ACE-HoT: Accelerating an Extreme Amount of Symmetric Cipher Evaluations for (High-order) Avalanche Tests

Emanuele Bellini[1], Juan Grados[1], Mohamed Rachidi[1], Nitin Satpute[1], Joan Daemen[2], and Solane El Hirch[2]

[1] Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE
{emanuele.bellini, juan.grados, mohamed.rachidi,nitin.satpute}@tii.ae
[2] Radboud University, Netherlands, joan@cs.ru.nl, solane.elhirch@ru.nl

Abstract. In this work, we tackle the problem of estimating the security of iterated symmetric ciphers in an efficient manner, with tests that do not require a deep analysis of the internal structure of the cipher. This is particularly useful during the design phase of these ciphers, especially for quickly testing several combinations of possible parameters defining several cipher design variants.

We consider a popular statistical test that allows us to determine the probability of flipping each cipher output bit, given a small variation in the input of the cipher. From these probabilities, one can compute three measurable metrics related to the well-known full diffusion, avalanche and strict avalanche criteria.

This highly parallelizable testing process scales linearly with the number of samples, i.e., cipher inputs, to be evaluated and the number of design variants to be tested. But, the number of design variants might grow exponentially with respect to some parameters.

The high cost of Central Processing Unit (CPU)s makes them a bad candidate for this kind of parallelization. As a main contribution, we propose a framework, ACE-HoT, to parallelize the testing process using multi-Graphics Processing Units (GPUs). Our implementation does not perform any intermediate CPU-GPU data transfers.

The diffusion and avalanche criteria can be seen as an application of discrete first-order derivatives. As a secondary contribution, we generalize these criteria to their *high-order* version. Our generalization requires an exponentially larger number of samples, in order to compute sufficiently accurate probabilities.

As a case study, we apply ACE-HoT on most of the finalists of the National Institute of Standards and Technologies (NIST) lightweight standardization process, with a special focus on the winner ASCON.

Keywords: GPU · CUDA programming · Avalanche tests · Symmetric ciphers · Statistical tests

1 Introduction

In this work, we describe how to perform a security assessment of encryption and authentication algorithms by means of statistical tests. These tests require a large amount of computations to be executed. We show how to perform these tests on Graphics Processing Units.

1.1 Background and Motivation

The cryptographic community is constantly trying to design more secure and better-performing ciphers. Several public selections took place to determine the best cryptographic primitives for standardization. Some notable examples by the American NIST are the Advanced Encryption Standard selection process [25] started in 1997, the Secure Hash Algorithm of third generation (SHA-3) competition [20] started in 2007, and the NIST lightweight cryptography standardization process [21] started in 2018 and terminated in 2023 with the selection of ASCON, a permutation-based hash and authenticated encryption cipher. Other examples include the eSTREAM competition [22] for stream ciphers, and the CAESAR competition [1] for Authenticated Encryption.

In order for these competitions to evaluate the candidates more fairly, it is important to establish a common framework that allows evaluation of the security of each primitive. One possible approach to establishing the quality of a round function is to define a certain measurable property, observe its variation across the rounds, and then compare it with the computational cost of the round function itself (which depends on the platform).

Avalanche tests A common way of performing this assessment is by measuring some statistical properties observed after evaluating the cipher under scrutiny over samples with certain characteristics. This work focuses on a particular type of statistical test, namely the avalanche tests and on their higher order version introduced in this work. The main challenge in performing high-order avalanche tests is the large number of samples that they require. For example, the most costly high-order avalanche test we perform requires $2^{49.29} \approx 10^{14.83}$ cipher evaluations.

Parallel computing GPUs can perform thousands of computations in parallel depending on the availability of the number of cores on the Streaming Multiprocessors (SMs) [12,18,10]. The computations are distributed on GPUs when the CPU launches an application in the form of a kernel. There are many challenges with regard to the multi-GPU implementation of avalanche tests, especially when these tests have to be executed for an extremely high number of variants of a cipher (during its design phase), or in their high-order version. These challenges are categorized in terms of CPU-bottleneck during iterative kernel calling, inter-block GPU synchronization, which is taken care of by iterative kernel calling, inter-GPU communication during the processing of avalanche tests and memory-based implementation of avalanche tests for random samples.

One of the ways to overcome the above challenges is to effectively parallelize the computations on the GPUs using the Compute Unified Device Architecture (CUDA) programming framework. Typically, CPU acts as a host and launches a device kernel with a required number of computation blocks on the GPU. The GPU schedules the computation blocks for the kernel on the SMs. Each SM can handle one or more computation blocks. The GPU resource manager schedules and allocates resources for each compute block. The blocks communicate and synchronize via a device memory on the GPU. Threads in a block execute in groups called warps and share a common memory in that block. Each warp uses the resources of the SM based on its register memory requirements. The ratio between the number of warps in process and the maximum number of warps defines the occupancy of the GPU [14,4]. It is important to maintain a high occupancy of the GPU to achieve maximum computing performance. In order to efficiently utilize the hardware, it is essential to understand the computation and communication resources. Based on the availability of the CPU-GPU resources, the tasks from an application can be scheduled and allocated efficiently.

1.2 Our contribution

In this work, we provide a framework, ACE-HoT, to perform avalanche tests requiring an extremely high number of cipher evaluations exploiting GPUs. This can be useful during the design phase of a cipher, when a very high number of parameters have to be quickly evaluated against differential properties. We also generalize avalanche tests to a high-order version. This generalization requires a very large amount of cipher executions that can be easily handled by our framework. As a case study, we provide a detailed analysis of our new test on the winner of the NIST lightweight standardization process [21], namely the ASCON permutation [6]. Due to space constraints, we provide a less detailed analysis of the other finalists.

