	0.998	0.999	0.998	0.999	0.998	0.998	0.998	0.999
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit16	bit17	bit18	bit19	bit20	bit21	bit22	bit23	bit24	bit25	bit26	bit27	bit28	bit29	bit30	bit31
round1	0.089	0.090	0.090	0.089	0.089	0.089	0.089	0.090	0.089	0.090	0.089	0.089	0.090	0.090	0.089	0.089
round2	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279
round3	0.685	0.685	0.684	0.684	0.684	0.684	0.685	0.684	0.685	0.684	0.684	0.684	0.684	0.684	0.684	0.684
round4	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit32	bit33	bit34	bit35	bit36	bit37	bit38	bit39	bit40	bit41	bit42	bit43	bit44	bit45	bit46	bit47
round1	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047
round2	0.270	0.268	0.270	0.269	0.269	0.269	0.270	0.268	0.269	0.269	0.270	0.268	0.270	0.269	0.269	0.269
round3	0.713	0.714	0.713	0.714	0.713	0.713	0.713	0.714	0.713	0.714	0.713	0.714	0.713	0.714	0.714	0.713
round4	0.998	0.999	0.998	0.999	0.998	0.998	0.998	0.999	0.998	0.999	0.998	0.998	0.998	0.998	0.998	0.998
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit48	bit49	bit50	bit51	bit52	bit53	bit54	bit55	bit56	bit57	bit58	bit59	bit60	bit61	bit62	bit63
round1	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.047
round2	0.269	0.270	0.268	0.270	0.268	0.269	0.269	0.270	0.268	0.268	0.269	0.269	0.268	0.270	0.268	0.270
round3	0.682	0.683	0.683	0.683	0.683	0.683	0.682	0.682	0.682	0.683	0.682	0.683	0.683	0.683	0.683	0.683
round4	0.998	0.998	0.999	0.998	0.999	0.998	0.998	0.998	0.999	0.999	0.999	0.999	0.999	0.999	0.998	0.999
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit64	bit65	bit66	bit67	bit68	bit69	bit70	bit71	bit72	bit73	bit74	bit75	bit76	bit77	bit78	bit79
round1	0.085	0.085	0.084	0.086	0.085	0.085	0.085	0.084	0.085	0.085	0.084	0.085	0.085	0.084	0.085	0.085
round2	0.279	0.279	0.279	0.278	0.279	0.278	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.278
round3	0.716	0.715	0.716	0.716	0.716	0.716	0.715	0.716	0.715	0.716	0.715	0.716	0.716	0.715	0.716	0.716
round4	0.999	0.998	0.998	0.999	0.999	0.998	0.998	0.999	0.998	0.998	0.998	0.999	0.998	0.999	0.998	0.998
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit80	bit81	bit82	bit83	bit84	bit85	bit86	bit87	bit88	bit89	bit90	bit91	bit92	bit93	bit94	bit95
round1	0.084	0.085	0.085	0.085	0.085	0.083	0.086	0.085	0.085	0.084	0.084	0.085	0.085	0.084	0.086	0.085
round2	0.278	0.279	0.279	0.279	0.278	0.279	0.279	0.278	0.279	0.279	0.279	0.279	0.279	0.279	0.279	0.279
round3	0.685	0.685	0.684	0.684	0.685	0.685	0.684	0.685	0.685	0.684	0.685	0.684	0.685	0.685	0.685	0.684
round4	0.998	0.999	0.999	0.998	0.999	0.999	0.998	0.999	0.999	0.999	0.999	0.999	0.998	0.998	0.999	0.999
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit96	bit97	bit98	bit99	bit100	bit101	bit102	bit103	bit104	bit105	bit106	bit107	bit108	bit109	bit110	bit111
round1	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070
round2	0.276	0.276	0.276	0.275	0.276	0.276	0.275	0.275	0.275	0.275	0.276	0.275	0.275	0.275	0.275	0.275
round3	0.714	0.714	0.714	0.714	0.715	0.714	0.714	0.714	0.715	0.715	0.715	0.714	0.715	0.714	0.715	0.714
round4	0.998	0.999	0.998	0.998	0.999	0.999	0.998	0.998	0.999	0.998	0.999	0.998	0.999	0.999	0.999	0.998
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999

	bit112	bit113	bit114	bit115	bit116	bit117	bit118	bit119	bit120	bit121	bit122	bit123	bit124	bit125	bit126	bit127
round1	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070	0.070
round2	0.275	0.275	0.275	0.275	0.275	0.275	0.276	0.275	0.275	0.276	0.275	0.276	0.275	0.276	0.275	0.276
round3	0.684	0.684	0.684	0.683	0.683	0.683	0.684	0.683	0.684	0.684	0.684	0.683	0.683	0.684	0.684	0.684
round4	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999
round5	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999