Simple Vs Vectorial: Exploiting Structural Symmetry to Beat the ZeroSum Distinguisher

Applications to SHA3, Xoodyak and Bash

Sahiba Suryawanshi[1*], Shibam Ghosh[2], Dhiman Saha[1], Prathamesh Ram[1]

[1]de.ci.phe.red Lab, Department of Computer Science and Engineering, Indian Institute of Technology Bhilai, Chhattisgarh, 491001, India.
[2]Department of Computer Science, University of Haifa, Haifa, Israel.

*Corresponding author(s). E-mail(s): sahibas@iitbhilai.ac.in;
Contributing authors: sghosh03@campus.haifa.ac.il; dhiman@iitbhilai.ac.in; rprathamesh@iitbhilai.ac.in;

Abstract

Higher order differential properties constitute a very insightful tool at the hands of a cryptanalyst allowing for probing a cryptographic primitive from an algebraic perspective. In FSE 2017, Saha *et al.* reported SymSum (referred to as $\mathsf{SymSum}_{\mathsf{Vec}}$ in this paper), a new distinguisher based on higher order *vectorial* Boolean derivatives of SHA-3, constituting one of the best distinguishers on the latest cryptographic hash standard. $\mathsf{SymSum}_{\mathsf{Vec}}$ exploits the difference in the algebraic degree of highest degree monomials in the algebraic normal form of SHA-3 with regards to their dependence on round constants. Later in Africacrypt 2020, Suryawanshi *et al.* extended $\mathsf{SymSum}_{\mathsf{Vec}}$ using linearization techniques and in SSS 2023 also applied it to NIST-LWC finalist Xoodyak. However, a major limitation of $\mathsf{SymSum}_{\mathsf{Vec}}$ is the maximum attainable derivative (MAD) which is *less than half* of the widely studied ZeroSum distinguisher. This is attributed to $\mathsf{SymSum}_{\mathsf{Vec}}$ being dependent on $\boldsymbol{m}-$fold *vectorial* derivatives while ZeroSum relies on $\boldsymbol{m}-$fold *simple* derivatives. In this work we overcome this limitation of $\mathsf{SymSum}_{\mathsf{Vec}}$ by developing and validating the theory of computing $\mathsf{SymSum}_{\mathsf{Vec}}$ with simple derivatives. This gives us a close to **100%** improvement in the MAD that can be computed. The new distinguisher reported in this work can also be combined with one/two-round linearization to penetrate more rounds. Moreover, we identify an issue with the two-round linearization claim made by Suryawanshi *et al.* which renders it invalid and also furnish an algebraic fix at the cost of some additional constraints.

Combining all results we report $\mathsf{SymSum_{Sim}}$, a new variant of the $\mathsf{SymSum_{Vec}}$ distinguisher based on $\boldsymbol{m}-$fold *simple* derivatives that outperforms ZeroSum by a factor of $\mathbf{2^{257}, 2^{129}}$ for 10-round SHA-3-384 and 9-round SHA-3-512 respectively while enjoying the same MAD as ZeroSum. For every other SHA-3 variant, $\mathsf{SymSum_{Sim}}$ maintains an advantage of factor 2. Combined with one/two-round linearization, $\mathsf{SymSum_{Sim}}$ improves upon all existing ZeroSum and $\mathsf{SymSum_{Vec}}$ distinguishers on both SHA-3 and Xoodyak. As regards Keccak-$\boldsymbol{p}$, the internal permutation of SHA-3, we report the best 15-round distinguisher with a complexity of $\mathbf{2^{256}}$ and the first better than birthday-bound 16-round distinguisher with a complexity of $\mathbf{2^{512}}$ (improving upon the 15/16-round results by Guo *et al.* in Asiacrypt 2016). We also devise the best *full-round* distinguisher on the Xoodoo internal permutation of Xoodyak with a *practically* verifiable complexity of $\mathbf{2^{32}}$ and furnish the first third-party distinguishers on the Belarushian hash function Bash. All distinguishers furnished in this work have been verified through implementations whenever practically viable. Overall, with the MAD barrier broken, $\mathsf{SymSum_{Sim}}$ emerges as a better distinguisher than ZeroSum on all fronts and adds to the state-of-the-art of cryptanalytic tools investigating non-randomness of crypto primitives.

Keywords: Boolean Derivative, SymSum Distinguisher, Hash Function, SHA-3, Xoodyak, Bash.

1 Introduction

Differential Cryptanalysis represents one of the earliest systematic and mathematically sound efforts to analyze the strength of symmetric crypto primitives. Since its introduction by Biham and Shamir to analyze DES [BS91], it has evolved to become a defacto cryptanalytic litmus test for modern ciphers. The link between the classical differential attack and first order Boolean derivative is well established. This was formalized by Ming *et al.* [DYS+14] and further used to connect the idea of higher order differential cryptanalysis with computation of higher order Boolean derivatives. The cryptanalytic tool-set had gained from this connection in various interesting ways. The Integral attack [GS20], Cube Attack [LBDW17, SG18, BDL+19, ZCWW21], Division property [YLW+19], ZeroSum [AM09a] and $\mathsf{SymSum_{Vec}}$[*] [SKC17] distinguishers all belong to the family of higher order differential cryptanalysis and hence in-turn are connected to computing higher order Boolean derivatives. These attacks primarily try to exploit the algebraic structure of the cipher by trying to influence the highest degree monomials (for instance finding superpoly in Cube attack [WHG+19], identifying input division set for the bit based division property [GD21]). The implication is in the form of a "Zero-Sum" which manifests as balancedness in almost all the above attacks except $\mathsf{SymSum_{Vec}}$. The $\mathsf{SymSum_{Vec}}$ distinguisher introduced in IACR ToSC 2017 is a one-of-a-kind attack on SPN constructions that leverages the difference in algebraic degree of highest degree monomials that are dependent on round-constants from the ones that are independent of them. It builds upon the fact that in certain

[*]Reported originally as SymSum by Saha *et al.* in FSE 2017

classes of SPN, the round-constant addition follows a non-linear operation like an Sbox and hence never affects the highest degree monomials. The difference in the degree of monomials affected and unaffected by round-constants is dictated by the degree (say η) of the non-linear operation. The property can be exploited by computing $(d-\eta+1)^{th}$ order *vectorial* derivative of a $d-$degree function. In case of SHA-3, where $\eta = 2$, computing $(d-1)^{th}$ order vectorial derivative over special linear subspaces manifests as a output-sum which is *deterministically* symmetric. It can be noted that the ZeroSum property requires computation of $(d+1)^{th}$ order derivative to realize the distinguisher. The $\mathsf{SymSum_{Vec}}$ distinguishers were extended in Africacrypt 2020 [SSS20] capitalizing on the idea of linear structures introduced by Guo *et al.* in Asiacrypt 2016 [GLS16]. In SSS 2023 [SS23], the idea was further generalized to newer classes of SPN with new distinguishers reported on NIST-LWC finalist Xoodyak and its internal permutation Xoodoo.

$\mathsf{SymSum_{Vec}}$ and ZeroSum are both based on higher order Boolean derivatives and are close competitors. While raw $\mathsf{SymSum_{Vec}}$ enjoys a factor 4 improvement ($(d-1)^{th}$ vs $(d+1)^{th}$ order derivative for Keccak) in complexity over ZeroSum, Suryawanshi *et al.* demonstrated that linearization reduces the advantage to a factor of 2 (d^{th} vs $(d+1)^{th}$). One crucial aspect that separates $\mathsf{SymSum_{Vec}}$ and ZeroSum is the nature of the Boolean derivative. ZeroSum relies on simple derivatives while $\mathsf{SymSum_{Vec}}$ leverages the vectorial counterpart. The primary difference is in the order of the maximum attainable derivative (MAD) that can be computed. For an n-variate function this amounts to computing $n-$fold derivative implying MAD $= n$. For ZeroSum we have MAD $= n$, while for $\mathsf{SymSum_{Vec}}$ as defined by Saha *et al.*, MAD $\leq \frac{n}{2}$ and this is attributed to the partition-size $= 2$ of each vectorial derivative used in $\mathsf{SymSum_{Vec}}$. So for an $m-$fold vectorial derivative, $\mathsf{SymSum_{Vec}}$ consumes $2m$ variables. *This constitutes the biggest limitation of $\mathsf{SymSum_{Vec}}$ despite its factor 4 advantage over ZeroSum as it reduces the MAD twice as fast.* The limitation is amplified when we augment $\mathsf{SymSum_{Vec}}$ with linearization since that imposes additional constraints on the system. Here too $\mathsf{SymSum_{Vec}}$ needs twice as many constraints as compared to ZeroSum. Owing to this $\mathsf{SymSum_{Vec}}$ fails to penetrate higher number of rounds, a limitation that constitutes the primary motivation behind the current work.

Our Contribution

$\mathsf{SymSum_{Vec}}$ with simple order derivatives. The current work overcomes all the above limitations of $\mathsf{SymSum_{Vec}}$. We propose a new distinguisher called $\mathsf{SymSum_{Sim}}$ which enjoys almost all benefits of $\mathsf{SymSum_{Vec}}$ but uses *simple* Boolean derivatives instead of vectorial ones. This brings $\mathsf{SymSum_{Sim}}$ at par with ZeroSum in terms of MAD allowing to penetrate rounds previously *unreachable* by $\mathsf{SymSum_{Vec}}$. To understand this, one needs to acknowledge that the $\mathsf{SymSum_{Vec}}$ property proposed by Saha *et al.* has two aspects. The first aspect is the degree difference that emerges due to the ordering of operations in the SPN round function. Secondly, the usage of vectorial derivatives with *fully* self-symmetric[†] inputs to exploit the degree difference resulting in a deterministically-symmetric output-sum for SHA-3. In this work, we target the

[†]The idea of symmetry is to make two-halves of a SHA-3 internal state equal in the $z-$axis.

second aspect of Saha *et al.*'s work. We argue that the requirement of a *fully* self-symmetric state is an *overkill.* We prove that the necessary and sufficient condition is to keep the state symmetric, *barring the variables across which the derivative is computed.* We call the variables with respect to which the derivative is computed as *independent* variables while the remaining variables are referred to as *fixed* variables. The main line of argument stems from the fact that for an $n-$variate function, the $m-$fold simple derivative is essentially a function that does not depend on the m variables across which the derivatives have been taken. From the perspective of $\mathsf{SymSum}_{\mathsf{Vec}}$, this implies that these m variables *need not* maintain a symmetric relation in *values* but *only* in *position.* This is the crux of the new distinguisher $\mathsf{SymSum}_{\mathsf{Sim}}$ proposed in this work. Since symmetry is not required for the m variables we can compute simple derivatives instead of vectorial ones. Hence, we are constrained to select $m/2$ variables in one half of the state and remaining $m/2$ variables in the corresponding symmetric positions. This adds the restriction that $2|m$, and hence the idea of $\mathsf{SymSum}_{\mathsf{Sim}}$ only applies to even order derivatives (this restriction is eventually fixed). So, for an $n-$variate (assuming $2|n$) function, $\mathsf{SymSum}_{\mathsf{Sim}}$ allows computing $(2, 4, \cdots, n)-$fold simple derivatives while $\mathsf{SymSum}_{\mathsf{Vec}}$ allows computing $(1, 2, \cdots, n/2)-$fold vectorial derivatives. Hence, this implies that $\mathsf{MAD\text{-}SymSum}_{\mathsf{Sim}} = 2 \times \mathsf{MAD\text{-}SymSum}_{\mathsf{Vec}} = \mathsf{MAD\text{-}ZeroSum}$, which makes $\mathsf{SymSum}_{\mathsf{Sim}}$ equivalent to $\mathsf{ZeroSum}$ in terms of MAD, thereby constituting the primary contribution of this work. We next show that the limitation of even order derivative can be easily handled by combining the even order simple derivative with an additional vectorial derivative reaching what we refer to as a hybrid derivative of odd order. We additionally show that $\mathsf{SymSum}_{\mathsf{Sim}}$ can be augmented by one round using linear structures due to Guo *et al.*. Interestingly, two rounds augmentation via linearization is not possible since the first aspect of degree difference due to Saha *et al.*'s work will not hold anymore and leads to fundamental observation that constitutes our next result.

Impossibility of $\mathsf{SymSum}_{\mathsf{Vec}}$ with 2-round linearization and its fix. Our second contribution is the proof of invalidity and eventual fix of a claim by Suryawanshi *et al.* in Africacrypt 2020 on extending the $\mathsf{SymSum}_{\mathsf{Vec}}$ property. The authors reported three extension techniques for augmenting $\mathsf{SymSum}_{\mathsf{Vec}}$: one and two-round linearizations and χ^{-1} trick on the hash digest. Our research reveals that all results reported with two-round linearization are invalid. We prove that after making two rounds linear, the highest degree monomials become dependent on the round-constants and hence Lemma 1 given by Saha *et al.* [SKC17] no longer holds. The proof idea is based on the observation that though the two rounds become linear, certain constants and linear terms get multiplied in the second round non-linear operation. We further show that though straight forward two-round linearization fails, some carefully chosen additional constraints (essentially making the linear subspace associated with constant terms null) can be added which will restore the independence of the highest degree monomials from constants, thereby making $\mathsf{SymSum}_{\mathsf{Vec}}$ effective again. The same can be achieved with $\mathsf{SymSum}_{\mathsf{Sim}}$ while doubling the MAD.

Table 1: Summary of the results using $\mathsf{SymSum_{Sim}}$ distinguisher. As evident $\mathsf{SymSum_{Sim}}$ improves (for SHA-3) and matches (for Xoodyak) all $\mathsf{SymSum_{Vec}}$ distinguishers. $\mathsf{SymSum_{Sim}}$ outperforms ZeroSum in all cases.

Variants	Rounds	ZeroSum	$\mathsf{SymSum_{Vec}}$	$\mathsf{SymSum_{Sim}}$	Augmentation Strategy†
SHAKE-128	11	2^{1025*}	~~MAD~~	$2^{1023*★}$	χ^{-1}
	10	2^{257}		2^{256}	1R Lin. $+\chi^{-1}$
	9	2^{65}		2^{64}	2R Lin. $+\chi^{-1}$
SHAKE-256	11	2^{1025*}	~~MAD~~	$2^{1023*★}$	χ^{-1}
	10	2^{257}		2^{256}	1R Lin. $+\chi^{-1}$
	9	2^{129}	2^{128}	2^{128}	1R Lin. $+\chi^{-1}$
	8	2^{33}	~~MAD~~	2^{32}	2R Lin. $+\chi^{-1}$
SHA-3-224	10	2^{1025*}	~~MAD~~	$2^{1023*★}$	Unaugmented
	9	2^{257}		2^{256}	1R Lin.
	8	2^{65}		2^{64}	2R Lin.
SHA-3-256	10	2^{1025*}	~~MAD~~	$2^{1023*★}$	Unaugmented
	9	2^{257}		2^{256}	1R Lin.
	8	2^{129}	2^{128}	2^{128}	1R Lin.
	7	2^{33}	~~MAD~~	2^{32}	2R Lin.
SHA-3-384	10	~~MAD~~		$\mathbf{2^{256}}$	1R Lin. $+\chi^{-1}$
	9	2^{129}	2^{128}	2^{128}	1R Lin. $+\chi^{-1}$
	8	2^{33}	~~MAD~~	2^{32}	2R Lin. $+\chi^{-1}$
SHA-3-512	10	2^{513}	~~MAD~~	$2^{511★}$	χ^{-1}
	9	~~MAD~~		$\mathbf{2^{128}}$	1R Lin. $+\chi^{-1}$
Xoodyak-Hash	6	2^{65}	2^{64}	2^{64}	Unaugmented
Bash-512	8	2^{257}	~~MAD~~	2^{256}	Unaugmented
Bash-265	9	2^{513}		2^{512}	Unaugmented
			Permutation		
Keccak	16	~~MAD~~		$\mathbf{2^{512}}$	6 + 10 (1B 1F)
	15	2^{257}	2^{256}	2^{256}	6 + 9 (1B 1F)
Xoodoo	12	2^{33}	~~MAD~~	2^{32}	6 + 6 (1B 1F)
Bash-f	20	2^{1025*}	~~MAD~~	2^{1024*}	10 + 10
	19	2^{1025*}		2^{1024*}	9 + 10
	18	2^{513}	2^{512}	2^{512}	9 + 9

$*$exceed complexity from birthday bound

† 1R: 1-round linearization

† 2R: 2-round linearization

† 1F: 1-round forward linearization

★: $\mathsf{SymSum_{Hyb}}$ (hybrid mode)

† 1B: 1-round backward linearization

~~MAD~~ denotes the inapplicability of the distinguisher as the required order of derivative exceeds the MAD for that distinguisher.