We refer to our new test as *high order avalanche test*. When the order d is known, we say *avalanche test of order d* or *d-order (or d-th order) avalanche test*. The contributions of the paper are presented below in more detail:

1. A framework to perform avalanche tests requiring an extremely high number of cipher evaluations exploiting GPUs.
2. We introduce a new high-order avalanche test for the assessment of a symmetric cipher in the black box scenario, i.e. where no knowledge is assumed of the internal structure of the cipher except its input/output bit size. From a cryptographic point of view, the already known first-order test allows to retrieve information about the applicability of certain attacks, such as impossible differentials [2] and truncated differentials [15]. With our generalization, we have information that might lead to the discovery of higher-order differentials distinguishers [17,16].
3. We provide an accelerated implementation of the avalanche tests of orders 1, 2, 3 and 4 for the ASCON permutation and show that the avalanche criteria (defined under 3 different metrics) are met after 4 rounds with avalanche tests of order 1 and 2 and after 5 rounds for order 3 and 4.

4. To the best of our knowledge, this is the first work towards multi-GPU acceleration for high-order avalanche tests to study the trend of avalanche metrics with respect to the number of samples and rounds. The proposed implementation includes efficient utilization of hardware resources without any intermediate CPU-GPU data transfers and no inter-GPU communications.
5. The 3$^{\text{rd}}$ order avalanche test requires 14 seconds (approx.) for 2,000 samples on 8xTITAN GPUs. The implementation on 4xA100 Ampere GPUs is generally faster compared to the implementation on 8xTITAN GPUs. Notice that in this case, $\binom{320}{3} = 5,410,240 \approx 2^{22.36}$ differences need to be evaluated.
6. The 4$^{\text{th}}$ order avalanche test requires 49.55 seconds and 40.30 minutes for 2,000 and 100,000 samples respectively on 4xA100 Ampere GPUs. Notice that in this case, $\binom{320}{4} = 428,761,520 \approx 2^{28.67}$ differences need to be evaluated, which corresponds to $\approx 2^{46.29}$ 5-round Ascon evaluations in the case of $100,000$ samples per difference. To highlight the potential of the framework, we note that 2^{47} 5-round Ascon evaluations on a single core CPU are estimated to last on average 113 days on an i9 Intel macOS laptop, which shows a significant reduction in timing by using GPUs. Additionally, we verified all the Ascon distinguishers presented by Raghvendra Rohit and Santanu Sarkar [23] in minutes, while for them, it took weeks (especially for 7-round Ascon distinguisher).
7. We release our source code to the community for future research (GitHub: Link Anonymous[3]).

1.3 Related works

In this subsection, we give an insight into some of the works that have been done previously. The notion of avalanche tests applied on ciphers was raised from the ideas of completeness and avalanche effect first introduced by Kam and Davida [11] and Feistel [8], respectively. A cipher is said *complete* (or that it reached *full diffusion*) when each of its output bits depends on all of the input bits. The avalanche effect of a cryptographic algorithm is observed when an average of one half of the output bits change whenever a single input bit is flipped. Webster and Tavares [27] explain how to build what are called perfect 4x4 S-Boxes by using the strict avalanche criterion. Later, Joan Daemen, Seth Hoffert, Gilles Van Assche and Ronny Van Keer [5] report in their paper on the performance of the cipher Xoodoo with respect to these criteria. Avalanche tests can be seen as a special case of statistical tests where the randomness of the output of a cipher is examined. One of the most frequently used test batteries is the NIST Statistical Test Suite [24]. In 2021, Kim and Yeom [13] propose a GPU based parallel implementation of the most time-consuming part of the entropy estimation in these tests and demonstrate that their implementation is about 3 to 25 times faster than that of the NIST package (measured on two different hardware configurations, see reference for details). While in this work we introduce the notion of high order avalanche test for a symmetric cipher, the

[3] we can provide the source code to the reviewers if requested

notion of high order Strict Avalanche Criterion has already been known for a long time in the case of small Boolean functions [9].

1.4 Outline of this work

The remainder of this paper is structured as follows. Section 2 and Section 3 describe high-order avalanche tests and criteria. Section 4 and Section 5 provide framework for multi-GPU acceleration for high-order avalanche tests. Sections 6 to 8 explains the evaluation methodology and presents the detailed experimental results and discussions. Finally, we conclude the paper in section 9.

2 Avalanche tests

We denote by $GF(2)$ the binary field with 2 elements, and with $GF(2)^n$ the n-dimensional vector space over $GF(2)$. Block ciphers are functions with inverse and they are iterated. That is, block ciphers apply an map repeatedly over a series of rounds. In other words, given a set of r maps $F_i : GF(2)^n \times GF(2)^m \to GF(2)^n$ with $i = 0, \ldots, r-1$, that takes as input a n bits block and a m bits subkey, an iterated block cipher F is such that $F = F_{r-1} \circ \ldots \circ F_0$.

Three tests to measure the avalanche properties of a symmetric iterated block cipher are presented in [5]. These tests evaluate the cipher with respect to three different criteria: the *full diffusion*, the *avalanche*, and the *strict avalanche* criteria. The goal of these tests is to measure the quantitative diffusion power of the round function. Note that the common behavior of an iterated cipher is not to meet the criterion for the first rounds and then to meet it for all the remaining ones.

2.1 The avalanche probability vector

The tests are 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 [5] experiments $M = 250,000$ was used. In this work, we observe the behavior of the tests for smaller values of M.

2.2 Avalanche criteria

The APV is used to derive 3 metrics, where $p_i = P_{\Delta F_i}$:

- **Avalanche dependence**: number of output bits that may flip, defined as $D_{av}(F, \Delta) = b - \sum_i \delta(p_i)$, with $\delta(x)$ equal to 1 if $x = 0$ and 0 otherwise. The **full diffusion** criterion is satisfied if $D_{av}(F, \Delta) = b$ for all Δ with Hamming weight 1.

- **Avalanche weight**: expected Hamming weight of the output difference, defined as $W_{av}(F, \Delta) = \sum_i p_i$. Given a certain threshold t, the **avalanche** criterion is satisfied if $b/2 - t \leq W_{av}(F, \Delta) \leq b/2 + t$ for all Δ with Hamming weight 1.
- **Avalanche entropy**: uncertainty about whether output bits flip, defined as an entropy:
  $H_{av}(F, \Delta) = \sum_i (-p_i \log_2(p_i) - (1 - p_i) \log_2(1 - p_i))$. Given a certain threshold t, the **strict avalanche** criterion (SAC) is satisfied if $b - t \leq H_{av}(F, \Delta) \leq b + t$ for all input differences Δ with Hamming weight 1.

The three metrics have values in the range $[0, \ldots, b]$ and for a random transformation F we have that for any input difference Δ then $D_{av}(F, \Delta) \approx b$, $W_{av}(F, \Delta) \approx b/2$ and $H_{av}(F, \Delta) \approx b$. We actually report on the minimum value over all first order input differences.

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 1-bit vectors (sometimes called *unit 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 High-order avalanche tests

In [5], the metrics are computed for all Δ of Hamming weight 1, i.e. for all 1^{st} order input differences of the cipher. This is equivalent to say that the APV is computed for the first order derivative of the n-bit vectorial Boolean function F with respect to the points Δ of Hamming weight 1. Such derivative is defined as $D_\Delta(x) = F(x) + F(x + \Delta)$, with $x, \Delta \in GF(2)^b$ [3]. The same approach can be easily extended to higher order derivatives of F with respect to a vector space $\mathcal{V}$ of length b and dimension d, i.e. $D_\mathcal{V}(x) = \sum_{v \in \mathcal{V}} F(x + v)$. Our technique is somewhat similar to computing higher order derivatives [17], however, the metrics which we evaluate in this work are completely different. For example, a traditional d-order derivative is utilized in integral/cube attacks to check the presence or absence of a superpoly and then later used for recovering key bits. In our case, we use the d-order derivative to generalize the first-order avalanche tests. In what follows, we first describe metrics of high-order avalanche tests and then discuss their computational challenges.

3.1 High-order avalanche probability vector

More precisely, the *avalanche probability vector* $P_{\Delta F}$ of order d of a cryptographic primitive F for a vector space $\mathcal{V}$ of length b and dimension d and generated by a basis of single-bit vectors, is the vector whose i-th component is the probability that bit i of the output of $\sum_{v \in \mathcal{V}} F(x+v)$ equals 1. The high-order APV can be computed following algorithm 1.

3.2 High-order avalanche criteria

The avalanche dependence, weight and entropy are then computed as for the first order and the three criteria are defined as follows.

- The **full diffusion** criterion of order d is satisfied if $D_{av}(F, \mathcal{V}) = b$ for all vector spaces $\mathcal{V}$ of length b and dimension d generated by a basis of single-bit vectors.
- Given a certain threshold t, the **avalanche** criterion of order d is satisfied if $b/2 - t \leq W_{av}(F, \mathcal{V}) \leq b/2 + t$ for all vector spaces $\mathcal{V}$ of length b and dimension d.
- Given a certain threshold t, the **strict avalanche** criterion of order d is satisfied if $b - t \leq H_{av}(F, \mathcal{V}) \leq b + t$.

4 Parallelization strategies and multi-GPU implementation

In this section, we discuss two parallelization strategies that are well-known in machine learning training, namely, *data* and *model*-level parallelization (see, e.g., [7, Section 5.1, 5.2]). We also provide a discussion on how to select the most appropriate strategy with reference to concrete use cases. In particular, for ease of explanation (and because it was never done before), we focus our description on the case of computing high-order avalanche probability vectors and their corresponding criteria. We briefly discuss how the same technique can be also applied in other use cases.

4.1 Determining the workload

Let us now focus on the case of computing high-order avalanche probability vectors and their corresponding criteria.

Recall that b is the bit size of the cipher input and $\mathcal{V}$ is a vector space over $GF(2)$ of dimension d, length b and whose basis is made of d 1-bit vectors. The number of samples needed for each vector space is indicated by M. Also, recall from section 3.2 that in order to compute the metrics avalanche dependence, avalanche weight, and avalanche entropy, we need to compute first the APV for a specific vector space $\mathcal{V}$. Furthermore, to compute an APV, we need to compute the sum B for every random sample (see algorithm 1). Finally, recall that once

we have the metrics, we can compute the full-diffusion, avalanche, and strict avalanche criteria.

From the description above, note that our main workload comes from cipher evaluations $F(\cdot)$ to compute B. Thus, to compute B for an APV of order d, we need $|\mathcal{V}|$ cipher evaluations. In turn, to compute this APV, we need M random samples. Thus, to compute the avalanche metrics for all vector spaces (i.e., $\binom{b}{d}$) in a d order derivative of a cipher, we need a total of $\binom{b}{d} \cdot |V| \cdot M$ cipher evaluations.

4.2 Parallelization techniques overview

To parallelize the workload of the high-order avalanche test, we explore two options: *data-level* and *model-level* parallelization. Essentially, these two techniques differ in how a dataset is distributed to the processing units, or CUDA threads in the case of GPUs. More in detail, in the high-order avalanche test, the two techniques differ as follows:

- *Data-level Parallelism*: Each CUDA thread is assigned a b-bit random sample x and it is responsible of computing $B = \sum_{v \in \mathcal{V}} F(x+v)$ for the *single* sample x. In this scenario, the same vector space is used across multiple threads until the right number of samples is exhausted.
- *Model-level Parallelism*: Each CUDA thread is assigned a vector space and computes $B = \sum_{v \in \mathcal{V}} F(x+v)$ for *all* M samples $x \in GF(2)^b$.