Best distinguisher on all SHA-3 variants, Keccak-p, Xoodyak-Hash, Xoodoo, Bash and Bash-f. Leveraging the doubling of MAD computable due to simple order derivatives, we mount $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguishers across all variants of fixed-length hash and extensible-output functions (XOFs). In the context of fixed-length hash functions, specifically in a 9-round scenario involving SHA-3-512, $\mathsf{SymSum}_{\mathsf{Sim}}$ outperforms ZeroSum by a significant factor of 2^{129}. When applied to 10-round SHA-3-384, $\mathsf{SymSum}_{\mathsf{Sim}}$ again outperforms ZeroSum by a substantial margin of 2^{257}. Moreover, in the same 10-round setting, for the remaining fixed-length SHA-3 variants SHA-3-224, SHA-3-256, and SHA-3-512, $\mathsf{SymSum}_{\mathsf{Sim}}$ maintains an advantage of factor 2 over ZeroSum. For the XOFs, SHAKE-128 and SHAKE-256, we successfully penetrate 11 rounds with the same advantage. It is worth mentioning that achieving 10 and 11 rounds for fixed-length hash functions and XOFs was previously unreachable with $\mathsf{SymSum}_{\mathsf{Vec}}$ due to the MAD barrier. $\mathsf{SymSum}_{\mathsf{Sim}}$ successfully overcomes the MAD barrier leading to an array of new distinguishers beating ZeroSum all along.

While $\mathsf{SymSum}_{\mathsf{Sim}}$ augmented with linearization improves or matches all existing results, for SHA-3-384, with one-round linearization and χ^{-1} trick, both $\mathsf{SymSum}_{\mathsf{Vec}}$ and ZeroSum fail to reach the MAD needed to distinguish. Interestingly, due to one order advantage over ZeroSum for the required MAD, $\mathsf{SymSum}_{\mathsf{Sim}}$ remains as the only working distinguisher with a complexity of 2^{256}. The same applies for SHA-3-512 with 2^{128}. Further, $\mathsf{SymSum}_{\mathsf{Sim}}$ is applied to Xoodyak-Hash giving the best result on 6 rounds with a distinguishing complexity of 2^{64}. Using the inside-out technique introduced by Aumasson and Meier on Keccak-f permutation [AM09a] and using one-round forward and one-round backward linearization, we mount $\mathsf{SymSum}_{\mathsf{Sim}}$ on *full-round* Xoodoo, giving us the lowest complexity distinguisher reported in literature with a complexity of 2^{32}. Following the same idea, we are able to distinguish 15 rounds of Keccak-p with a $(6 + 9)$ split for the inside-out technique. Combining Guo *et al.*'s [GLS16] linear structures with $\mathsf{SymSum}_{\mathsf{Sim}}$, we devise one-round forward and one-round backward linearization leading to a distinguisher with a complexity of 2^{256} that gives a symmetric-sum in the *forward* direction and a zero-sum in the *backward* direction. Moreover, we extend our analysis to 16 rounds of Keccak-p with a $(6 + 10)$ split, achieving a 2^{512} complexity distinguisher with similar properties. We also analyse the Bash hashing algorithms [AMMS16], which operate based on the sponge construction. The core of Bash is the Bash-f sponge function, representing the Logical-Rotation-Xor (LRX) symmetric cryptography schemes. Notably, this function has gained recognition as a standard in Belarusian cryptography. To the best of our knowledge, this is the first analysis conducted on the Bash cryptographic primitive. Using the inside-out technique, we have applied $\mathsf{SymSum}_{\mathsf{Sim}}$ up to 20 (out of 24) rounds with a complexity of 2^{1024}, though this complexity exceeds the birthday bound. It is worth noting that for 18 rounds, $\mathsf{SymSum}_{\mathsf{Sim}}$ exhibits a lower complexity than the birthday bound, specifically 2^{512}. Furthermore, the $\mathsf{SymSum}_{\mathsf{Sim}}$ strategy has been applied to the Bash-256 and Bash-512 algorithms for up to 9 and 8 rounds, respectively. All the results are summarized in Table 1. All provided distinguishers in this work have been verified through practical implementations whenever practically viable. Verification code is publicly available at the following link.

Organization

The remaining sections of the paper are organized as follows. In Section 2, we redefine the well-known definitions and Lemmas to ensure a clearer understanding of the concepts discussed in the paper. We also introduce and define the concept of MAD and the notion of symmetric states. Section 3 briefly describes cryptographic primitives. We then describe the $\mathsf{SymSum_{Vec}}$ distinguisher in Section 4. Also, exhibits its advantages over $\mathsf{SymSum_{Vec}}$, particularly in terms of the MAD. Section 5 explores the applicability of $\mathsf{SymSum_{Sim}}$ to the cryptographic primitives: SHA-3/Keccak-p, Xoodyak-Hash/Xoodoo and Bash/Bash-f, discussing the potential benefits and limitations in each case. Section 6 explores inapplicability of 2 round linearization with $\mathsf{SymSum_{Vec}}$ and provide a solution. Later in Section 7 we discussed the applicability and the advantage of $\mathsf{SymSum_{Sim}}$/$\mathsf{SymSum_{Hyb}}$ over ZeroSum and $\mathsf{SymSum_{Sim}}$. To validate our claims, Section 8 presents experimental verification and results. Finally, Section 9 concludes the paper by summarizing our findings and contributions.

2 Preliminaries

In this section, we begin by introducing the notations to be utilized. Subsequently, we provide a redefined version of some key definitions and lemmas that are widely recognized. The intention is to enhance the overall clarity and comprehension of the content. We also introduce additional definitions that are essential for achieving an improved comprehension of our proposed approach.

2.1 Notations

We use bold lowercase letters to represent vectors in a binary field. For any n-bit vector $\mathbf{x} \in \mathbb{F}_2^n$, its i-th coordinate is denoted by x_i, thus we have $\mathbf{x} = (x_{n-1}, ..., x_0)$.

The Algebraic Normal Form (ANF) of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be defined as $f(\mathbf{x}) = \bigoplus_{\mathbf{u}\in\mathbb{F}_2^n} a_{\mathbf{u}}^f \mathbf{x}^{\mathbf{u}}$ where $a_{\mathbf{u}}^f \in \mathbb{F}_2$. Using ANF, we can represent any Boolean function. The degree of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, denoted by $d^\circ f$, is the degree of the largest monomial in the ANF of f, i.e., $d^\circ f = \max_{\mathbf{u}\in\mathbb{F}_2^n, a_{\mathbf{u}}^f \neq 0} wt(\mathbf{u})$.

2.2 Derivative Operations on Boolean Functions

Boolean Differential Calculus [BP81, Tha81] allows us to capture the settings when changes of the values of Boolean variables forms an integral part in analysis of Boolean or vectorial functions. Consider an $n-$variate Boolean function, denoted as $f(x_0, x_1, ..., x_{n-1})$. The simple derivative of this Boolean function, with respect to a particular variable x_i, is denoted as $(\frac{\delta f}{\delta x_i})$ and represents the *change* in the function's output when the value of x_i is changed and is formally captured as below.

Definition 1 (Simple Derivative [PS19]). *Let $f(x_0, x_1, x_2, \ldots, x_{n-1})$ be a Boolean function. Then the Boolean derivative of f with respect to x_i is defined as:*

$$\frac{\delta f}{\delta x_i} = f(x_0, x_1, \ldots, x_i, \ldots, x_{n-1}) \oplus f(x_0, x_1, \ldots, \overline{x}_i, \ldots, x_{n-1})$$

Calculating a sequence of m simple derivatives with respect to m variables of a Boolean function is an m-fold simple derivative operation. The definition is given below.

Definition 2 ($m-$fold Simple Derivative [PS04])**.** *Let $f(\mathbf{x_0}, \mathbf{x_1})$ be an $n-$variate Boolean function. Then the $m-$fold simple derivative of f with respect to $\mathbf{x_0} = \{x_{i_1}, \ldots, x_{i_m}\}$ is defined as:*

$$\frac{\delta^m f}{\delta \mathbf{x_0}} = \frac{\delta f}{\delta x_{i_m}}\left(\ldots\left(\frac{\delta f}{\delta x_{i_2}}\left(\frac{\delta f}{\delta x_{i_1}}\right)\right)\ldots\right) = \bigoplus_{\mathbf{x_0} \in \mathbb{F}_2^m} f(\mathbf{x_0}, \mathbf{x_1} = \mathbf{const} \in \mathbb{F}_2^{n-m})$$

Remark 1. *Due to the commutativity property, the order in which the variables are processed does not affect the result. Therefore, when computing a derivative of order m, we consider a subset $\mathbf{x_0}$ of input variables, where $|\mathbf{x_0}| = m$, from the set $\{x_0, ..., x_{n-1}\}$. The main concept is to let variable $\in \mathbf{x_0}$ take values from $\mathbb{F}_2^m$ while the remaining variables have a constant (**const**) value. The following lemma is well-known and forms the basis of the distinguisher that is devised in this work.*

Lemma 1 ([PS19])**.** *The result of $m-$fold simple derivative operations of $f(x_1, x_2, \cdots, x_k, x_{k+1}, \cdots, x_n)$ with regard to $\mathbf{x_0} = (x_1, x_2, \cdots, x_k)$ depends only on $(n-k)$ variables $\mathbf{x_1} = (x_{k+1}, \cdots, x_n)$. Therefore, $m-$fold simple derivative operations describe properties of whole subspaces specified by $\mathbf{x_1} = \mathbf{const}$.*

ZeroSum as Higher Order Simple Derivatives

A concept closely related to the m-fold simple derivative is the concept of ZeroSum Distinguishers [AM09b, BC10, BCC11]. For a given function F , the underlying idea of ZeroSum is to show the existence of a set of input states whose XOR-sum leads to zero and whose images under F also sum up to zero. If the same can be achieved with a complexity that beats the generic effort, we have a distinguisher for F. It is easy to note that for an $n-$variate Boolean function ($\mathbb{B}$) with algebraic degree d, the $m-$fold simple derivative (input-sum is zero, since it is computed over $\mathbb{F}_2^m$) for any $m > d$ will lead to an output-sum of zero because the derivative-order is greater than the algebraic degree giving us a ZeroSum distinguisher. For an unknown $\mathbb{B}$, the research challenge lies in tightly estimating the value of d which directly impacts the complexity of computing the ZeroSum.

Vectorial Derivatives

Vectorial derivatives capture the value change between pairs of function values due to *simultaneous* value change of an arbitrary but fixed subset of variables $\mathbf{x_1} \subseteq \mathbf{x}$. Simple derivatives may be considered as a special case of the vectorial derivatives where the vector $\mathbf{x_1}$ contains only the single variable x_i.

Definition 3 (Vectorial Derivative [PS04])**.** *Let $\mathbf{x_1} = \{x_0, x_1, \ldots, x_{k-1}\}$, $\mathbf{x_2} = \mathbf{x} \setminus \mathbf{x_1}$ be two disjoint sets of n variables, $\mathbf{x} = \{x_0, x_1, \ldots, x_{n-1}\}$ and $f(\mathbf{x}) = f(x_0, x_1, \ldots, x_{n-1}) = f(\mathbf{x_1}, \mathbf{x_2})$ be a Boolean function over n variables, then the* *vectorial derivative of f with respect to $\mathbf{x_1}$ is defined as:*

$$\frac{\delta f}{\delta \mathbf{x_1}} = f(\mathbf{x_1}, \mathbf{x_2}) \oplus f(\overline{\mathbf{x}}_\mathbf{1}, \mathbf{x_2})$$

Analogous to higher order simple derivatives, an m-fold vectorial derivative operation is formally defined as below.

Definition 4 (m-fold Vectorial Derivative [PS04, SKC17])**.** *Suppose the input variables of a Boolean function $f(x_0, x_1, \dots, x_{n-1})$) has $(m+1)$ partitions and $f(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m}, \mathbf{x_{m+1}}) = f(x_0, x_1, \cdots, x_{n-1})$. Then,*

$$\left.\frac{\delta^m f}{\delta \mathbf{x_m} \cdots \delta \mathbf{x_2} \delta \mathbf{x_1}}\right|_{\substack{(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m}) \\ = (\mathbf{c_1}, \mathbf{c_2}, \cdots, \mathbf{c_m})}} = \left.\frac{\delta}{\delta \mathbf{x_m}} \left(\cdots \left. \left(\frac{\delta}{\delta \mathbf{x_2}} \left(\left. \frac{\delta f}{\delta \mathbf{x_1}} \right|_{\mathbf{x_1} = \mathbf{c_1}} \right) \right|_{\mathbf{x_2} = \mathbf{c_2}} \right) \cdots \right) \right|_{\mathbf{x_m} = \mathbf{c_m}}$$

*is the $\mathbf{m}-$**fold vectorial derivative** of the Boolean function $f(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m}, \mathbf{x_{m+1}})$ with regards to the m partitions $\{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m}\}$.*

Again, due to commutativity, the sequence in which the variables are processed does not affect the result. Thus, the partitioning of the variables corresponds to the selection of special subspaces ν of size 2^m, where

$$\nu = \begin{bmatrix} \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_{m-1}} & \mathbf{c_m} \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_{m-1}} & \overline{\mathbf{c_m}} \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \overline{\mathbf{c_{m-1}}} & \mathbf{c_m} \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \overline{\mathbf{c_{m-1}}} & \overline{\mathbf{c_m}} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \overline{\mathbf{c_1}} & \overline{\mathbf{c_2}} & \cdots & \overline{\mathbf{c_{m-1}}} & \overline{\mathbf{c_m}} \end{bmatrix}.$$

Consequently, we have the following expression for computing such a derivative:

$$\left.\frac{\delta^m f}{\delta \mathbf{x_m} \cdots \delta \mathbf{x_2} \delta \mathbf{x_1}}\right|_{\substack{(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m}) \\ = (\mathbf{c_1}, \mathbf{c_2}, \cdots, \mathbf{c_m})}} = \bigoplus_{(\mathbf{x_1}, \dots, \mathbf{x_m}) \in \nu} f(\mathbf{x})$$

Example 1. *To clarify the idea of m-fold vectorial derivative, we consider the following example. Let us consider the 6 variable Boolean function $f(x_0, x_1, x_2, x_3, x_4, x_5) = x_0x_1x_2x_3 + x_1x_2x_3x_4 + x_2x_3 + x_0x_5$. We take 2-fold vectorial derivative where $\mathbf{x_1} = \{x_0, x_3\}$, $\mathbf{x_2} = \{x_1, x_4\}$. We take derivative in the direction of $\mathbf{x_1} = \mathbf{c_1}$ and $\mathbf{x_2} = \mathbf{c_2}$ where $\mathbf{c_1} = 01$, $\mathbf{c_2} = 11$. Thus, we have*

$$\nu = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Finally, we compute 2-fold vectorial derivative as follows:

$$\left.\frac{\delta^2 f}{\delta \mathbf{x_2 x_1}}\right|_{\substack{(\mathbf{x_1},\mathbf{x_2})\\=(\mathbf{c_1},\mathbf{c_2})}} = \bigoplus_{(\mathbf{x_1},\mathbf{x_2})\in\nu} f(\mathbf{x}) = x_2.$$

Lemma 2 ([PS19])**.** *The vectorial derivative $\frac{\delta f(\mathbf{x_1},\mathbf{x_2})}{\delta \mathbf{x_1}}$ is a Boolean function that depends generally on all variables of $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2})$.*

Lemma 2 given above is a well-known result that has been extensively studied and proven in various references. Therefore, it is a fundamental concept in the field and its proof is readily available in [PS04]. It is worth noting the sharp contrast that lemma 2 has with lemma 1 in terms of dependence of the derivative on the constituent variables of the original function.

SymSum$_{\mathsf{Vec}}$ as Higher Order Vectorial Derivatives

In FSE 2017 Saha *et al.* introduced the SymSum$_{\mathsf{Vec}}$ distinguisher and applied it to SHA-3 to achieve a factor 4 improvement over ZeroSum. For SymSum$_{\mathsf{Vec}}$ over a function f, $\sum x = 0$ like ZeroSum but $\sum f(x)$ admits a deterministic symmetry. Saha *et al.* established the equivalence of SymSum$_{\mathsf{Vec}}$ with $m-$fold vectorial derivatives taken over special symmetric subspaces with partition-size $= 2$.

Maximum Attainable Derivative

Lemma 1 and 2 provide valuable insights into the behavior of derivatives (both simple and vectorial) in relation to the variables they are dependent on. In this context, it is crucial to understand an important parameter: the maximum attainable derivative (MAD). This forms an important observation in this study and is formally defined below. Our main contribution is the doubling of MAD-SymSum$_{\mathsf{Vec}}$ for MAD-SymSum$_{\mathsf{Sim}}$ making it equal to MAD-ZeroSum.