In data-level parallelism, the number of threads depends on the number of samples needed to compute the avalanche criteria. On the other hand, in model-level parallelism, the number of threads only depends on the number of vector spaces.

4.3 Choosing the parallelization technique

Choosing which type of parallelism technique to adopt might not be trivial. In particular, for the case of avalanche tests, it seems natural to distribute the samples over each thread. This might turn out to be a good solution for the case of first-order avalanche tests, since the number of vector spaces (determined by weight 1 differences) is very small, i.e., b. However, for higher dimensions, the number of vector spaces quickly outnumbers the number of samples, and model-level parallelization becomes more useful. Another factor in deciding which technique to exploit is the number of physical cores available in the machine. For example, in our experiments with high-order avalanche tests, we used two types of GPUs: TITAN RTX, with a capability of 18432 CUDA cores, and A100-SXM4, with a capability of 27648 CUDA cores per GPU). If the number of samples is $\approx 10^4$, then not all cores will be used in both cases. Finally, distributing the samples over the processing units, requires some communication cost among the units, to compute the value B of the summation. This is not the case in model-level parallelization, where the generation of the elements of the vector space $\mathcal{V}$, the computation of B, the APVs, and the metrics are all computed in the same thread. Note that all these operations have a relatively low cost for a GPU.

For the case of high-order avalanche tests, and for all the reasons stated in the paragraphs above, we decided to use model-level parallelism instead of data-level parallelism.

4.4 Use cases

Another important use case for applying model-level parallelization is when evaluating first-order avalanche tests for a high number of variants of a cipher. This highly parallelizable testing process scales linearly with the number of samples, i.e., cipher inputs, to be evaluated and the number of design variants to be tested. But, the number of design variants might grow exponentially with respect to some parameters. For example, in ASCON, freeing the 10 rotation offsets in the linear layer gives $63^5 \cdot 62^5 \approx 2^{60}$ possible variants of the cipher.

One third use case for applying model-level parallelization is when evaluating first-order avalanche tests for input differences with Hamming weight greater than one. This would be useful to have a preliminary understanding of the resistance of the cipher against differential cryptanalysis with a low Hamming weight initial difference. Notice that is very common for high probability differential trails to start with low Hamming weight differences. Nevertheless, such a test cannot replace automated differential trail search techniques, which, on the other hand, require quite heavy computations and dedicated modeling of the cipher.

A fourth scenario where model-level parallelization is beneficial is in the evaluation of high-order truncated differentials. Specifically, consider an APV $P_{\Delta F}$ of order d for a cryptographic primitive F. Each entry ρ in $P_{\Delta F}$ represents the probability of a d-order truncated differential, starting with the vectors used to compute $P_{\Delta F}$, and ending at the index indicated by ρ. We will leave the implementation of this scenario as future work.

5 Implementation challenges

In this section, we describe the main practical challenges in implementing model-level parallelization in GPUs, and how we overcame them.

5.1 Avoiding CPU bottleneck

The logical number of threads might be different from the actual number of CUDA cores. We already mentioned that when the number of threads is smaller than the number of cores, there is a poor utilization of the GPU resources. On the other hand, when the number of threads is greater than the number of cores, then the threads are divided into batches and executed one batch at a time. This iteration is concretized through an iterative call of the kernel from the CPU. The CPU assigns workloads to the GPU through iterative calls, which is time-consuming due to CPU-GPU communication. To optimize performance, we

used the so-called *grid-stride loop* technique to execute the iterative task solely on the GPU without CPU intervention, enhancing performance [19].

In CUDA programming, a *grid-stride loop* is a common technique used to efficiently parallelize certain operations on NVIDIA GPUs. It involves breaking down a large data set or computational task into smaller chunks and assigning each chunk to a different thread block. The threads within each block then process elements of the data in a loop.

The term grid in CUDA refers to the collection of thread computational blocks that are launched to execute a kernel function on the GPU. Stride in a *grid-stride loop* refers to the distance between consecutive elements that each thread processes. It allows multiple threads to work on non-contiguous elements in memory simultaneously. By using a stride, threads can efficiently load and process data, minimizing memory access conflicts and improving memory coalescing. Here's a brief overview of how a *grid-stride loop* is typically implemented:

- Determine the grid and block dimensions: The data or task is divided into a grid of thread blocks. The grid and block dimensions are chosen based on the problem's requirements and the available GPU resources.
- *Calculate the global thread index*: Each thread is assigned a unique global index that represents its position within the entire grid of thread blocks.
- *Calculate the stride*: Stride is often computed as the total number of threads in the grid multiplied by the number of elements each thread should process.
- *Perform the grid-stride loop*: Each thread enters a loop and processes the data or computation assigned to it based on its global thread index and the calculated stride. The loop continues until all elements have been processed.

Grid-stride loop is particularly useful when the data or task involves irregular memory access patterns or when the data set is too large for a single thread block to handle. *Grid-stride loops* help achieve better performance by maximizing parallelism and minimizing memory access conflicts. In particular, we use grid-stride loop in high-order test to efficiently compute B (see algorithm 1), APVs, and metrics for multiple vector spaces. Instead of assigning a single CUDA thread to handle the computations associated with a single vector space, we map a thread to a group of vector spaces in a stride way.

5.2 Synchronization and communication

In the context of parallel programming, synchronization refers to the coordination of parallel resources to avoid race conditions or to ensure the correct order of operations, and communication refers to the transfer of data between parallel resources. For example, moving data between different memory spaces, such as transferring data from the host memory to the GPU memory or between different GPUs. In our implementation, we basically have the following parallel resources, CUDA threads, CUDA blocks, GPUs, and CPUs.

As mentioned in section 2, a cipher is considered to meet the criteria if the values of each metric, namely avalanche dependence, avalanche weight, and

avalanche entropy, are b, $b/2$, and b, respectively, for all vector spaces $\mathcal{V}$ of dimension d and length b. This means that the cipher meets the criteria if the minimum values (worst-case scenario) for each metric are approximately b, $b/2$, and b, respectively. Thus, besides the computations of the APVs and metrics (as stated in section 4.2), we compute the minimum value of each metric by using an atomic operation. This minimum value is computed in each GPU and communicated to the CPU to perform the final computation to obtain the criteria.

By utilizing model-level parallelism, communication among the parallel resources can be significantly reduced. As mentioned in section 4.2, we use model-level parallelism to assign each thread a unique vector space and compute the corresponding metrics independently. The only communication required is limited to 1) the initial transfer of avalanche test configurations from the CPU to the GPU, and 2) the transmission of the minimum metric values from the GPUs to the CPUs to compute the avalanche criteria.

Two methods can be used to achieve inter-block synchronization in the GPU: atomic operations or transferring control to the CPU. In our approach, we use the former method to compute the minimum value of each metric and check if a cipher meets the criteria. We do not require inter-GPU synchronization since we send the minimum value computed from each GPU to the CPU. However, synchronization at the CPU is necessary to determine the resultant minimum value after receiving the minimum values from corresponding GPU.

5.3 Distributing the workload

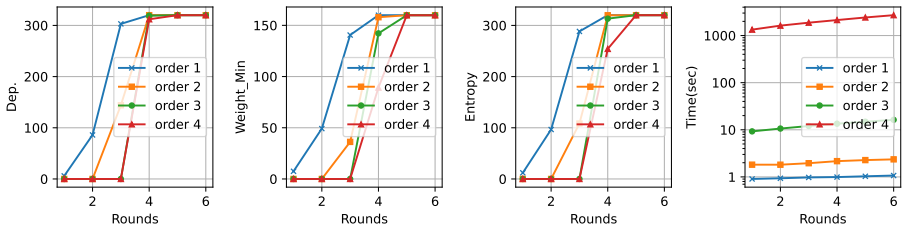
Below we describe the tasks of high-order avalanche tests on the CPU. Specifically, we describe the tasks to distribute the main workload. The proposed implementation takes the following inputs: the cipher, the order d of the test (e.g., 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} order), the number of samples M, the number of rounds of the cipher, and the number of GPUs. The number of computational threads and blocks are evaluated dynamically at runtime, and the workload is assigned to each GPU by considering that we need to distribute $|\mathcal{V}| \cdot \binom{n}{d} \cdot M$ cipher evaluations among these resources.

A high-order avalanche test kernel is launched with the number of parallel threads and blocks on each GPU. To compute the number of parallel threads and blocks, we use the CUDA API. Specifically, we use the function `cudaOccupancyMaxPotentialBlockSize`, which returns the grid and block size that achieves maximum potential occupancy for a device function. In our case, this device function is the kernel high-order test.

As previously mentioned, a cipher performs well with respect to the high-order avalanche tests if the following scores are satisfied for each vector space: the avalanche dependence must be $\approx b$, the weight must be $\approx b/2$, and the entropy must be $\approx b$. Otherwise, the respective criterion is not satisfied. CPU evaluates the resultant worst-case values of the avalanche metrics received from all the GPUs.

The device kernel executes the following steps: 1) it generates vectors $v \in \mathcal{V}$, 2) generates a seed, 3) uses the seed to generate random samples for evaluat-

ing avalanche metrics, 4) the cipher is evaluated on the random samples and vectors v to compute, B, the APV and the metrics, 5) the minimum values of avalanche metrics amongst threads on a GPU are obtained using atomic operation in CUDA. These minimum values from each GPU are communicated to the CPU.



The avalanche criteria are satisfied when Dep. ≈ 320, Weight_Min ≈ 160 and Entropy ≈ 320.

Fig. 1: High-order Avalanche Test for ASCON on 4xA100 Ampere GPUs for 100,000 samples. The above metrics are computed from the vector space that gives the worst values of the corresponding criterion.

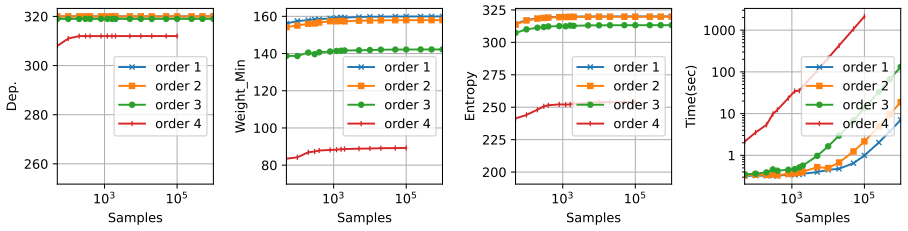


Fig. 2: High-order Avalanche Test for ASCON 4 rounds on 4xA100 Ampere GPUs

Note: 4^{th} order test is not included for more than 100,000 samples due to linearly increasing time dependency for evaluating all differences.

6 Detailed results in the ASCON use case

Due to the page limit and because it is the winner of the NIST lightweight standardization process, we have decided to perform a detailed report on the

performance of ASCON with respect to each criterion and for each different number of rounds. We will summarize the results of other primitives in the next section 7.

Specifically, this section presents a report for up to the 4^{th} order and for different rounds. There are two aspects to analyze the results: 1) evaluate whether ASCON performs well or not the tests along the rounds and 2) timings and speedup of the experiments with respect to the number of rounds and the order of the tests. We evaluated our approaches on the following platforms: 4xA100 GPUs and 8xTITAN GPUs. In section 6.1, we present experiments we did for aspect 1) and in section 6.2, we present experiments we did for aspect 2). Also, here, we make a comparison of our implementations with respect to the aforementioned platforms (4xA100 GPUs and 8xTITAN).