Definition 5 (Maximum Attainable Derivative (MAD))**.** *The maximum attainable derivative refers to the highest value of m for an $m-$fold derivative such that the dimension of the subspace spanned by fixed variables*[‡] */becomes zero.*

2.3 Notion of Partial/Full Self-Symmetric State

To formally define the SymSum$_{\mathsf{Sim}}$ distinguisher, it is necessary to revisit and introduce some basic definitions of symmetric states. We define symmetric states with respect to the Keccak, Xoodoo and Bash-f states. A brief description of these functions along with their states are given in the next section.

Definition 6 (Substate)**.** *Let us consider a state S with n slices. A Substate (σ) is a collection of either the first or the last $n/2$ slices of a state. The value of n is 64 for*

[‡]Recall, that fixed variables are the ones that remain fixed across the derivative computation.

Keccak and Bash-f while the value of n is 32 for Xoodoo. Thus we have

$$\sigma \in \begin{cases} \{0,1\}^{25\times 32} & \textit{for Keccak} \\ \{0,1\}^{12\times 16} & \textit{for Xoodoo} \\ \{0,1\}^{24\times 32} & \textit{for Bash-}f \end{cases}$$

We can now define a self-symmetric state as follows

Definition 7 (Self-Symmetric State [SKC17]). *Let $\mathcal{S} = \sigma_1 || \sigma_2$ be a state with two substates σ_1 and σ_2. Then $\mathcal{S}$ is called Self-Symmetric if the following condition holds for its substates*

$$\mathcal{S} = \sigma_1 || \sigma_2 \textit{ where } \sigma_1 = \sigma_2 \in \{0,1\}^{|\sigma_1|=|\sigma_2|}$$

A visual representation of *self-symmetric state* is shown in the left half of Fig. 1.

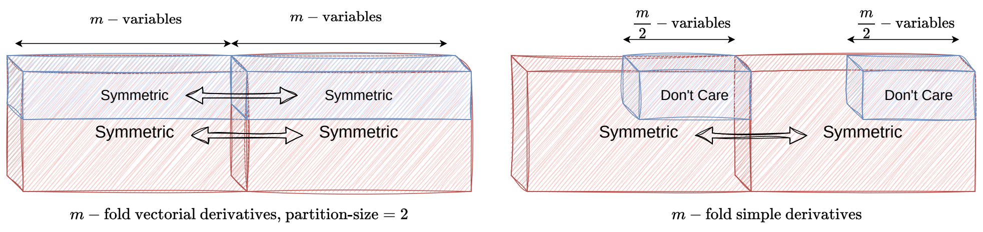

m − variables
m − variables
Symmetric
Symmetric
Symmetric
Symmetric
m − fold vectorial derivatives, partition-size = 2
$\frac{m}{2}$ − variables
$\frac{m}{2}$ − variables
Don't Care
Don't Care
Symmetric
Symmetric
m − fold simple derivatives

Fig. 1: Comparative view of $m-$fold vectorial and simple derivatives in terms of the nature of self-symmetry induced while computing $\mathsf{SymSum}_{\mathsf{Vec}}$ (left) and $\mathsf{SymSum}_{\mathsf{Sim}}$ (right)

We now define a relaxed version of the self-symmetric state known as a structurally symmetric state

Definition 8 (Structurally Symmetric State). *Let $\mathcal{S} = \sigma_1 || \sigma_2$ be a state with two substates σ_1 and σ_2. Then $\mathcal{S}$ is called Structurally Symmetric if the following conditions hold for its substates. Note that the symmetric and asymmetric parts of the substates are always in respective symmetric positions.*

$$\mathcal{S} = \sigma_1 || \sigma_2 = (\sigma_1^s, \sigma_1^a) || (\sigma_2^s, \sigma_2^a) \textit{ where } \begin{cases} \sigma_1^s = \sigma_2^s \in \{0,1\}^{|\sigma_1^s|=|\sigma_2^s|} \\ (\sigma_1^a || \sigma_2^a) \in \{0,1\}^{|\sigma_1^a|+|\sigma_2^a|=2\times|\sigma_1^a|} \end{cases}$$

It is evident from the given definition that a *Structurally Symmetric* State is partially symmetric in terms of its values. For example, σ_1^s and σ_2^s represent the symmetric part, which have the same size and same values. On the other hand, σ_1^a and σ_2^a maintain only symmetric positions[§] and the same size, but they do not necessarily have the same values. A visual representation of this is provided in the right half of Fig. 1.

3 Background Details

We apply our proposed distinguisher on SHA-3 [BDPA11], Xoodoo permutation [DHAK18], and Xoodyak-Hash [DHP+20], which are well-known SPN functions that have attracted much interest and are frequently used in various applications. Furthermore, we extend our analysis to include the application of $\mathsf{SymSum}_{\mathsf{Sim}}$ on Bash, a standardized Belarusian hash function [AMMS16]. This section provides concise descriptions of these SPN functions. A thorough introduction to the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher and its extension are also given. It is important to comprehend these components to fully appreciate how our method addresses the problem, specifically low MAD.

3.1 Secure Hash Algorithm (SHA-3) [BDPA11]

SHA-3 is a cryptographic hash function standardized by NIST in 2015. The SHA-3 family of algorithms is based on the Keccak-p permutation [BDPA13]. The Keccak structure is based on the Sponge construction [BDPA08], which produces an element of length $\mathbb{F}_2^n$ from an element of length $\mathbb{F}_2^m$ with arbitrary lengths n and m. The permutation is applied on a finite-state of $b = r + c$ bits, where r is rate and c is capacity. Each round of Keccak-p permutation has 5 mappings, $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

θ is a linear mapping where the parity ($P[x, *, z]$) of the neighboring two columns are XORed with every bit $A[x, y, z]$ of the state. ρ is another linear mapping which rotates ($\lll$) each lane by predefined offset values. π is the third linear mapping which permutes the positions of the lanes. χ is the only non-linear mapping and operates on rows as shown below. Finally, ι adds a unique round-constant (RC) to lane $A[0, 0, *]$.

$$\theta : \begin{cases} A[x,y,z] & = A[x,y,z] \oplus P[(x-1) \bmod 5, *, z] \\ & \oplus P[(x+1) \bmod 5, *, (z-1) \bmod 64] \\ & \text{where } P[x,*,z] = \oplus_{j=0}^{4} A[x,j,z] \end{cases}$$

$$\rho : \quad A[x,y,z] = A[x,y,z \lll t] \ \forall \ x,y \in \{0,\ldots,4\}$$

$$\pi : \quad A[y,(2x+3y) \bmod 5, *] = A[x,y,*] \ \forall \ x,y \in \{0,\cdots,4\}$$

$$\chi : \quad A[x,y,z] = A[x,y,z] \oplus (\sim A[x+1,y,z]) \wedge A[x+2,y,z]$$

$$\iota : \quad A[0,0,*] = A[0,0,*] \oplus RC$$

The standard defines four hash functions: SHA-3-224, SHA-3-256, SHA-3-384 and SHA-3-512. The ability of SHA-3 to produce variable-length hash values utilizing the SHAKE-128/SHAKE-256 method is an intriguing feature. In contrast to the other SHA-3 variations, SHAKE-128/SHAKE-256 allows for producing hash outputs of any length.

[§]The positions may be selected in any random order while maintaining the symmetric position constraint.

They use the same underlying construction as SHA-3 but with a different padding rule i.e. "0110*1" for SHA-3 and "11110*1" for SHAKE.

3.2 Xoodyak-Hash [DHP+20]

Xoodyak was one of the ten NIST-LWC finalists, a versatile cryptographic primitive combining sponge construction and Xoodoo [DHAK18] permutation based on an operational mode termed Cyclist. Xoodoo is a 48-byte permutation introduced by Daemen *et al.* at ToSC 2018. It operates on a 3-dimensional array of size $4 \times 3 \times 32 = 384$ bits, where row, column, and lane are acronyms that refer to the 1-dimensional array in the x, y, and z directions. Slices, sheets, and planes refer to the 2-dimensional arrays in the $(x, y), (y, z)$, and (x, z) directions. Each round of the Xoodoo permutation has five mappings $X = \rho_{east} \circ \chi \circ \iota \circ \rho_{west} \circ \theta$.
θ is a linear mapping that provides diffusion. The column parity of 2 near-by columns are XORed with each bit of the column. ρ_{west} is another linear mapping that rotates the bits of a plane in the x and z directions. ι adds a unique round constant to lane 0 of plane A_0. χ is the only non-linear mapping and operates on each column independently. ρ_{east} is another linear mapping like ρ_{west} with different rotations to each plane. The notation $A \lll (t, v)$ used below rotates the bits of plane A, such that a bit at position (x, z) goes to the position $(x + t, z + v)$ in a cyclic manner.

$$\theta : \begin{cases} A_y &= A_y \oplus E \text{ where } y \in \{0,1,2\} \\ E &= P \lll (1,5) \oplus P \lll (1,14) \\ P &= A_0 \oplus A_1 \oplus A_2 \end{cases}$$

$$\rho_{west} : \begin{cases} A_1 &= A_1 \lll (1,0) \\ A_2 &= A_2 \lll (0,11) \end{cases}$$

$$\iota : \quad A_0 = A_0 \oplus RC_i$$

$$\chi : \begin{cases} A_y &= A_y \oplus B_y \text{ where } y \in \{0,1,2\} \\ B_y &= \sim A_{y+1 \bmod 3} \cdot A_{y+2 \bmod 3} \end{cases}$$

$$\rho_{east} : \begin{cases} A_1 &= A_1 \lll (0,1) \\ A_2 &= A_2 \lll (2,8) \end{cases}$$

Xoodyak can be initialized in either hash or keyed mode. The number of rounds in Xoodyak-Hash is 12, which provides the designed primitive with a sufficient safety margin against all potential attacks. Xoodyak-Hash generates 256-bits (32 bytes) of digest and as a result, offers 128-bit security. The rate part of the Xoodyak state consists of 130 bits which includes the maximum input string size of 16 bytes along with 2 bits from the padding. In our analysis, we only consider input strings whose size equals 16 bytes. The input string is appended by padding of the type '01'||'00'*||'01' where 'xx' depicts 1 byte of the state. The final byte '01' is known as the domain separator.

3.3 Bash [AMMS16]

Bash is a family of hashing algorithms standardized in Belarus. It implements the sponge construction over the underlying LRX permutation Bash-f. Bash-f maintains a state size of 1536 bits maintained as a 3-dimensional array of size $3 \times 8 \times 64$. The Bash-f permutation consists of 24 rounds, where each round is a composition of the transformations $L3$, $S3$, P applied on each of the 8 vertical planes and a round constant addition. $L3$ is a linear mapping which uses rotations and XORs to *overwrite* the initial state. $S3$ is a non-linear mapping that uses the $\neg, \wedge, \vee$ and $\oplus$ operations in decreasing order of precedence. P shuffles the 24 words present in the state by rotating the horizontal planes across the vertical axis and simultaneously permuting the words in each plane. The round constants added are updated based on a linear feedback shift register with a primitive characteristic polynomial.

$$L3 : \begin{cases} W_0 & \leftarrow w_0 \oplus w_1 \oplus w_2 \\ W_1 & \leftarrow w_1 \oplus \mathrm{RotHi}^{m_1}(w_0) \oplus \mathrm{RotHi}^{n_1}(W_0) \\ W_2 & \leftarrow w_2 \oplus \mathrm{RotHi}^{m_2}(w_2) \oplus \mathrm{RotHi}^{n_2}(w_1 \oplus \mathrm{RotHi}^{n_1}(W_0)) \end{cases}$$

$$S3 : \Big\{ (W_0, W_1, W_2) \quad \leftarrow (W_0 \oplus W_1 \vee \neg W_2, W_1 \oplus W_0 \vee W_2, W_2 \oplus W_0 \wedge W_1)$$

$$P(u) : \begin{cases} \pi_0(u) + 8, & 0 \le u < 8 \qquad \pi_0(v) = (v + 2(v \bmod 2) + 7) \bmod 8, \\ \pi_1(u-8) + 16, & 8 \le u < 16 \qquad \pi_1(v) = v + 1 - 2(v \bmod 2), \\ \pi_2(u-16), & 16 \le u < 24 \qquad \pi_2(v) = (5v + 6) \bmod 8 \end{cases}$$

The different Bash algorithms differ by their security level $l \in \{16, 32, 48, \dots, 256\}$. For the Bash algorithm with security level l, the first $1536 - 4l$ bits form the rate and the remaining $4l$ bits form the capacity. The length of the returned hash digest is $2l$. The input is padded with the padding rule "010*" such that the padded message is a multiple of $r = 1536 - 4l$. The initial state of Bash is initialized to all zeros except the last lane which is initialized with the binary representation of the integer $l/4$. The input is then split into blocks of length r and absorbed according to the sponge construction and then, the output of size $2l$ is squeezed out.

3.4 SymSum Distinguisher [SKC17]

The $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher [SKC17], introduced by Saha *et al.*, was designed to analyze the round function of SHA-3/Keccak. This distinguisher takes advantage of the algebraic properties of the Keccak round function. It is well-known that all the components of the Keccak round function, except for the round-constant addition are translation invariant in the $z-$axis. One implication of this observation is that they map self-symmetric (Refer Definition 7) input states to self-symmetric output states, a property that was exploited in the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher. Thus, to observe the self-symmetry property at the output, it is necessary to eliminate the influence of the round constants. The key insight of the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher is that the round constants used in the SPN round function of Keccak do not affect the highest degree monomials. In order to remove the impact of the round constants, Saha *et al.* used the concept of the $m-$fold vectorial derivative (Refer Definition 4) which allows for differentiation over a specific subspace while attempting to replicate the creation of self-symmetric input states. Through the computation of the $m-$fold vectorial derivative, a function that is independent of round constants can be generated. This function is specifically designed to maintain self-symmetry. Saha *et al.* observed that when SHA-3 was applied to these states, the XOR-sum of the resulting outputs over all hash values was also symmetric. In contrast to the ZeroSum distinguisher, which calculates the m-fold simple derivatives, the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher makes use of the m-fold vectorial derivative achieving a factor of 2^2 improvement over the ZeroSum distinguisher.

Extension and Generalization of $\mathsf{SymSum}_{\mathsf{Vec}}$

Suryawanshi *et al.* [SSS20] enhanced the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher by combining the $\mathsf{SymSum}_{\mathsf{Vec}}$ property with linear structures proposed in [DMP$^+$15, GLS16]. Although the combination of linear structures reduced the advantage of $\mathsf{SymSum}_{\mathsf{Vec}}$, the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher still maintained a factor of 2 advantage compared to the ZeroSum distinguisher. The $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher is not only limited to the analysis of Keccak/SHA-3 alone but its versatility was shown by Suryawanshi *et al.* [SS23] by applying it to other SPN functions such as Xoodoo and Xoodyak-Hash. Overall, the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher always outperforms the ZeroSum distinguisher with a minimum improvement factor of 2 for those SPN functions. In the next section we introduce the idea of computing symmetric-sum using simple derivatives while making it orders of magnitude more effective than ZeroSum.

4 Introducing $\mathsf{SymSum}_{\mathsf{Sim}}$: Symmetric-Sum with Simple Derivatives

In this section, we present a new observation that redefines the entire body of work inspired by Saha *et al.*'s research on the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher in FSE 2017 [SKC17]. Upon a closer examination it becomes clear that the authors aimed to explain why applying Keccak on self-symmetric input states leads to self-symmetry in the output-sum. The entire theory was developed with the purpose of explaining this result. Consequently, it is evident that there is an affinity for self-symmetric states when generating inputs for the computation of higher order derivatives. The concept of self-symmetry is illustrated in the left portion of Fig. 1. The authors of [SKC17] noted that in order to differentiate functions across specific subspaces and reproduce self-symmetric input states in the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher, an m-fold vectorial derivative with a partition-size of 2 is essential. However, this partition-size implies that $2m$ variables will be consumed, resulting in a doubling of the rate of exhaustion in the number of variables that can be used for the derivative. Conversely, this restriction does not apply to the ZeroSum distinguisher as it utilizes m-fold simple derivatives. It is worth noting that simple derivatives can be viewed as a special case of vectorial derivatives with a partition-size of 1. As a result, the MAD for $\mathsf{SymSum}_{\mathsf{Vec}}$ is half that of the MAD for ZeroSum. Hence, while $\mathsf{SymSum}_{\mathsf{Vec}}$ outperforms traditional ZeroSum distinguishers for smaller rounds, the application of vectorial derivatives imposes a significant limitation on the applicability of $\mathsf{SymSum}_{\mathsf{Vec}}$ in higher rounds.

Structurally Symmetric States. To address this limitation, we have introduced a novel distinguisher named $\mathsf{SymSum}_{\mathsf{Sim}}$, leveraging an interesting property of the simple order derivative (given in Lemma 1), which states that *the resultant of a simple order derivative is independent of the derivative variables.* $\mathsf{SymSum}_{\mathsf{Sim}}$ deliberately uses structurally symmetric input states which permit asymmetry in the variables used for the derivative computation while maintaining symmetry in the remaining state. The visual representation of this structural configuration is depicted on the right side of Fig. 1, accompanied by a formal definition of $\mathsf{SymSum}_{\mathsf{Sim}}$ outlined below.