6.1 High-Order Avalanche Tests on ASCON

We evaluate three avalanche metrics: avalanche dependence, weight, and entropy. In order to perform well with respect to the criteria associated with these metrics, ASCON must satisfy the following scores for their respective criterion:

1. avalanche dependence must be equal to 320
2. avalanche weight must be approximately[4] 160 and
3. avalanche entropy must be approximately 320.

We evaluate these tests for the variations in the number of rounds when the sample size is constant. Secondly, we assess the avalanche metrics for the variations in the number of samples for a given number of rounds.

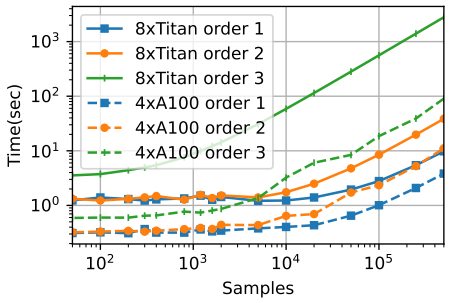


Fig. 3: Performance Comparison of Avalanche Tests for ASCON - on 4xA100 and 8xTITAN RTX GPUs.

[4] A certain threshold has to be fixed.

Avalanche properties versus the number of rounds We evaluate three avalanche tests with respect to the number of rounds and the sample sizes. We show the results corresponding to 100,000 samples in fig. 1. We observe the following in terms of avalanche tests:

1. **Full diffusion criterion:** For all orders, 1^{st}, 2^{nd}, 3^{rd} and 4^{th} order, this criterion needs 100 samples before saturation (when the values of the metric increase with very slow rate). Regarding the numbers of rounds, 1^{st} and 2^{nd} order need 4 rounds to meet the criterion, while 3^{rd} and 4^{th} order require five rounds.
2. **Avalanche criterion:** Similar behaviour is observed here. For all orders, 800 samples are needed to reach 158 value and 2,000 samples to reach 159 value and then saturates. Regarding the numbers of rounds, 1^{st} and 2^{nd} need four rounds, and 3^{rd} and 4^{th} require five rounds.
3. **Strict Avalanche criterion:** Here again, for all orders, 400 samples are enough to reach 319 value and then saturates. Similarly to both previous cases, 1^{st} and 2^{nd} need four rounds, and 3^{rd} and 4^{th} require five rounds.

For each criterion, the 3^{rd} and 4^{th} order tests require at least 5 rounds to reach it. An important point to note is that our framework can also spot the state bits for which the criteria are not satisfied for 4-round Ascon.

Avalanche Metrics versus The Number of Samples In what follows, we observe the evolution of the values of the metrics with respect to the number of samples. We evaluate the three avalanche metrics for 4 rounds of Ascon with different number of samples as it is shown in fig. 2^5. We have not included 4^{th} order avalanche test for more than 100,000 samples (equivalent to $\approx 2^{46.29}$ 5-rounds evaluations) due to linearly increasing time dependency. We observe from fig. 2 that the values of the criteria are almost similar whether using 10^3 or 10^5 samples. To be more specific, fig. 6 shows how accurate the values of the criteria are according to the chosen number of samples.

6.2 Time and Speedup of 4xA100 w.r.t. 8xTITAN GPUs

We measure evaluation time on 4xA100 GPUs and 8xTITAN GPUs and observe the following:

1. **Execution time** for the 3^{rd} order avalanche test requires 14 seconds for 2,000 samples on 8xTITAN GPUs. However, the implementation on 4xA100 GPUs is generally faster in comparison to the 8xTITAN GPUs as shown in fig. 3. The number of multiprocessors per GPU is more in A100 (i.e. 108) in comparison to the TITAN RTX (i.e. 72). Ampere A100 have more cores (6912) and higher FLOPS (single and double) in comparison to the TITAN RTX (4608 cores).

[5] We recall that we report on the minimum value of each metric. This is why all metrics are monotonically non-decreasing.

2. **Speedup** The performance of 4xA100 GPUs is generally faster in comparison to the 8xTITAN GPUs as shown in fig. 3. The range of speedup varies depending upon the order of the tests.

6.3 Note on the inverse of ASCON

We have seen that in the 1st order case, the 3 criteria are reached after 4 rounds of ASCON. The inverse of ASCON reaches the 3 criteria even faster, as we can see in fig. 4. Each cell of this figure is green if the probability of flipping of the underlying bit is close to $\frac{1}{2}$ with a 0.01 bias due to a single input bit difference, red otherwise. We can see that after the 2nd round, all cells are green, meaning that at this round, the weight criterion is satisfied.

This behavior is common on ciphers with a linear layer whose inverse is more dense than its forward operation. For instance, it is well known that one bit difference in the input of the inverse of Ascon linear layer affects at least 31 output bits, while it affects only 3 output bit for the forward linear layer. In most cases, the inverse of a linear layer is chosen to be complex and therefore allows a better diffusion, which prevent successful backward extension of linear or differential trails for example.

Fig. 4: Weight criterion for the inverse of ASCON. Input bit difference injected in position 0.

7 Results for the most popular ciphers

In this section, we decided to focus our attention on the permutations of the finalists of the NIST lightweight standardization process, plus some of the most famous block ciphers and cipher underlying permutations. More specifically, we focused on the following primitives shown in table 1:

Primitives	plaintext size	key size	rounds
Ascon-p	320	-	12
Xoodoo	384	-	12
Gift	128	128	40
Keccak-f[400]	400	-	20
Photon	256	-	12
Skinny	128	384	40
Speck	128	128	32
AES	128	128	10
Chacha	512	-	20
DES	64	64	16
Present	64	80	31

Table 1: Selected primitives for our analysis and their respective plaintext and key bit sizes.