Definition 9 ($\mathsf{SymSum}_{\mathsf{Sim}}$)**.** *Let us consider an* SPN *function,* $F : f_2^n \to f_2^m$*. If* $\{X_1, X_2, \ldots, X_l\}$ *be the structurally-symmetric input set of* F *such that the input-sum* $\bigoplus_{i=1}^{l} X_i = 0$*, then the output-sum* $\bigoplus_{i=1}^{l} F(X_i)$ *will exhibit a symmetry referred to as* $\mathsf{SymSum}_{\mathsf{Sim}}$*. Here, we compute the* k^{th} *order derivative, where* $l = 2^k$*. Additionally, any state* X_i *exhibits symmetry for* $n - k$ *positions owing to structural-symmetry.*

$\mathsf{SymSum}_{\mathsf{Sim}}$ necessitates a structurally symmetric state, indicating that the derivative variables are placed symmetrically without a strict requirement for identical values. Similar to the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher, $\mathsf{SymSum}_{\mathsf{Sim}}$ takes advantage of the insight that round constants do not affect the highest-degree monomials. However, unlike $\mathsf{SymSum}_{\mathsf{Vec}}$, the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher leverages simple derivatives to overcome the aforementioned limitation. This increases the effective MAD, enabling penetration into higher rounds of SPN functions.

Although the requirement of symmetry in the derivative variables can be relaxed, the requirement for the rest of the state to exhibit symmetry dictates the derivative variables to assume symmetric positions. Thus, as demonstrated in Fig. 1, each half of the state will need $m/2$ derivative variables in symmetric positions for an m-fold simple derivative. This state acts as a representative state based upon which our input states will be generated. The following proposition can now be expressed based on the insights from the observation.

Proposition 1. *The* (d°*SHA-3* $- 1$) *and* (d°*Xoodoo*)$-$*fold* ***simple*** *derivative of SHA-3 and Xoodoo respectively evaluated using only* ***structurally*** *symmetric input states will preserve the symmetric property.*

It is evident that the symmetric part of the structurally symmetric state constitutes the fixed variables while the asymmetric part constitutes the independent variables. By definition, the symmetric part of both the substates are identical, which means that the overall size of the symmetric part must be even. Hence, computing an even order derivative is straightforward as the size of the remaining state is even. However, the size of the remaining state when computing an odd order derivative will be odd, thereby making symmetry in the fixed variables impossible. The complete process for the even order and the odd order derivative is described here.

$\mathsf{SymSum}_{\mathsf{Sim}}$ and the Even Order Derivative

The $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher offers the significant advantage of doubling the MAD compared to the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher, thereby increasing its effectiveness in penetrating more rounds. For an n-variate (assuming $2|n$) function, $\mathsf{SymSum}_{\mathsf{Sim}}$

allows for computing $(2, 4, \cdots, n)$-fold simple derivatives, while $\mathsf{SymSum}_{\mathsf{Vec}}$ allows for computing $(1, 2, \cdots, n/2)$-fold vectorial derivatives. This implies that $\mathsf{MAD}\text{-}\mathsf{SymSum}_{\mathsf{Sim}} = 2 \times \mathsf{MAD}\text{-}\mathsf{SymSum}_{\mathsf{Vec}} = \mathsf{MAD}\text{-}\mathsf{ZeroSum}$, making $\mathsf{SymSum}_{\mathsf{Sim}}$ equivalent to ZeroSum in terms of MAD. For the even order derivative we prove the following result.

Theorem 1. *The symmetry property is preserved when the $(d^\circ f)-$fold simple derivative of a function f is calculated at symmetric positions using structurally-symmetric input states where $(d^\circ f)$ is even.*

Proof. Let f be a Boolean function over variables $(x_0, x_1, \ldots, x_{n-1})$ and $d = d^\circ f$. The core idea lies in the fact that the output of a simple derivative is determined by the fixed variables only (Recall Lemma 1).

Let us consider $d-$fold simple derivative of f on symmetric state X, where the derivative is taken at symmetric positions and $2|d$. Let the derivative variables be denoted by vector $\mathbf{x_i}$ and the $d-$fold simple derivative as $\frac{\delta^d f}{\delta \mathbf{x_i}}$ such that

$$\begin{aligned}\frac{\delta^d f}{\delta \mathbf{x_i}} &= \frac{\delta f}{\delta x_{i_1} \delta x_{i_2} \ldots \delta x_{i_d}} \\ &= \frac{\delta f}{\delta x_{i_d}} \left(\cdots \left(\frac{\delta f}{\delta x_{i_2}} \left(\frac{\delta f}{\delta x_{i_1}} \right) \right) \cdots \right) \\ &= f(X^{'}) \text{ such that } X^{'} = X \setminus \mathbf{x_i}\end{aligned}$$

It is evident that $X^{'}$ consists of the symmetric part of the state, thus $f(X^{'})$ will also be symmetric. From Lemma 1, we know that the result of a $d-$fold simple derivative of a function only depends on the fixed variables (symmetric part of the state). Therefore, the resultant output $\frac{\delta^d f}{\delta x_{\mathbf{i}}}$ will also be symmetric. Hence, the symmetry property is preserved. □

Beyond Conventions: $\mathsf{SymSum}_{\mathsf{Sim}}$ and the Odd Order Derivative

The primary limitation of $\mathsf{SymSum}_{\mathsf{Sim}}$ lies in its applicability being restricted solely to even-order derivatives due to its structural constraints. When applied to structurally symmetric states, computing odd-order derivatives disrupts the symmetry of the fixed variables, rendering the use of odd-order derivatives impractical in the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher. We propose a novel approach called $\mathsf{SymSum}_{\mathsf{Hyb}}$ to address this issue. In $\mathsf{SymSum}_{\mathsf{Hyb}}$, odd-order derivatives are computed in a hybrid mode, while even-order derivatives are calculated using simple-order derivatives. This hybrid approach allows us to overcome the limitations associated with odd-order derivatives in the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher. Now, we present a formal definition of $\mathsf{SymSum}_{\mathsf{Hyb}}$.

Definition 10 ($\mathsf{SymSum}_{\mathsf{Hyb}}$)**.** *Let us consider an* SPN *function, $F : f_2^n \rightarrow f_2^m$. If $\{X_1, X_2, \ldots X_l\}$ be the structurally-symmetric input set of F such that the input-sum $\bigoplus_{i=1}^{l} X_i = 0$, then the output-sum $\bigoplus_{i=1}^{l} F(X_i)$ will exhibit a symmetry referred to* *as the* $\mathsf{SymSum}_{\mathsf{Hyb}}$. *Here, we compute the* k^{th} *order derivative, where* $l = 2^k$. *Additionally, the state* X_i *exhibits symmetry for* $n - k$ *(if* $2|k$*) and* $n - (k+1)$ *(if* $2\not|k$*) positions respectively owing to structural-symmetry.*

Suppose we want to compute the k-th order derivative of an n-bit function where k is odd, then we will take $n-(k+1)$ size structurally symmetric state and compute the derivative on the remaining $k+1$ variables. In order to compute the k-th order derivative using $k+1$ variables, we use a hybrid method where we compute $k-1^{th}$ order simple derivative and a single order vectorial derivative, thereby consuming $(k-1)+2 = k+1$ variables. We also ensure that the positions of the vectorial derivative are symmetric, thereby satisfying all symmetry requirements.

5 $\mathsf{SymSum}_{\mathsf{Sim}}$ on Different SPN Hash Constructions

In this section, the practical application of the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher is explored. The proposed $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher is applied to the SPN functions Xoodyak-Hash [DHP+20]/Xoodoo [DHAK18] and SHA-3 [BDPA11]/Keccak-f [BDPA11]. Additionally, we showcase the practical implementation of the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher on the Belarusian hash function Bash/Bash-f [AMMS16], which to the best of our knowledge, has not been previously analyzed. Through this exploration, we highlight the specific advantages of the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher compared to the ZeroSum and $\mathsf{SymSum}_{\mathsf{Vec}}$.

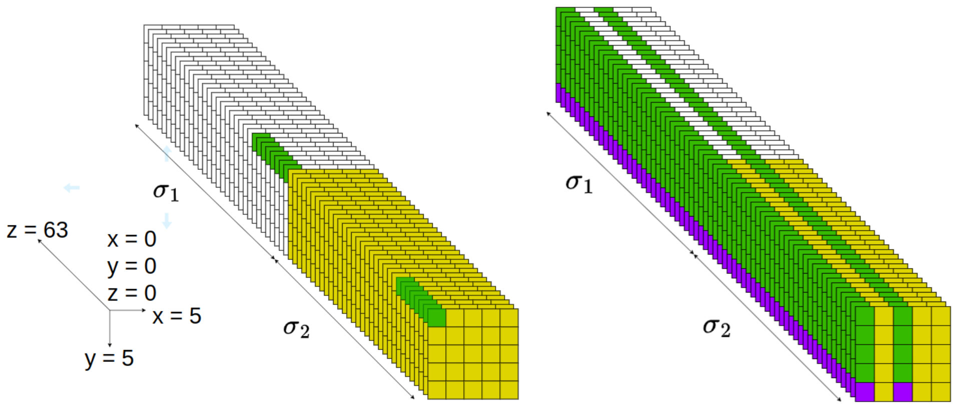
5.1 $\mathsf{SymSum}_{\mathsf{Sim}}$ Distinguisher on Keccak-p

We start the application of $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher to analyze symmetric behaviour of Keccak-p and use it for distinguishing attack. Our main objective is to gain insight into how the symmetry property is preserved when computing the d-fold simple derivative of Keccak-p using *structurally symmetric* input states as stated below.

Corollary 1. *By using structurally symmetric input states,* $(d^\circ \mathsf{Keccak}{-}p){-}$*fold simple derivative of* Keccak-p *is symmetric.*

This corollary can be derived from Corollary 2 of [SKC17] and Theorem 1. While Corollary 2 of [SKC17] states the need for a *self-symmetric* state, Theorem 1 suggests that a *structurally symmetric* state is sufficient. To visually illustrate this process, Fig. 2a presents the input structure of Keccak-p, highlighting the symmetric part in yellow 🟨 and white ⬜, while the green 🟩 parts signify symmetric positions for the simple derivative. It is important to emphasize that the state is only *structurally symmetric*, meaning that only the positions (and not necessarily values) of the green part need to be symmetric while computing the derivative. Consequently, we can consider all the variables in the green part as derivative variables, achieving the same MAD as ZeroSum. On the other hand, for $\mathsf{SymSum}_{\mathsf{Vec}}$, we can only utilize one part of the green variables as we require a *self-symmetric* state. A more detailed comparison of MAD is discussed below.

MAD. As mentioned earlier, for $\mathsf{SymSum}_{\mathsf{Sim}}$, a *structurally symmetric* state is sufficient. This means that we have up to 2^{1600} possible ways to generate Keccak-p states



(a) Without Linearization (b) 1-Round Linearization

Fig. 2: Keccak-p state for $m-$fold simple derivatives

while maintaining the symmetric structure. Based on this observation, we can successfully apply the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher to round-reduced Keccak-p up to 10 rounds, with a complexity of 2^{1024}. In comparison, the complexity of the ZeroSum distinguisher is 2^{1025}. On the other hand, the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher has an effective MAD of 800. This enables us to penetrate up to *only* 9 rounds with the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher.

5.1.1 Extending SimSum Distinguisher on Keccak-*p* using Linearization

In this section, we employ the concept of 1-round linearization of Keccak-p with $\mathsf{SymSum}_{\mathsf{Sim}}$. The linearization technique was first introduced by Dinur *et al.* in [DMP+15] and subsequently formalized by Guo *et al.* in [GLS16]. For the sake of completeness, the idea of linearization is described in Appendix A. Here we proceed to discuss $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher along with linearization technique. Suryawanshi *et al.* improved $\mathsf{SymSum}_{\mathsf{Vec}}$ with a 1-round linearization in [SSS20]. In the case of $\mathsf{SymSum}_{\mathsf{Vec}}$, the linearization process needs additional constraints to maintain *self-symmetric* state. Specifically, the first 32 slices must be identical to the other 32 slices. However, for $\mathsf{SymSum}_{\mathsf{Sim}}$, no further constraints are needed in addition to the linearization requirements.

Fig. 2b depicts an input state to achieve linearization in Keccak-p with the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher. The yellow 🟨 and white ⬜ colors symbolize symmetry, while the green 🟩 cells indicate variables capable of assuming all possible values. In other words, these green cells represent the variables on which derivatives are computed. The purple 🟪 cells, on the other hand, are responsible for handling the θ operation and maintaining column parity for achieving linearization.

MAD. When employing a 1-round linearization technique, the effective MAD for Keccak-p is 512, as stated in [GLS16]. In the case of $\mathsf{SymSum}_{\mathsf{Sim}}$, we are able to utilize all 512 derivative variables, while $\mathsf{SymSum}_{\mathsf{Vec}}$ restricts us to a maximum of 256 variables.

By utilizing the available MAD, we can distinguish up to 10 rounds in the forward direction with a complexity of 2^{512}. The ZeroSum distinguisher penetrates an equivalent number of rounds with a complexity of 2^{513}, which is greater than MAD. Therefore, using 1-round linearization, ZeroSum can *only* distinguish up to 9 rounds. Without linearization, ZeroSum can distinguish 10 rounds at a complexity of 2^{1025}, which exceeds the birthday bound. Therefore, $\mathsf{SymSum}_{\mathsf{Sim}}$ holds an advantage over ZeroSum by a factor of 2^{513}. On the other hand, the $\mathsf{SymSum}_{\mathsf{Vec}}$ distinguisher distinguishes up to 9 rounds with the complexity 2^{512}.

5.1.2 Inside-out Approach

The inside-out technique was first introduced by Aumasson and Meier for the Keccak-p permutation in the context of a ZeroSum distinguisher [AM09a]. This technique can also be applied to the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher. In our approach, we utilize the inside-out technique with linearization of one forward round and one backward round. In the forward direction of the Keccak-p permutation, we employ $\mathsf{SymSum}_{\mathsf{Sim}}$, while in the backward direction, we compute ZeroSum (for details, see section 5 in [GLS16]).

Utilizing the available MAD, we can successfully differentiate up to $(m + n + 2)$ rounds. Here, m and n represent the maximum rounds we can distinguish in the forward and backward directions respectively. The algebraic degrees of Keccak-p and its inverse are 2 and 3, respectively. Consequently, we can distinguish up to 6 rounds $(n = 5)$ in the backward direction and up to 10 rounds $(m = 9)$ in the forward direction. Therefore, employing a complexity of 2^{512}, we can distinguish up to 16 rounds giving us the first better than birthday-bound distinguisher for 16-rounds.

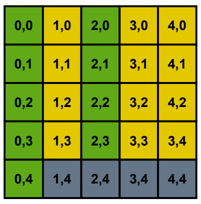
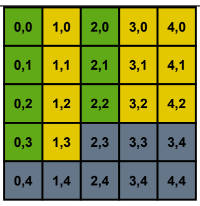
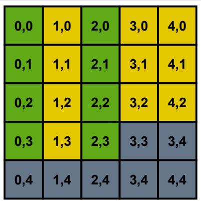
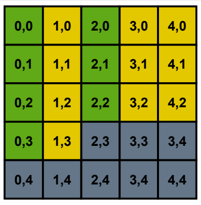
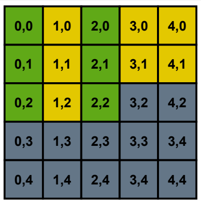
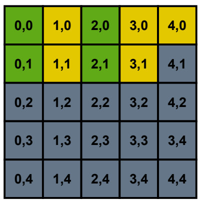
5.1.3 $\mathsf{SymSum}_{\mathsf{Sim}}$ Distinguisher on SHA-3

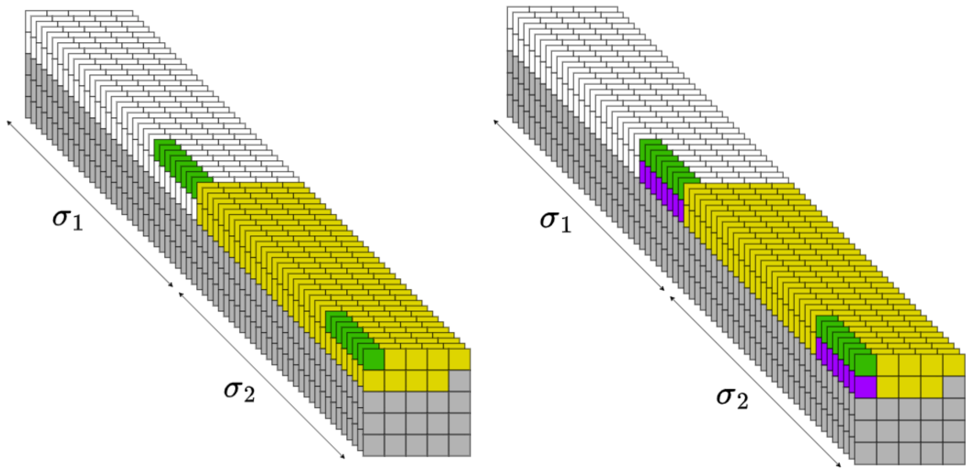
The $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguishers discussed for the Keccak-p construction can also be applied in the context of SHA-3. However, when considering the MAD, we need to account for the constraint imposed by the capacity part of the hash function. Since the adversary only has access to the rate part, not all available variables can be used for derivative computation. To visualize this process refer Fig. 3a, which illustrates a representative state of SHA-3-512. The capacity part is depicted in grey ▨, indicating that we cannot access it.