From our experiments, we observed that the selected primitives had a similar behavior than Ascon, that is satisfying all the 3 criteria for the 3^{rd} and 4^{th} orders one round after they get satisfied for the 1^{st} and 2^{nd} orders.

Due to the page limit, we could not display the comparison between all the previous ciphers for each of the 3 criteria. We had to choose one and we believe that the most important criterion is the strict avalanche criterion based on the avalanche entropy. The comparison is shown in fig. 5, where we display the percentage of the number of rounds needed to reach the strict avalanche criterion over the total number of rounds. We show this value for 1st, 2nd, 3rd, and 4th orders. We notice that a more interesting comparison, rather than the number of rounds, would take into account the number of gates in the implementation on a specific platform. On the other hand, this metric is not easy to measure, and we leave it for future work.

8 Discussion

In this paper, we propose and evaluate high-order avalanche tests on multi-GPU platforms. We apply our new test to the selected candidate of the NIST lightweight standardization process, namely the ASCON permutation, a round-based cipher. We conclude the following in terms of avalanche metrics and execution time from the experimental analysis:

1. **Full diffusion criterion** needs 100 samples enough for full diffusion (see section 6.1).
2. **Avalanche criterion** needs 800 samples (see section 6.1) to reach 158 (around 160) value and 2,000 samples to reach 159 (around 160) value and then saturates (increases with very slow rate).
3. **Strict Avalanche criterion** needs 400 samples (see section 6.1) to reach 319 (around 320) value and then saturates.
4. **Execution time** for the 3^{rd} order avalanche test requires 14 sec for 2,000 samples on 8xTITAN GPUs.
5. Following analysis can be made in order to satisfy all avalanche metrics:

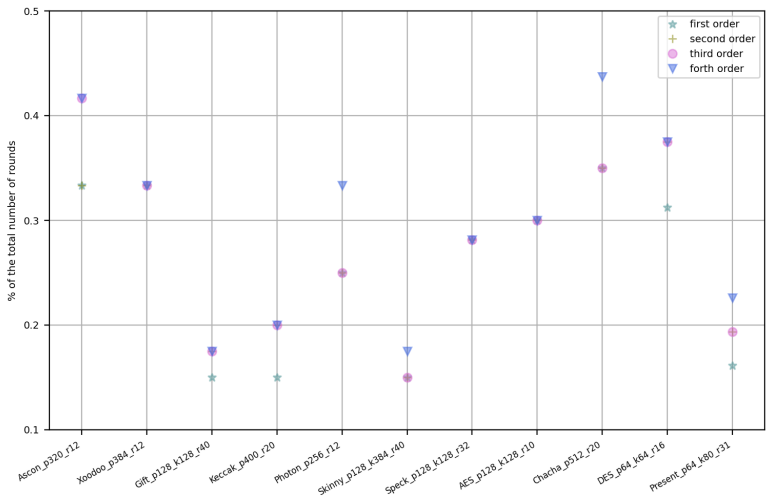


Fig. 5: Avalanche entropy comparison: percentage of number of rounds needed to reach the entropy criterion over the total number of rounds, for 1st, 2nd, 3rd, and 4th order.

(a) 1^{st} and 2^{nd} order avalanche tests require at least 4 rounds to satisfy all criteria.
(b) 3^{rd} and 4^{th} order avalanche tests require at least 5 rounds to satisfy all criteria.

6. **Generalization of the number of threads required for performing n-th order tests in parallel:** Our initial implementation requires memory for seeds corresponding to each thread in order to generate the number of random samples for each thread. The memory requirements increase with the increase in the number of threads corresponding to each combination difference. The memory dependency with respect to the number of computational threads is detailed as follow: i-th order test requires at least $\binom{320}{i} \cdot 48$ KB memory and $\binom{320}{i}$ threads.
However, the above memory dependency has been successfully removed in our memory-less implementation for high-order avalanche tests.
7. **Statistical Analysis of the Results** We use a confidence level of 99% and a population of 2^{320} input samples (for ASCON) as shown in fig. 6. It can be seen that already from samples=100, for 1^{st} order difference, 3 rounds is outside the confidence interval. This cannot be observed only for 10 samples. However, we will not worry too much as these are really very few samples. The result can be interpreted as follows: given
(a) a Confidence Level of 99%

Confidence Interval	Sample size needed
0.1	1664100
0.91216	20,000
1	16641
2.8845	2,000
3.724	1,200
10	166
12.9	100
40	10

Fig. 6: Confidence Interval for the Sample Sizes for Confidence Level 99% on population 2^{320}

Samples	Rounds	Weight_Min	Weight_Max	Confidence Interval
10	3	131.1	160.0	40
10	4	150.2001	160.0	40
100	3	137.63	160.0	12.9
100	4	157.49	161.9	12.9
1,200	3	139.7333	160.0	3.724
1,200	4	159.3608	160.7158	3.724
2,000	3	139.8149	160.0	2.8845
2,000	4	159.4271	160.6405	2.8845
20,000	3	140.4578	160.0	0.91216
20,000	4	159.8024	160.1752	0.91216

Fig. 7: Analysis of Confidence Intervals for Various Sample Sizes on 1^{st} order Avalanche Tests

(b) a Confidence Interval C (a positive integer). Note that C is a function of the sample size, population size and the confidence level.
(c) an input difference D of 1^{st} order,

then 99% of all possible plaintexts P will generate a pair $(P, P + D)$ for which the Avalanche Weight will fall inside the confidence interval of $[160 - C, 160 + C]$. For example, for a sample size of 20,000, we have $C = 0.91216$. We see that at round 4, the avalanche weight is always inside the interval $[160 - 0.91216, 160 + 0.91216]$, so we can consider the test passed at round 4 as shown in fig. 7.