To provide a comparison in terms of the MAD, complexity, and the maximum number of rounds that can be reached, one can refer Table 1, which summarizes the results for $\mathsf{SymSum}_{\mathsf{Sim}}$, $\mathsf{SymSum}_{\mathsf{Vec}}$, and ZeroSum. Additionally, we can apply $\mathsf{SymSum}_{\mathsf{Sim}}$ with a 1-round linearization approach to SHA-3. Table 2 provides the necessary constraints and the MAD for this scenario.

Exponential Advantage Over ZeroSum. It is important to highlight that for some variants of SHA-3, the ZeroSum distinguisher is not applicable using linearization for certain rounds due to limited MAD. Taking SHA-3-384 as an example, $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguishes with a complexity of 2^{256} for 10 rounds using 1-round linearization and the χ^{-1} technique. On the other hand, ZeroSum requires 2^{257}, which is more than the MAD. As a result, ZeroSum only applies when utilizing the χ^{-1} technique, which leads to a complexity of 2^{513}. Consequently, $\mathsf{SymSum}_{\mathsf{Sim}}$ clearly has an advantage over

Table 2: We present one potential slice configuration along with its associated conditions for 1-round linearization and the corresponding MAD for various SHA-3 variants. In the diagram, the colors represent specific elements: green 🟩 for variables, gray ▨ for capacity, and yellow 🟨 for fixed lane [SSS20].

Variant	Slice Configuration	Restrictions on variables	MAD
SHAKE-128		$A[0,4] = \alpha_1 \oplus \sum_{i=0}^{3} A[0,i]$ $A[2,3] = \alpha_2 \oplus \sum_{i=0}^{2} A[2,i]$	2^{448}
SHAKE-256		$A[0,3] = \alpha_1 \oplus \sum_{i=0}^{2} A[0,i]$ $A[2,2] = \alpha_2 \oplus \sum_{i=0}^{1} A[2,i]$	2^{320}
SHA-3-224		$A[0,3] = \alpha_1 \oplus \sum_{i=0}^{2} A[0,i]$ $A[2,3] = \alpha_2 \oplus \sum_{i=0}^{2} A[2,i]$	2^{384}
SHA-3-256		$A[0,3] = \alpha_1 \oplus \sum_{i=0}^{2} A[0,i]$ $A[2,3] = \alpha_2 \oplus \sum_{i=0}^{2} A[2,i]$	2^{320}
SHA-3-384		$A[0,2] = \alpha_1 \oplus \sum_{i=0}^{1} A[0,i]$ $A[2,2] = \alpha_2 \oplus \sum_{i=0}^{1} A[2,i]$	2^{256}
SHA-3-512		$A[0,1] = \alpha_1 \oplus A[0,0]$ $A[2,1] = \alpha_2 \oplus A[2,0]$	2^{128}



(a) Without Linearization (b) With Linearisation

Fig. 3: SHA-3-512 state for 16^{th}−fold Simple Derivative

ZeroSum by a factor of 2^{257}. Similarly, for SHA-3-512, $\text{SymSum}_{\text{Sim}}$ holds an advantage over ZeroSum by a factor of 2^{129}.

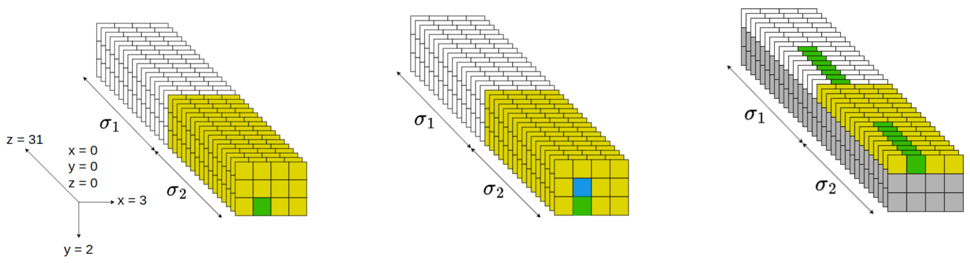
5.2 $\text{SymSum}_{\text{Sim}}$ Distinguisher on Xoodoo

Xoodoo has a different ordering of sub-functions in the SPN round structure, specifically the round constant for Xoodoo is added before the non-linear operation. Its effect on $\text{SymSum}_{\text{Vec}}$ in the form of a factor two improvement over ZeroSum has already been discussed in Proposition 1 in [SS23]. Note that the same improvement applies to $\text{SymSum}_{\text{Sim}}$ and hence the complexity of the distinguisher is same for both $\text{SymSum}_{\text{Vec}}$ and $\text{SymSum}_{\text{Sim}}$. However, for $\text{SymSum}_{\text{Sim}}$ we are able to double the MAD thereby reaching previously unreachable rounds making it a more powerful tool in analyzing the security of Xoodoo. Fig. 4a visually illustrates a representative state for applying $\text{SymSum}_{\text{Sim}}$ on Xoodoo.

MAD. The state-size of Xoodoo is 384. Thus in the case of $\text{SymSum}_{\text{Sim}}$ distinguisher we can utilize the full state and get effective MAD up to 384. This implies that the $\text{SymSum}_{\text{Sim}}$ distinguisher can be effectively applied to round-reduced Xoodoo up to 8 rounds, with complexity of 2^{256}. On the other hand, the $\text{SymSum}_{\text{Vec}}$ distinguisher, which maintains symmetry on both values and structure, can be employed on round-reduced Xoodoo up to 7 rounds only.

5.2.1 Extending $\text{SymSum}_{\text{Sim}}$ Distinguisher on Xoodoo using Linearization

The application of the linearization technique with $\text{SymSum}_{\text{Vec}}$ was initially introduced for Xoodoo in [SS23]. In the case of the $\text{SymSum}_{\text{Sim}}$ distinguisher, the main result is



(a) Xoodoo state without linearisation) (b) Xoodoo state with linearisation (c) Xoodyak-Hash state without linearisation

Fig. 4: State with $16^{th}-$fold simple derivative

similar to the linearization technique used for Keccak-p. To illustrate this, refer to Fig. 4b, which provides a visual representation of the input state for the linearization of Xoodoo for the $\mathsf{SymSum_{Sim}}$ distinguisher. The forward linearization structure can also be applied for backward linearization while maintaining the structural symmetry. The degree of the forward and backward round functions for Xoodoo are both two. Consequently, symmetry can be preserved by employing the same structure when calculating the d-fold simple derivative in both the forward and backward directions.

MAD. With linearized Xoodoo, 2^{32} potential states are possible, aligning with the $\mathsf{SymSum_{Sim}}$ count. This enables us to analyze 6 rounds in the forward direction with complexity 2^{32}. Similarly, for inverse, we are able to distinguish up to 6 rounds. For $\mathsf{SymSum_{Vec}}$, MAD is limited to 16 and $\mathsf{SymSum_{Vec}}$ can target *only* up to 5 rounds using a complexity of 2^{16}.

5.3 Extending to Full Rounds Using Inside-out Approach

We extend the above $\mathsf{SymSum_{Sim}}$ distinguishers to the best full-round distinguisher on Xoodoo using the inside-out approach. We start in the middle of the Xoodoo permutation and compute the degree outwards. There are 12 rounds in Xoodoo and we use 1 round linearization in the forward and the backward directions. Hence, with a complexity of 2^{32}, we get a full-round distinguisher of Xoodoo. We consider an affine subspace ν of dimension 32 and for all 2^{32} possible intermediate states we compute the outputs. Suppose that we consider the state after r_1 rounds for an r-round Xoodoo permutation where $r_1 + r_2 = r$. For all these 2^{32}, we compute Xoodoo^{r_2} and Xoodoo^{-r_1} and we get $\mathsf{SymSum_{Sim}}$ on both outputs. The idea is depicted below:

$$\begin{tikzcd}\mathsf{SymSum_{Sim}} & \nu \arrow[l, "\mathsf{Xoodoo}^{-r_1}"'] \arrow[r, "\mathsf{Xoodoo}^{r_2}"] & \mathsf{SymSum_{Sim}}\end{tikzcd}$$

It is important to note a significant difference between the inside-out approach applied to Keccak-p and Xoodoo. For Keccak-p, we obtain ZeroSum in the backward direction and $\mathsf{SymSum_{Sim}}$ in the forward direction. However, for Xoodoo, we obtain the $\mathsf{SymSum_{Sim}}$ distinguisher in both the forward and backward directions.

5.4 $\mathsf{SymSum_{Sim}}$ Distinguisher on Xoodyak-Hash

In this section, we will demonstrate the usage of the $\mathsf{SymSum_{Sim}}$ distinguisher for Xoodoo on Xoodyak-Hash. The initial state for $\mathsf{SymSum_{Sim}}$ of Xoodyak-Hash is shown in Fig. 4c. In the capacity portion of Xoodyak-Hash, there is a domain separator, which we have no control over. However, this does not pose a problem for us. Assuming that the state begins with a structurally symmetric state, the constant is introduced to create the actual input for Xoodyak-Hash. Therefore, the symmetry property remains preserved when computing the d^{th}-fold simple derivative of Xoodyak-Hash, especially with structurally symmetric input states.

MAD. The rate part of Xoodyak-Hash is 128 bits. Thus, we have up to 2^{128} possible ways to generate Xoodyak-Hash states while preserving the symmetric structure. As a result, the highest order derivative that can be computed is the 128^{th} order. However, due to *birthday bound*, we can distinguish 128 bit hash value with 2^{64} complexity with high probability. Hence, we restrict the comparisons of our attack to the complexity of this generic attack. With 2^{64} complexity we can distinguish upto 6 rounds which is same for the $\mathsf{SymSum_{Vec}}$ distinguisher. While, with ZeroSum distinguisher we can distinguish 6 rounds with complexity 2^{65}.

5.5 $\mathsf{SymSum_{Sim}}$ Distinguisher on Bash-f

Our final application of $\mathsf{SymSum_{Sim}}$ distinguisher is on the Bash-f permutation. Corollary 2 clarifies how the symmetry property is maintained for the $d-$fold simple derivative of Bash-f when computed with structurally symmetric input states. However, before we delve into the specifics, let us understand how the Bash-f function is structured. The Bash-f function's structure can be expressed as $\mathcal{C} \circ \mathcal{L} \circ \mathcal{N} \circ \mathcal{L}$, where round constants are added after the non-linear and linear operations. The following theorem examines the maximum degree of a monomial influenced by the round constant.

Theorem 2. *In a Bash-f permutation consisting of n_r rounds, the highest degree of a monomial that includes a round constant is $d^\circ F^{n_r} - 2$.*

Proof. The basic idea of this proof is similar to Keccak. We provide the proof in Appendix C for the sake of completeness. □

The theorem establishes that the higher degree terms of the Bash-f remain unaffected by the subsequent round constant addition. This insight leads to the following corollary for Bash-f:

Corollary 2. *When the $(d^\circ \mathsf{Bash}{-}f)-$fold simple derivative of Bash-f is evaluated using exclusively structurally-symmetric input structures and the derivative is taken at a symmetric point, the property of symmetry will remain preserved.*

MAD. With 2^{1536} possible ways to generate Bash-f states while maintaining their symmetric structure, the highest order derivative that can be computed is the 1536^{th} order. Consequently, the $\mathsf{SymSum_{Sim}}$ distinguisher can be applied to analyze round-reduced Bash-f up to 10 rounds with a complexity of 2^{1024}.

$\mathsf{SymSum_{Sim}}$ Distinguisher on Bash-f using Inside-Out technique

The structure of the inverse function of Bash-f is $\mathcal{L}\circ\mathcal{N}\circ\mathcal{L}\circ\mathcal{C}$, which can be represented as $\mathcal{N}^{'}\circ\mathcal{L}\circ\mathcal{C}$, where $\mathcal{N}^{'}=\mathcal{L}\circ\mathcal{N}$. Suryawanshi *et al.* analyzed different types of SPN functions and studied their behavior in the context of $\mathsf{SymSum_{Vec}}$ in [SS23]. The Bash-f belongs to Type-CLN of their classification. Notably, since the algebraic degree of the inverse Bash-f function is the same as that of Bash-f, using Lemma 2 of [SS23], Corollary 2 is also applicable to the inverse function of Bash-f.

The $\mathsf{SymSum_{Sim}}$ distinguisher can effectively analyze the inverse Bash-f function in the backward direction for up to 10 rounds, with a complexity of 2^{1024}. Additionally, with the same complexity, $\mathsf{SymSum_{Sim}}$ can be applied in the 10 rounds in the forward direction. Consequently, using the inside-out technique, with a complexity of 2^{1024}, the $\mathsf{SymSum_{Sim}}$ can be effectively employed for analyzing up to 20 (out of 24) rounds.

5.6 $\mathsf{SymSum_{Sim}}$ Distinguisher on Bash

The $\mathsf{SymSum_{Sim}}$ distinguisher of Bash-f can be applied to Bash. In this case we have to take the rate part into account and the effective MAD will be decided based on the rate part. In the capacity part of Bash state we have the encoding of a fixed constant value $l/4$ and the size of this capacity part is $4l$ bits where l is the security level and $l\in\{16,32,48,...,256\}$. Thus the rate is $1536-4l$. Fig. 5 provides a pictorial depiction of how the $\mathsf{SymSum_{Sim}}$ technique was applied to Bash. The green 🟩 positions represent variables that can take all possible values. Even though the variables' values may not be symmetrical, their positions in the state show symmetry. The yellow 🟨 and white ⬜ positions represent symmetry. The red 🟥 lane represents the encoding of the fixed constant value.

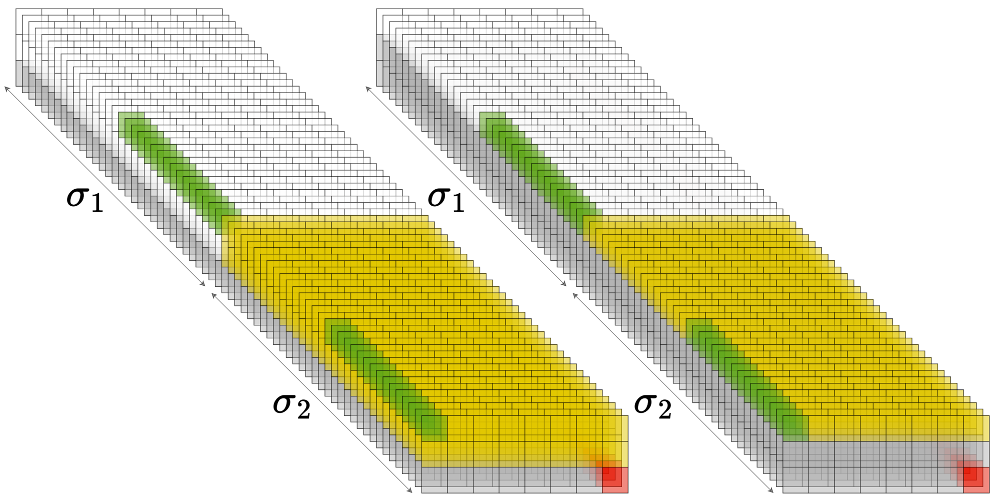


Fig. 5: Structurally symmetric states pertaining to Bash for different values of l: (left) $l=128$, (right) $l=256$.

MAD. As the rate of the hash function varies based on the value of l, the MAD for $\mathsf{SymSum}_{\mathsf{Sim}}$ also varies. l is the security level of Bash where the hash output is $2l$ bits. Considering security concerns, hash sizes smaller than 160 are not deemed secure. Therefore, our attention is directed toward hash sizes 256 and 512, corresponding to rate sizes 1022 and 510, respectively with minimum padding (2-bits).

In the case of Bash-256, the MAD is 1022 and consequently the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher can effectively analyze up to 9 rounds with complexity 2^{512}. Note that in this case $\mathsf{SymSum}_{\mathsf{Vec}}$ can also reach the same number of rounds with the same complexity. While ZeroSum can reach the same number of rounds, it requires a complexity of 2^{513}.