8. **Scalability:** The model level implementation supports scalability in terms of (the large) number of vector spaces computed per GPU. As the number of GPUs increases, the total number of vector spaces computed per GPU decreases and hence decreasing the total evaluation time. It will provide flexibility to process high-order avalanche tests (where order ≥ 4) in lesser time. Our implementation performs reasonably well for up to 4-th order tests. Of course, adding or removing CUDA cores would improve or diminish the performances in a trivial way. The main challenge, which for now remains an open problem, is to perform 5-th order tests. If one would want to implement the same tests over CPUs, the main issue he would face is the cost of purchasing several cores. For comparison, we used around 100,000 CUDA cores. The CPUs are efficient when the total number of vector spaces to be processed is reasonable. The CPUs are limited by the number of cores and the number of operations drastically increases when the order of tests increases. CPUs under-performs already for 3-th order avalanche tests.

9 Conclusion

In this paper, we propose and evaluate high-order avalanche tests on multi-GPU platforms through our ACE-HoT framework. This framework is general and can be easily adapted to test other block ciphers and permutations on several GPU models.

We provided a detailed analysis of the permutation of the ASCON cipher and a comparison of how different ciphers perform under this test. We showed

that for some ciphers (e.g. ASCON) the avalanche criteria are reached at a later round when considering higher orders than 1.

The main challenge of this test was the huge number of cipher evaluations that need to be performed, especially for order 4 tests. We leave for future work to optimize the code and reach even higher orders and to study how the newly found biases could be exploited as a base to mount new or improve existing attacks.

Another open challenge for future work is to explore the use of this generic framework for other applications of cryptanalysis like the work that has been done for the first collision of full SHA-1 [26].

References

1. Authors, V.: CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `https://competitions.cr.yp.to/caesar.html` (2014), accessed: 2022-04-14
2. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18. pp. 12–23. Springer (1999)
3. Carlet, C., Crama, Y., Hammer, P.L.: Vectorial boolean functions for cryptography. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 398–470. Cambridge University Press (2010). https://doi.org/10.1017/cbo9780511780448.012, `https://doi.org/10.1017/cbo9780511780448.012`
4. Corp., N.: Dynamic parallelism in CUDA (2012), `https://goo.gl/KhEhve`
5. 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`
6. 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`
7. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. J. Mach. Learn. Res. **23**, 120:1–120:39 (2022)
8. Feistel, H.: Cryptography and computer privacy. Scientific american **228**(5), 15–23 (1973)
9. Forré, R.: The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition. In: Goldwasser, S. (ed.) Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings. Lecture Notes in Computer Science, vol. 403, pp. 450–468. Springer (1988). https://doi.org/10.1007/0-387-34799-2_31, `https://doi.org/10.1007/0-387-34799-2_31`
10. Hetherington, T.H., Lubeznov, M., Shah, D., Aamodt, T.M.: Edge: Event-driven gpu execution. In: 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 337–353. IEEE Computer Society, Los Alamitos, CA, USA (Sep 2019). https://doi.org/10.1109/PACT.2019.00034

11. Kam, J.B., Davida, G.I.: Structured design of substitution-permutation encryption networks. IEEE Transactions on Computers **28**(10), 747–753 (1979)
12. Kim, J., Hur, S., Lee, E., Lee, S., Kim, J.: Nlp-fast: A fast, scalable, and flexible system to accelerate large-scale heterogeneous nlp models. In: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 75–89. IEEE Computer Society, Los Alamitos, CA, USA (Sep 2021). https://doi.org/10.1109/PACT52795.2021.00013
13. Kim, Y., Yeom, Y.: Accelerated implementation for testing IID assumption of NIST SP 800-90B using GPU. PeerJ Computer Science **7**, e404 (2021)
14. Kirk, D.B., Hwu, W.m.W.: Programming Massively Parallel Processors, Third Edition: A Hands-on Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2016)
15. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994). https://doi.org/10.1007/3-540-60590-8_16, https://doi.org/10.1007/3-540-60590-8_16
16. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption. pp. 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
17. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and cryptography, pp. 227–233. Springer (1994)
18. Liu, G., Wang, S., Bao, Y.: Seer: A time prediction model for cnns from gpu kernel's view. In: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 173–185. IEEE Computer Society, Los Alamitos, CA, USA (Sep 2021). https://doi.org/10.1109/PACT52795.2021.00020
19. Mark, H.: CUDA Pro Tip: Write Flexible Kernels with Grid-Stride Loops (2013)
20. NIST: SHA-3 Competition. https://csrc.nist.gov/projects/hash-functions/sha-3-project (2007), accessed: 2022-04-14
21. NIST: Lightweight Cryptography Standardization Process. https://csrc.nist.gov/Projects/lightweight-cryptography (2018), accessed: 2022-04-14
22. Preneel, B., Robshaw, M., Johansson, T., Bosselaers, A.: eSTREAM: the ECRYPT Stream Cipher Project. https://www.ecrypt.eu.org/stream/ (2005), accessed: 2022-04-14
23. Rohit, R., Sarkar, S.: Diving deep into the weak keys of round reduced Ascon. IACR Transactions on Symmetric Cryptology pp. 74–99 (2021)
24. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. rep., Booz-allen and hamilton inc mclean va (2001)
25. Smid, M., Foti, J.: Development of the advanced encryption standard (2021-08-16 04:08:00 2021). https://doi.org/https://doi.org/10.6028/jres.126.024, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=931014
26. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The First Collision for Full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 570–596. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_19, https://doi.org/10.1007/978-3-319-63688-7_19
27. Webster, A., Tavares, S.E.: On the design of s-boxes. In: Conference on the theory and application of cryptographic techniques. pp. 523–534. Springer, Linz, Austria (1985)