Similarly, for Bash-512, the MAD is 2^{510}. Thus, the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher can reach 8 rounds with a complexity of 2^{256}. Again a similar number of rounds can be reached by $\mathsf{SymSum}_{\mathsf{Vec}}$ and ZeroSum with complexity 2^{256} and 2^{257} respectively.

6 Addressing Challenges in Two-Round Linearization in $\mathsf{SymSum}_{\mathsf{Vec}}/\mathsf{SymSum}_{\mathsf{Sim}}$

In this section, we will revisit the concept of 2-round linearization and its compatibility with $\mathsf{SymSum}_{\mathsf{Vec}}/\mathsf{SymSum}_{\mathsf{Sim}}$ on SHA-3(Keccak). Previously in section 5.1.1, we discussed the basic idea of linearization, which is to choose the input state in a way that retains linearity after one round of the Keccak round function. The technique of 2-round linearization was originally proposed by Guo *et al.* in [GLS16]. When aiming to maintain linearity for two rounds, it is vital to consider the θ, ρ, π, and χ mappings of the Keccak function. To achieve linearity in the first round, we rearrange the input variables such that, in the state before the first round's χ operation, there are no adjacent variables in each row, as already discussed in the case of 1-round linearization. Additionally, it is crucial to manage the second round's θ operation effectively to prevent propagation of variables. A pictorial representation of one such arrangement is given in Fig. 6, where constant values are set to either 0 or 1 based on specific rules.

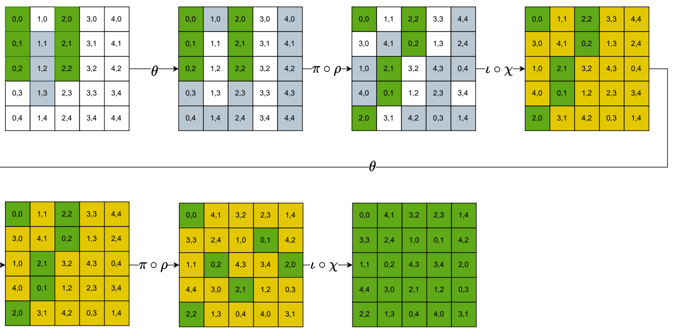


Fig. 6: 2-round linearization with MAD up to 54. Green 🟩: degree is 1, Yellow 🟨: degree is 0 (value can be either 0 or 1), White ⬜: all 0 lane, light gray ▨: all 1 lane

For the arrangement given in Fig. 6, the variables must meet the following conditions to handle the first round's θ operation.

$$\begin{aligned} A[0,0] \oplus A[0,1] \oplus A[0,2] &= 0xff\cdots f \\ A[2,0] \oplus A[2,1] \oplus A[2,2] &= 0x00\cdots 0 \end{aligned} \tag{1}$$

Similarly, the management of the second round's θ operation is necessary in order to achieve linearization in the second round. This involves careful handling of the variables' locations at the input of the second round, as the ρ and π operations can cause potential changes. The linearity after the second round is ensured by the following conditions.

$$\begin{aligned} A[2,0]_{\lll 62} &= A[0,0] \oplus A[2,2]_{\lll 43} \\ A[2,1]_{\lll 6} &= A[0,1]_{\lll 36} \\ A[2,2]_{\lll 43} &= A[0,2]_{\lll 3} \end{aligned} \tag{2}$$

Therefore, for the given arrangement in Fig. 6, there are a total of 6×64 variables and 5×64 linearly independent equations. As a result, the number of free variables comes down to 64. It is possible to rearrange the equations in a manner that allows all the variables in lane $(0,0)$ to become free variables.

Suryawanshi et al. [SSS20] proposed $\mathsf{SymSum}_{\mathsf{Vec}}$ with 2-round linearization. We present evidence to disprove the claim made in Corollary 1 of [SSS20]. The fundamental concept behind $\mathsf{SymSum}_{\mathsf{Vec}}/\mathsf{SymSum}_{\mathsf{Sim}}$ is the differentiation between two types of monomial categories: those that are unaffected by round constants (Type-I) and those that depend on round constants (Type-II). However, our findings indicate that it is not possible to achieve both 2-round linearization and $\mathsf{SymSum}_{\mathsf{Vec}}$ ($\mathsf{SymSum}_{\mathsf{Sim}}$) simultaneously. By applying the condition of 2-round linearization, we observe that the state attains a maximum algebraic degree of 1 after 2 rounds. Nevertheless, we provide evidence that certain bits of the state, after undergoing 2 rounds, still contain monomials of Type-II nature in their polynomial representation. The symmetry property is not maintained in those specific bit positions, preventing us from achieving $\mathsf{SymSum}_{\mathsf{Vec}}$ ($\mathsf{SymSum}_{\mathsf{Sim}}$). Suppose that c_1 is one non-zero bit of the round constant of first round. To demonstrate the interaction between c_1 and the variables through multiplication, we examine the state before the second round's χ operation. We note that c_1 is added to lane $(0,0)$ after the first round's χ operation. As a result, in the subsequent operation, namely second round's θ operation, it affects two columns. Thus, before the second round's χ operation, there are at least two consecutive positions that contain both c_1 and a variable. Consequently, after the χ operation, they multiply with each other and form a Type-II monomial. To validate this, we have performed two rounds of Keccak using SageMath and have confirmed that there are 6 bits in the output of the second round that contain Type-II monomials for the initial state provided in Fig. 6. In the following, we propose a novel approach to handle this problem.

$\mathsf{SymSum}_{\mathsf{Sim}}$ meets 2-round linearization. To address this challenge, it becomes essential to effectively handle the round constant introduced in the first round. Upon thorough analysis of the algebraic structure resulting from two rounds, we have identified 6 bits that are affected by the round constant of the first round and subsequently

multiply with variables in the second round. As a result, the polynomial representation of these six output bits after the second round can be expressed as $c_1 f_i + g_i$, where f_i and g_i are linear polynomials over the variables $A[0,0]$.

To eliminate the influence of these constants, we introduce an additional set of six equations in the form of $f_i = 0$, in addition to the existing 5×64 equations given above. The complete set of equations can be found in Table B1 (equations 1-6). It is important to note that the first equation is linearly dependent on the other 5 equations, so adding these last 5 equations is sufficient to address the issue. Moreover, these additional equations do not affect the 2-round linearization process. By incorporating these constraints, we can still achieve the desired 2-round linearization while effectively canceling out the influence of the round constant at the output of second round.

However, adding these equations has a crucial side effect: it reduces the number of free variables to $64 - 5 = 59$, and the free variables are no longer in symmetric positions. To maintain symmetry, an additional set of 6 constraints is required. These constraints ensure that the symmetric positions remain unchanged. To construct the symmetric polynomial f_i' corresponding to f_i, we follow these steps: we initially set $f_i' = 0$. For each variable $A[0,0,j]$ that is present in the ANF of f_i, we update f_i' by adding $A[0,0,(j+32)\%64]$.

Thus, we add total of 12 additional equations, as specified in Table B1. However, it is worth noting that only 10 of these equations (equations 2-6 and 8-12) are necessary to achieve the desired outcome, given their dependencies. As a result, we obtain the number of free variables as $64 - 10 = 54$, with all the free variables located in the lane $(0,0)$ at positions $5, 6, ..., 31$ and $37, 38, ..., 63$ maintaining the desired symmetry. This gives us reachable MAD upto 54 with 2-round linearization. We describe a similar linear structure with MAD increased up to 114 in next subsection.

6.1 Increase MAD upto 114 in 2-round Linear Structure

Let us consider an initial state arrangement that increases the MAD up to 114 along with a 2-round linearization. To illustrate this, we can examine the arrangement shown in Fig. 7.

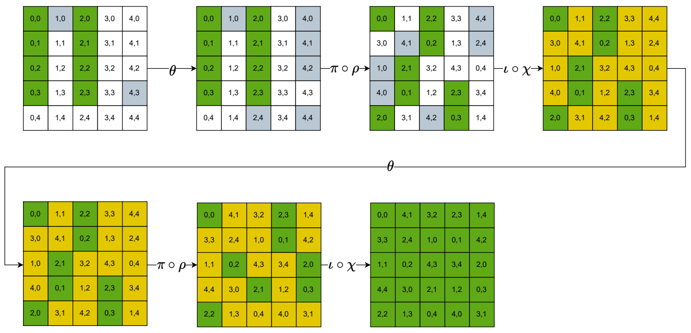
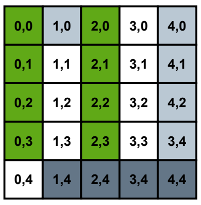
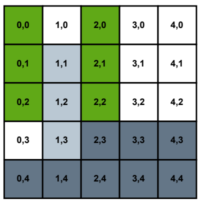
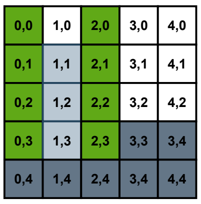
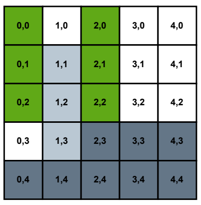
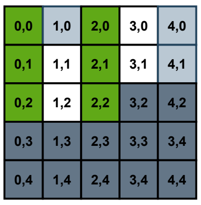


Fig. 7: 2-round linearization with MAD up to 114

Table 3: We present one potential slice configuration along with its associated conditions for 2-round linearization and the corresponding MAD for various SHA-3 variants. We also present number of equations needed to handle the round constant. The colors represent specific elements: green 🟩 for variables, white ⬜ for all 0 lane, light gray ▨ for all 1 lane and dark gray ⬛ for the capacity part. Note that this strategy is not applicable for SHA-3-512

Variant	Slice Configuration	Restrictions on variables	#Restrictions to handle constant	MAD
SHAKE-128		$A[0,0] \oplus A[0,1] \oplus A[0,2] = 0xff\cdots f$ $A[2,0] \oplus A[2,1] \oplus A[2,2] = 0x00\cdots 0$ $A[2,0]_{\lll 62} = A[0,0] \oplus A[2,2]_{\lll 43}$ $A[2,1]_{\lll 6} = A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15}$ $A[2,2]_{\lll 43} = A[0,2]_{\lll 3}$ $A[2,3]_{\lll 15} = A[0,3]_{\lll 41} \oplus A[2,0]_{\lll 62}$	14	2^{114}
SHAKE-256		$A[0,0] \oplus A[0,1] \oplus A[0,2] = 0xff\cdots f$ $A[2,0] \oplus A[2,1] \oplus A[2,2] = 0x00\cdots 0$ $A[2,0]_{\lll 62} = A[0,0] \oplus A[2,2]_{\lll 43}$ $A[2,1]_{\lll 6} = A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15}$ $A[2,2]_{\lll 43} = A[0,2]_{\lll 3}$	10	2^{54}
SHA-3-224		$A[0,0] \oplus A[0,1] \oplus A[0,2] = 0xff\cdots f$ $A[2,0] \oplus A[2,1] \oplus A[2,2] = 0x00\cdots 0$ $A[2,0]_{\lll 62} = A[0,0] \oplus A[2,2]_{\lll 43}$ $A[2,1]_{\lll 6} = A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15}$ $A[2,2]_{\lll 43} = A[0,2]_{\lll 3}$ $A[2,3]_{\lll 15} = A[0,3]_{\lll 41} \oplus A[2,0]_{\lll 62}$	10	2^{114}
SHA-3-256		$A[0,0] \oplus A[0,1] \oplus A[0,2] = 0xff\cdots f$ $A[2,0] \oplus A[2,1] \oplus A[2,2] = 0x00\cdots 0$ $A[2,0]_{\lll 62} = A[0,0] \oplus A[2,2]_{\lll 43}$ $A[2,1]_{\lll 6} = A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15}$ $A[2,2]_{\lll 43} = A[0,2]_{\lll 3}$	10	2^{54}
SHA-3-384		$A[0,0] \oplus A[0,1] \oplus A[0,2] = 0xff\cdots f$ $A[2,0] \oplus A[2,1] \oplus A[2,2] = 0x00\cdots 0$ $A[2,0]_{\lll 62} = A[0,0] \oplus A[2,2]_{\lll 43}$ $A[2,1]_{\lll 6} = A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15}$ $A[2,2]_{\lll 43} = A[0,2]_{\lll 3}$	10	2^{54}

To keep the first round linear, the variables must meet these conditions

$$\begin{aligned} A[0,0] \oplus A[0,1] \oplus A[0,2] \oplus A[0,3] &= \texttt{0xff...f} \\ A[2,0] \oplus A[2,1] \oplus A[2,2] \oplus A[2,3] &= \texttt{0xff...f} \end{aligned} \tag{3}$$

Similarly, the following conditions guarantee that the function maintains its linearity after the second round.

$$\begin{aligned} A[2,0]_{\lll 62} &= A[0,0] \oplus A[2,2]_{\lll 43} \\ A[2,1]_{\lll 6} &= A[0,1]_{\lll 36} \oplus A[2,3]_{\lll 15} \\ A[2,2]_{\lll 43} &= A[0,2]_{\lll 3} \\ A[2,3]_{\lll 15} &= A[0,3]_{\lll 41} \oplus A[2,0]_{\lll 62} \end{aligned} \tag{4}$$

Considering the arrangement shown in Fig. 7, we observe a total of 8×64 variables and 6×64 linearly independent equations, resulting in 128 free variables. By organizing the equations appropriately, we can have the free variables in lanes $(0,1)$ and $(0,2)$.

Analyzing the algebraic structure after 2 rounds, we find that 8 bits are influenced by the round constant of the first round. To eliminate the effect of constants, we introduce an additional set of 8 equations. Additionally, to maintain symmetry, we require 8 more equations. The detailed equations can be found in the Appendix (Table B3). Out of the 16 equations, equations $2-7$ and $9-14$ are sufficient due to their dependencies. Consequently, we achieve a MAD up to $128-14=114$. Among the 114 free variables, 64 variables are located in lane $(0,1)$, while the remaining 50 free variables are symmetrically positioned in lane $(0,2)$. Specifically, the free variables in lane $(0,2)$ correspond to symmetric locations as follows:

$$4,5,6,7,9,11...,23,25,...,31,36,37,38,39,41,43...,55,56,...,63.$$

6.2 Application on SHA-3

We apply $\mathsf{SymSum}_{\mathsf{Sim}}$ with 2-round linearization on SHA-3. To apply two round linear structures on SHA-3, we need to consider the equations along with the condition that we can access only the rate part. The detailed equations and MAD are given in Table 3. Applying $\mathsf{SymSum}_{\mathsf{Sim}}$ with 2-round linearization we distinguish 8 rounds of SHAKE-128 and SHA-3-224 with complexity 2^{64}. Also, we distinguish 7 rounds of SHAKE-256, SHA-3-256 and SHA-3-384 with complexity 2^{32}.

7 Discussion

In this work, we introduce $\mathsf{SymSum}_{\mathsf{Sim}}$, an innovative distinguisher that addresses and surpasses the limitations of $\mathsf{SymSum}_{\mathsf{Vec}}$ offering comparable advantages to ZeroSum while utilizing simpler order derivatives rather than vectorial order derivatives. $\mathsf{SymSum}_{\mathsf{Sim}}$ achieves equivalence to ZeroSum regarding MAD enabling penetration into rounds previously unreachable by $\mathsf{SymSum}_{\mathsf{Vec}}$. $\mathsf{SymSum}_{\mathsf{Sim}}$, in contrast to $\mathsf{SymSum}_{\mathsf{Vec}}$, eliminates the need for fully self-symmetric inputs by relaxing the requirement for symmetry among the variables across which derivatives are computed. We illustrate $\mathsf{SymSum}_{\mathsf{Sim}}$'s effectiveness across various SHA-3 variants, Keccak, Xoodyak-Hash, Xoodoo, Bash, and Bash-f showing a significant advantage over ZeroSum in MAD. Furthermore, we solve the challenge of even-order derivatives by combining them with vectorial derivatives, resulting in a hybrid derivative of odd order, which we denote as $\mathsf{SymSum_{Hyb}}$. Furthermore, we assert the impracticality of the two-round linearization of $\mathsf{SymSum_{Vec}}$ as proposed by Suryawanshi *et al.* We propose a solution to address this by introducing additional constraints. $\mathsf{SymSum_{Sim}}$ showcases significant performance advantages over ZeroSum in various scenarios, as illustrated in Fig. 8. Here, ZeroSum

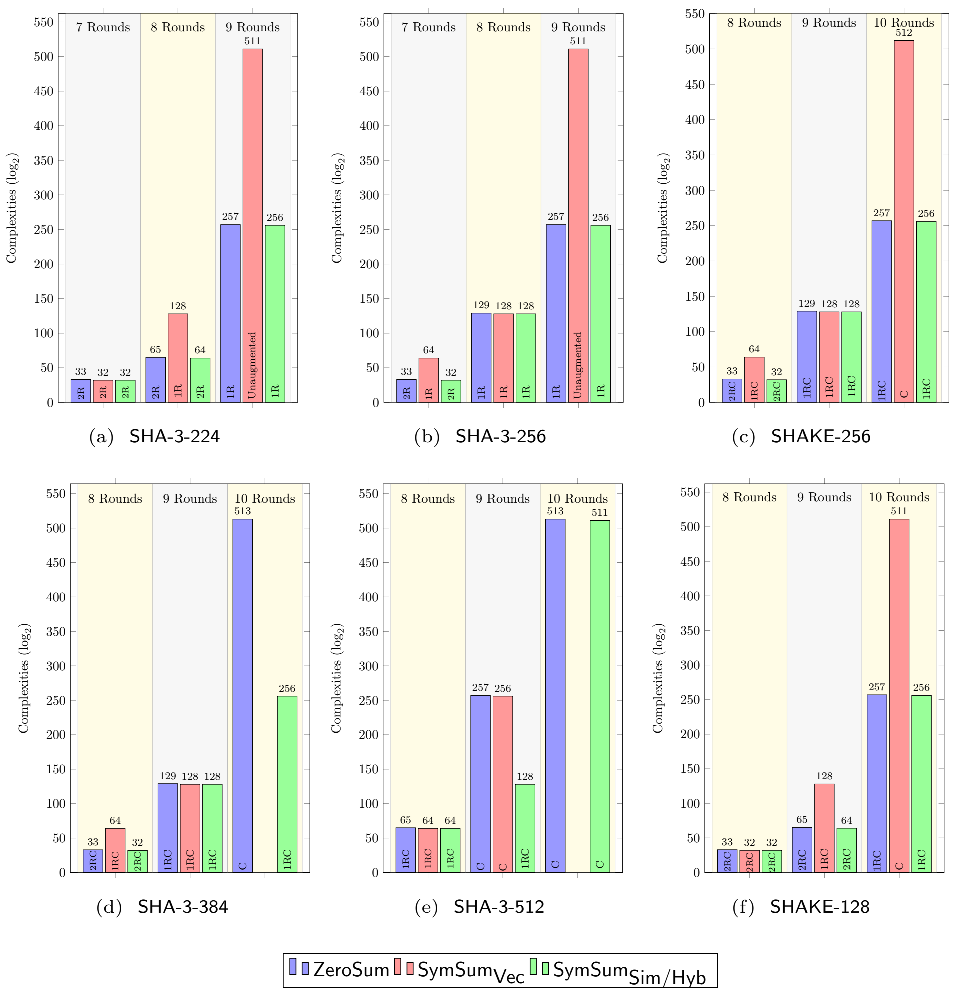


Fig. 8: Comparing the complexities (less than birthday bound) of $\mathsf{SymSum_{Sim}}/\mathsf{SymSum_{Hyb}}$ with ZeroSum and $\mathsf{SymSum_{Vec}}$ for all variants of fixed-length hash and XOFs of SHA-3. The captions inside the histogram captures the distinguishing strategies, 1RC: 1-round linearization with χ^{-1}, 2RC: 2-round linearization with χ^{-1}, C: only χ^{-1}, 1R: only 1-round linearization, 2R: only 2-round linearization.

is illustrated in blue, $\mathsf{SymSum}_{\mathsf{Vec}}$ in red and both $\mathsf{SymSum}_{\mathsf{Sim}}$ and $\mathsf{SymSum}_{\mathsf{Hyb}}$ are illustrated in green. The abbreviations 1RC, 2RC, C, 1R and 2R stand for augmentation of 1-round linearization with χ^{-1}, 2-round linearization with χ^{-1}, only χ^{-1}, only 1-round linearization and only 2-round linearization respectively.

Fig. 8d and Fig. 8e illustrates that $\mathsf{SymSum}_{\mathsf{Vec}}$ is not applicable on 10 rounds of SHA-3-384 and SHA-3-512, whereas our technique $\mathsf{SymSum}_{\mathsf{Sim}}$ not only keeps its applicability, but also outperforms ZeroSum by factors of 2^{257} and 4. It can be noted that $\mathsf{SymSum}_{\mathsf{Sim}}$ beats ZeroSum by a factor of 2^{129} for a lower round of the SHA-3-512, specifically for 9 rounds. For SHA-3-224, $\mathsf{SymSum}_{\mathsf{Sim}}$ outperforms $\mathsf{SymSum}_{\mathsf{Vec}}$ over 8 and 9 rounds with almost 100% improvement. Similar improvements can be seen across various rounds of SHA-3-256, SHAKE-128 and SHAKE-256 Also, $\mathsf{SymSum}_{\mathsf{Sim}}$ outperforms ZeroSum with at least a factor of 2 and at most factor of 2^{257}.

8 Experimental Verification

To validate the presence of the distinguishers discussed in the paper, we performed a series of experiments. We experimentally confirmed the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher for up to 7 rounds of SHA-3-224, SHA-3-256, SHA-3-384, SHAKE-128, and SHAKE-256. Additionally, we also verified the $\mathsf{SymSum}_{\mathsf{Sim}}$ distinguisher for up to 12 rounds (6 rounds forward + 6 rounds backward) of Xoodoo. Verification code is publicly available at the following link.

In this section, we first present a demonstration of a distinguisher on a 7-round Keccak using the 2-round linearization and $\mathsf{SymSum}_{\mathsf{Sim}}$ techniques. To initiate the attack, we begin with the following state, as shown in Table 4.

Table 4: Initial Symmetric input state of Keccak

C491E2A0C491E2A0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	FFFFFFFFFFFFFFFF	0000000000000000	0000000000000000	0000000000000000
0000000000000000	FFFFFFFFFFFFFFFF	0000000000000000	0000000000000000	0000000000000000
0000000000000000	FFFFFFFFFFFFFFFF	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

By employing the 2-round linearization, each bit at the output after 7 rounds exhibits a degree of 32. Hence, it suffices to consider simple derivatives with respect to 32 variables located in symmetric positions. In practice, we have the flexibility to select these variables from any symmetric positions from the set of 54 free variables ranging from $5, ..., 31$ and $37, ..., 63$ of lane $(0, 0)$. This differs from the constraints imposed by $\mathsf{SymSum}_{\mathsf{Vec}}$, where all 32 variables must be chosen from one of the two symmetric parts. For our analysis, we have selected the following bit positions as the derivative variables.

$$8, 9, .., 23, \ \ 40, 41, ..., 55.$$

The derivative variables are denoted with * in Table 5. By altering the values of the derivative variables, we generate 2^{32} distinct values for lane $(0, 0)$. In order to achieve 2-round linearization, the dependent variables located in lanes

Table 5: Derivative state of Keccak

C4****A!C4****A!	0000000000000000	!!!!!!!!!!!!!!!!	0000000000000000	0000000000000000
!!!!!!!!!!!!!!!!	FFFFFFFFFFFFFFFF	!!!!!!!!!!!!!!!!	0000000000000000	0000000000000000
!!!!!!!!!!!!!!!!	FFFFFFFFFFFFFFFF	!!!!!!!!!!!!!!!!	0000000000000000	0000000000000000
0000000000000000	FFFFFFFFFFFFFFFF	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

`In first row A is actually 101! instead of "1010"`

$(0,1),(0,2),(2,0),(2,1),(2,2)$, as well as at the bits $0,...,4$ and $32,...,36$ in lane $(0,0)$, are calculated based on the equations presented in equations (1) and (2) above, and in the Table B1 from the free variables. The dependent variables are marked with an ! in Table 5. By modifying the $*$ values of the input base message, we generate 2^{32} individual inputs and compute the XOR-sum of the outputs after 7 rounds of the Keccak function. The symmetric output sum is provided in Table 6. The source codes of the verification is given in supplementary material.

Table 6: The XOR-sum of the outputs from 2^{32} iterations of a 7-round Keccak

631E8F2C631E8F2C	5423691E5423691E	1997B8E11997B8E1	7514167B7514167B	4D1174B24D1174B2
33BAD16533BAD165	429B50D9429B50D9	17AD101617AD1016	EF1686C3EF1686C3	9F14CF929F14CF92
C217B830C217B830	8C13A20B8C13A20B	6E8638706E863870	5750B4755750B475	3881C3ED3881C3ED
813E31CA813E31CA	DC30F993DC30F993	11D0F2E411D0F2E4	5758F72E5758F72E	5C8865535C886553
C5F02DFFC5F02DFF	CC3015AECC3015AE	4ABCCE1F4ABCCE1F	320D0293320D0293	70B28FF370B28FF3

Practical verification of full-round distinguisher on Xoodoo. We conducted a similar experiment for 12 rounds of Xoodoo using inside-out technique. To do this, we use 1-round forward and 1-round backward linearization. When employing 1-round linearization in both the directions for Xoodoo, the degree after 6 rounds in both directions will become 32. The starting middle state is presented in Table 7.

Table 7: Initial state of Xoodoo

Symmetric input state				Derivative state			
62446244	75DA75DA	F90DF90D	DFB5DFB5	62446244	75DA75DA	F90DF90D	DFB5DFB5
4C9C4C9C	1D851D85	E40FE40F	B95BB95B	4C9C4C9C	********	E40FE40F	B95BB95B
C963C963	5F015F01	E065E065	754F754F	C963C963	!!!!!!!!	E065E065	754F754F

We select the 32 variables located at lane $(1,1)$ as the derivative variables for our analysis. The derivative variables and the dependent variables are denoted with $*$ and ! respectively in Table 7. By modifying the $*$ values of the base input state, we generate 2^{32} individual inputs where the value of ! will be same as $*$ to maintain the constraint required for linearization. The resultant XOR-sum after 6 forward and 6 backward rounds are given in Table 8. The source code of the verification is given in supplementary material.

Table 8: The XOR-sum of the outputs from 2^{32} iterations of a 6-round Xoodoo

Forward direction				Backward direction			
13AE13AE	65646564	57315731	93AD93AD	F6DFF6DF	5E725E72	57A057A0	20CE20CE
91E791E7	3C623C62	26212621	75E175E1	8F1A8F1A	23C823C8	3E433E43	ADBAADBA
DC88DC88	EE6DEE6D	B217B217	363C363C	0EC00EC0	7B417B41	0AB30AB3	8E578E57

9 Conclusion

This paper introduces a new variant of the SymSum$_{\text{Vec}}$ distinguisher, called SymSum$_{\text{Sim}}$, which overcomes the limitations of SymSum$_{\text{Vec}}$ by utilizing simple derivatives instead of vectorial derivatives. Furthermore, by utilizing simple order derivatives, SymSum$_{\text{Sim}}$ significantly improves distinguishing power while maintaining a comparable MAD to ZeroSum. Thus, by leveraging the doubling of MAD, SymSum$_{\text{Sim}}$ outperforms the conventional ZeroSum distinguisher. The feasibility of SymSum$_{\text{Sim}}$ is demonstrated through its successful application to various cryptographic primitives, including SHA-3, Xoodyak-Hash, Bash and their corresponding internal permutations. Furthermore, it combines the benefits of SymSum$_{\text{Sim}}$ with one-round linearization to penetrate more rounds in cryptographic primitives such as SHA-3 and Xoodyak-Hash. Additionally, the work highlights the impossibility of applying SymSum$_{\text{Vec}}$ or SymSum$_{\text{Sim}}$ to more than 1 round linearization, providing insights into the fundamental constraints of these techniques. We also provide an algebraic fix to regain 2-round linearization capability while augmenting both SymSum$_{\text{Sim}}$ and its predecessor SymSum$_{\text{Vec}}$. Finally, experimental verification of the proposed methods is presented to validate their effectiveness and versatility. SymSum$_{\text{Sim}}$ outperforms ZeroSum by a factor of 2^{257} and 2^{129} for 10-round SHA-3-384 and 9-round SHA-3-512 respectively while enjoying the same MAD as ZeroSum. SymSum$_{\text{Sim}}$ also reaches 16 rounds (which was previously not reachable) of Keccak with a better than birthday bound complexity of 2^{512}. Our distinguisher also gives the best result for *full-round* Xoodoo with a complexity of 2^{32}. As SymSum$_{\text{Sim}}$ overcomes the barrier of the maximum attainable derivative, it overcomes the limitation of SymSum$_{\text{Vec}}$ and thus gets almost a deterministic edge over ZeroSum on all fronts leading to a 100% improvement in almost all distinguishers while penetrating higher rounds, thereby reaching a place where *no distinguisher has gone before.*

Information on Supplementary Material:

We provide the verification codes for practically verifying the distingushers. Our code can be compiled on Linux or on Windows with Visual Studio.

References

[AM09a] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced keccak-f and for the core functions of luffa and hamsi. *rump session of Cryptographic Hardware and Embedded Systems-CHES*, 2009:67, 2009.

[AM09b] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. *rump session of Cryptographic Hardware and Embedded Systems-CHES*, 2009:67, 2009.

[AMMS16] Sergey Agievich, Vadim Marchuk, Alexander Maslau, and Vlad Semenov. Bash-f: another LRX sponge function. *IACR Cryptol. ePrint Arch.*, page 587, 2016.

[BC10] Christina Boura and Anne Canteaut. Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2010.

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and *Luffa*. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.

[BDL$^+$19] Wenquan Bi, Xiaoyang Dong, Zheng Li, Rui Zong, and Xiaoyun Wang. Milp-aided cube-attack-like cryptanalysis on keccak keyed modes. *Des. Codes Cryptogr.*, 87(6):1271–1296, 2019.

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011. http://keccak.noekeon.org/Keccak-submission-3.pdf.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[BP81] D. Bochmann and C Posthoff. *Binäre dynamische Systeme (German Edition).* Oldenbourg, 1981.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptol.*, 4(1):3–72, 1991.

[DHAK18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of xoodoo and xoofff. *IACR Trans. Symmetric Cryptol.*, 2018(4):1–38, 2018.

[DHP$^+$20] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Trans. Symmetric Cryptol.*, 2020(S1):60–87, 2020.

[DMP$^+$15] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761. Springer, 2015.

[DYS$^+$14] Ming Duan, Mohan Yang, Xiaorui Sun, Bo Zhu, and Xuejia Lai. Distinguishing properties and applications of higher order derivatives of boolean functions. *Inf. Sci.*, 271:224–235, 2014.

[GD21] Shibam Ghosh and Orr Dunkelman. Automatic search for bit-based division property. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings*, volume 12912 of *Lecture Notes in Computer Science*, pages 254–274. Springer, 2021.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, 2016.

[GS20] Lorenzo Grassi and Markus Schofnegger. Mixture integral attacks on reduced-round AES with a known/secret s-box. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, volume 12578 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2020.

[LBDW17] Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on keccak keyed modes with MILP method. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 99–127. Springer, 2017.

[PS04] Christian Posthoff and Bernd Steinbach. *Logic functions and equations. Binary models for computer science.* Springer Publishing Company, Incorporated., 2004.

[PS19] Christian Posthoff and Bernd Steinbach. *Logic functions and equations. Binary models for computer science.* Springer Publishing Company, Incorporated., 2019.

[SG18] Ling Song and Jian Guo. Cube-attack-like cryptanalysis of round-reduced keccak using MILP. *IACR Trans. Symmetric Cryptol.*, 2018(3):182–214, 2018.

[SKC17] Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. Symsum: Symmetric-sum distinguishers against round reduced SHA3. *IACR Trans. Symmetric Cryptol.*, 2017(1):240–258, 2017.

[SS23] Sahiba Suryawanshi and Dhiman Saha. Where are the constants? new insights on the role of round constant addition in the symsum distinguisher. In *(SSS 2023): The 25th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Lecture Notes in Computer Science, 2023. https://eprint.iacr.org/2023/789.

[SSS20] Sahiba Suryawanshi, Dhiman Saha, and Satyam Sachan. New results on the symsum distinguisher on round-reduced SHA3. In Abderrahmane Nitaj and Amr M. Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *Lecture Notes in Computer Science*, pages 132–151. Springer, 2020.

[Tha81] André Thayse. *Boolean calculus of differences.* Springer, 1981.

[WHG+19] SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. A practical method to recover exact superpoly in cube attack. *IACR Cryptol. ePrint Arch.*, page 259, 2019.

[YLW$^+$19] Hailun Yan, Xuejia Lai, Lei Wang, Yu Yu, and Yiran Xing. New zero-sum distinguishers on full 24-round keccak-f using the division property. *IET Inf. Secur.*, 13(5):469–478, 2019.

[ZCWW21] Zishen Zhao, Shiyao Chen, Meiqin Wang, and Wei Wang. Improved cube-attack-like cryptanalysis of reduced-round ketje-jr and keccak-mac. *Inf. Process. Lett.*, 171:106124, 2021.

Appendix A 1-round Linear Structure

The core idea of 1-round linearization is to consider restricted input state in order to handle the linear (θ) and non-linear (χ) operations within Keccak-p. This involves analyzing the Boolean expression of the χ function. The function $\chi : \mathbb{F}_2^5 \rightarrow \mathbb{F}_2^5$ is defined as

$$\begin{cases} (y_0, y_1, y_2, y_3, y_4) = \chi(x_0, x_1, x_2, x_3, x_4) \\ y_i = x_i \oplus (x_{(i+1) \ mod \ 5} \oplus 1) x_{(i+2) \ mod \ 5}, \ \text{for } i \in \{0, 1, 2, 3, 4\} \end{cases} \tag{A1}$$

Therefore, the only way for χ to increase the algebraic degree is through two neighboring bits, as indicated by the term $(x_{(i+1) \mod 5} \oplus 1) x_{(i+2) \mod 5}$. Consequently, if we ensure that for any two consecutive bits in the input, one is constant, the algebraic degree of the state bits will remain at most 1 after the χ operation. With this in mind, we set the initial state such that, just before the χ operation of the first round, there are no consecutive variables in each row of the state. To illustrate this, consider the following example shown in Fig. A1. In this example, the green 🟩 cells have an algebraic degree of exactly 1, while the orange 🟧 cells have a degree of at most 1. The yellow 🟨 cells represent constant values.

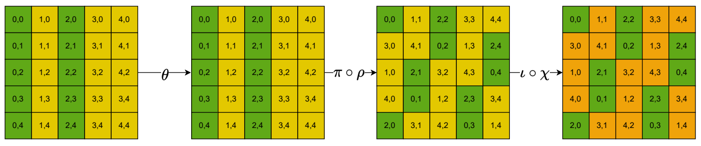


Fig. A1: Keccak State Linearization with 512 Degrees of Freedom [GLS16, SSS20]

In Fig. A1, the initial state contains variables in columns 0 and 2. At first, to prevent the spread of variables to other columns we need the column parity to be constant. We consider the equation given below to ensure constant column parity:

$$A[i, 4] = \bigoplus_{j=0}^{3} A[i, j] \oplus \alpha_i$$

where $i = 0, 2$, $j = 0, 1, 2, 3$ and α_i is arbitary constant. As a result, after the θ operation, the variables remain in columns 0 and 2, and do not spread to other columns. The subsequent operation, ρ, does not spread the variables to other lanes. Finally, due to the absence of variables in two consecutive columns, the π operation ensures that there are no consecutive variables in each row of the state, as intended. The complete propagation of variables is illustrated in Fig. A1.

Appendix B State Configuration for 1 and 2 Round Linearization

It is important to note that for the SHA-3-512 variant, 2-round linearization is not feasible due to restrictions in the lower rate part, which limit the formulation of the necessary constraints. The detailed constraints are given in the Table B1, B2, B3 and B4 for MAD 56 and 114 respectively.

Table B1: This table illustrates the additional set of expressions (set to zero) to handle the effect of round constants to ensure 2 round linearisation for $\mathsf{SymSum_{Vec}}/\mathsf{SymSum_{Sim}}$. The constraints are represented as $\mathcal{L}_j$. Here i^{th} variable of lane (0,0) is represented by $x_{0,i}$

	Equations
$\mathcal{L}_1$	$x_{0,3} + x_{0,6} + x_{0,8} + x_{0,9} + x_{0,12} + x_{0,15} + x_{0,16} + x_{0,19} + x_{0,22} + x_{0,23} + x_{0,24} + x_{0,25} + x_{0,29} + x_{0,30} + x_{0,33} + x_{0,34} + x_{0,35} + x_{0,36} + x_{0,37} + x_{0,39} + x_{0,42} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,47} + x_{0,51} + x_{0,53} + x_{0,54} + x_{0,55} + x_{0,57} + x_{0,58} + x_{0,62} + x_{0,63}$
$\mathcal{L}_2$	$x_{0,0} + x_{0,1} + x_{0,2} + x_{0,4} + x_{0,7} + x_{0,8} + x_{0,9} + x_{0,10} + x_{0,12} + x_{0,16} + x_{0,18} + x_{0,19} + x_{0,20} + x_{0,22} + x_{0,23} + x_{0,27} + x_{0,28} + x_{0,32} + x_{0,35} + x_{0,37} + x_{0,38} + x_{0,41} + x_{0,44} + x_{0,45} + x_{0,48} + x_{0,51} + x_{0,52} + x_{0,53} + x_{0,54} + x_{0,58} + x_{0,59} + x_{0,62} + x_{0,63}$
$\mathcal{L}_3$	$x_{0,0} + x_{0,1} + x_{0,5} + x_{0,6} + x_{0,9} + x_{0,10} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,15} + x_{0,18} + x_{0,19} + x_{0,20} + x_{0,21} + x_{0,23} + x_{0,27} + x_{0,29} + x_{0,30} + x_{0,31} + x_{0,33} + x_{0,34} + x_{0,38} + x_{0,39} + x_{0,43} + x_{0,46} + x_{0,48} + x_{0,49} + x_{0,52} + x_{0,55} + x_{0,56} + x_{0,59} + x_{0,62} + x_{0,63}$
$\mathcal{L}_4$	$x_{0,1} + x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} + x_{0,7} + x_{0,10} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,15} + x_{0,19} + x_{0,21} + x_{0,22} + x_{0,23} + x_{0,25} + x_{0,26} + x_{0,30} + x_{0,31} + x_{0,35} + x_{0,38} + x_{0,40} + x_{0,41} + x_{0,44} + x_{0,47} + x_{0,48} + x_{0,51} + x_{0,54} + x_{0,55} + x_{0,56} + x_{0,57} + x_{0,61} + x_{0,62}$
$\mathcal{L}_5$	$x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} + x_{0,9} + x_{0,10} + x_{0,13} + x_{0,14} + x_{0,15} + x_{0,16} + x_{0,17} + x_{0,19} + x_{0,22} + x_{0,23} + x_{0,24} + x_{0,25} + x_{0,27} + x_{0,31} + x_{0,33} + x_{0,34} + x_{0,35} + x_{0,37} + x_{0,38} + x_{0,42} + x_{0,43} + x_{0,47} + x_{0,50} + x_{0,52} + x_{0,53} + x_{0,56} + x_{0,60} + x_{0,63}$
$\mathcal{L}_6$	$x_{0,1} + x_{0,5} + x_{0,8} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,14} + x_{0,18} + x_{0,19} + x_{0,22} + x_{0,23} + x_{0,24} + x_{0,25} + x_{0,26} + x_{0,28} + x_{0,31} + x_{0,32} + x_{0,33} + x_{0,34} + x_{0,36} + x_{0,40} + x_{0,42} + x_{0,43} + x_{0,44} + x_{0,46} + x_{0,47} + x_{0,51} + x_{0,52} + x_{0,56} + x_{0,59} + x_{0,61} + x_{0,62}$

Table B2: The constraints given in Table B1 handle the round constants but not the symmetry, thus this table illustrates the additional set of expressions (set to zero) to handle the effect of round constant and maintain the symmetry to ensure 2 round linearisation for $\mathsf{SymSum}_{\mathsf{Vec}}/\mathsf{SymSum}_{\mathsf{Sim}}$. The constraints are represented as $\mathcal{L}'_j$.

	Equations
$\mathcal{L}'_1$	$x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} + x_{0,6} + x_{0,8} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,14} + x_{0,16} + x_{0,20} + x_{0,22} + x_{0,23} + x_{0,24} + x_{0,26} + x_{0,27} + x_{0,31} + x_{0,32} + x_{0,35} + x_{0,38} + x_{0,40} + x_{0,41} + x_{0,44} + x_{0,47} + x_{0,48} + x_{0,51} + x_{0,54} + x_{0,55} + x_{0,56} + x_{0,57} + x_{0,61} + x_{0,62}$
$\mathcal{L}'_2$	$x_{0,1} + x_{0,4} + x_{0,6} + x_{0,7} + x_{0,10} + x_{0,13} + x_{0,14} + x_{0,17} + x_{0,20} + x_{0,21} + x_{0,22} + x_{0,23} + x_{0,27} + x_{0,28} + x_{0,31} + x_{0,32} + x_{0,32} + x_{0,33} + x_{0,34} + x_{0,36} + x_{0,39} + x_{0,40} + x_{0,41} + x_{0,42} + x_{0,44} + x_{0,48} + x_{0,50} + x_{0,51} + x_{0,52} + x_{0,54} + x_{0,55} + x_{0,59} + x_{0,60}$
$\mathcal{L}'_3$	$x_{0,0} + x_{0,2} + x_{0,3} + x_{0,7} + x_{0,8} + x_{0,12} + x_{0,15} + x_{0,17} + x_{0,18} + x_{0,21} + x_{0,24} + x_{0,25} + x_{0,28} + x_{0,31} + x_{0,32} + x_{0,32} + x_{0,33} + x_{0,37} + x_{0,38} + x_{0,41} + x_{0,42} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,47} + x_{0,50} + x_{0,51} + x_{0,52} + x_{0,53} + x_{0,55} + x_{0,59} + x_{0,61} + x_{0,62}$
$\mathcal{L}'_4$	$x_{0,0} + x_{0,4} + x_{0,7} + x_{0,9} + x_{0,10} + x_{0,13} + x_{0,16} + x_{0,17} + x_{0,20} + x_{0,23} + x_{0,24} + x_{0,25} + x_{0,26} + x_{0,30} + x_{0,31} + x_{0,33} + x_{0,34} + x_{0,35} + x_{0,36} + x_{0,37} + x_{0,39} + x_{0,42} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,47} + x_{0,51} + x_{0,53} + x_{0,54} + x_{0,55} + x_{0,57} + x_{0,58} + x_{0,62}$
$\mathcal{L}'_5$	$x_{0,0} + x_{0,2} + x_{0,3} + x_{0,4} + x_{0,6} + x_{0,7} + x_{0,11} + x_{0,12} + x_{0,16} + x_{0,19} + x_{0,21} + x_{0,22} + x_{0,25} + x_{0,29} + x_{0,32} + x_{0,34} + x_{0,35} + x_{0,36} + x_{0,37} + x_{0,41} + x_{0,42} + x_{0,45} + x_{0,46} + x_{0,47} + x_{0,48} + x_{0,49} + x_{0,51} + x_{0,54} + x_{0,55} + x_{0,56} + x_{0,57} + x_{0,59}$
$\mathcal{L}'_6$	$x_{0,0} + x_{0,1} + x_{0,2} + x_{0,3} + x_{0,5} + x_{0,9} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,15} + x_{0,16} + x_{0,20} + x_{0,21} + x_{0,25} + x_{0,28} + x_{0,30} + x_{0,31} + x_{0,33} + x_{0,37} + x_{0,40} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,46} + x_{0,50} + x_{0,51} + x_{0,54} + x_{0,55} + x_{0,56} + x_{0,57} + x_{0,58} + x_{0,60}$

Table B3: This table illustrates the additional set of expressions (set to zero) to handle the effect of round constants to ensure 2 round linearisation for $\mathsf{SymSum}_{\mathsf{Vec}}/\mathsf{SymSum}_{\mathsf{Sim}}$ with increased MAD upto 114. The constraints are represented as $\mathcal{L}_j$. Here, the i^{th} variable of the lane (0,0) is denoted as $x_{0,i}$, while the variable of the lane (0,2) is represented as $x_{2,i}$.

	Equations
$\mathcal{L}_1$	$x_{0,0} + x_{0,1} + x_{0,2} + x_{0,16} + x_{0,20} + x_{0,21} + x_{0,22} + x_{0,24} + x_{0,25} + x_{0,29} + x_{0,31} + x_{0,32} + x_{0,46} + x_{0,47} + x_{0,48} + x_{0,49} + x_{0,51} + x_{0,52} + x_{0,53} + x_{0,54} + x_{0,56} + x_{0,57} + x_{0,60} + x_{0,62} + x_{2,0} + x_{2,1} + x_{2,2} + x_{2,3} + x_{2,7} + x_{2,9} + x_{2,10} + x_{2,12} + x_{2,13} + x_{2,14} + x_{2,15} + x_{2,16} + x_{2,21} + x_{2,22} + x_{2,23} + x_{2,25} + x_{2,26} + x_{2,27} + x_{2,29} + x_{2,32} + x_{2,33} + x_{2,35} + x_{2,38} + x_{2,39} + x_{2,41} + x_{2,45} + x_{2,46} + x_{2,47} + x_{2,49} + x_{2,52} + x_{2,54} + x_{2,55} + x_{2,56} + x_{2,61} + x_{2,62}$
$\mathcal{L}_2$	$x_{0,3} + x_{0,5} + x_{0,6} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,14} + x_{0,16} + x_{0,17} + x_{0,18} + x_{0,19} + x_{0,20} + x_{0,23} + x_{0,26} + x_{0,28} + x_{0,29} + x_{0,34} + x_{0,36} + x_{0,38} + x_{0,39} + x_{0,40} + x_{0,45} + x_{0,49} + x_{0,50} + x_{0,51} + x_{0,53} + x_{0,57} + x_{0,59} + x_{0,61} + x_{0,62} + x_{0,63} + x_{2,1} + x_{2,4} + x_{2,7} + x_{2,9} + x_{2,10} + x_{2,11} + x_{2,13} + x_{2,14} + x_{2,15} + x_{2,17} + x_{2,23} + x_{2,27} + x_{2,28} + x_{2,31} + x_{2,32} + x_{2,35} + x_{2,40} + x_{2,42} + x_{2,43} + x_{2,44} + x_{2,47} + x_{2,48} + x_{2,51} + x_{2,53} + x_{2,55} + x_{2,56} + x_{2,58} + x_{2,59} + x_{2,60} + x_{2,62} + x_{2,63}$
$\mathcal{L}_3$	$x_{0,1} + x_{0,3} + x_{0,4} + x_{0,5} + x_{0,10} + x_{0,14} + x_{0,15} + x_{0,16} + x_{0,18} + x_{0,22} + x_{0,24} + x_{0,26} + x_{0,27} + x_{0,28} + x_{0,32} + x_{0,34} + x_{0,35} + x_{0,40} + x_{0,41} + x_{0,42} + x_{0,43} + x_{0,45} + x_{0,46} + x_{0,47} + x_{0,48} + x_{0,49} + x_{0,52} + x_{0,55} + x_{0,57} + x_{0,58} + x_{0,63} + x_{2,0} + x_{2,5} + x_{2,7} + x_{2,8} + x_{2,9} + x_{2,12} + x_{2,13} + x_{2,16} + x_{2,18} + x_{2,20} + x_{2,21} + x_{2,23} + x_{2,24} + x_{2,25} + x_{2,27} + x_{2,28} + x_{2,30} + x_{2,33} + x_{2,36} + x_{2,38} + x_{2,39} + x_{2,40} + x_{2,42} + x_{2,43} + x_{2,44} + x_{2,46} + x_{2,52} + x_{2,56} + x_{2,57} + x_{2,60} + x_{2,61}$
$\mathcal{L}_4$	$x_{0,2} + x_{0,4} + x_{0,5} + x_{0,10} + x_{0,12} + x_{0,14} + x_{0,15} + x_{0,16} + x_{0,21} + x_{0,25} + x_{0,26} + x_{0,27} + x_{0,29} + x_{0,33} + x_{0,35} + x_{0,37} + x_{0,38} + x_{0,39} + x_{0,43} + x_{0,45} + x_{0,46} + x_{0,51} + x_{0,52} + x_{0,53} + x_{0,54} + x_{0,56} + x_{0,57} + x_{0,58} + x_{0,59} + x_{0,60} + x_{0,63} + x_{2,3} + x_{2,4} + x_{2,7} + x_{2,8} + x_{2,11} + x_{2,16} + x_{2,18} + x_{2,19} + x_{2,20} + x_{2,23} + x_{2,24} + x_{2,27} + x_{2,29} + x_{2,31} + x_{2,32} + x_{2,34} + x_{2,35} + x_{2,36} + x_{2,38} + x_{2,39} + x_{2,41} + x_{2,44} + x_{2,47} + x_{2,49} + x_{2,50} + x_{2,51} + x_{2,53} + x_{2,54} + x_{2,55} + x_{2,57} + x_{2,63}$
$\mathcal{L}_5$	$x_{0,2} + x_{0,4} + x_{0,6} + x_{0,7} + x_{0,8} + x_{0,13} + x_{0,17} + x_{0,18} + x_{0,19} + x_{0,21} + x_{0,25} + x_{0,27} + x_{0,29} + x_{0,30} + x_{0,31} + x_{0,35} + x_{0,37} + x_{0,38} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,46} + x_{0,48} + x_{0,49} + x_{0,50} + x_{0,51} + x_{0,52} + x_{0,55} + x_{0,58} + x_{0,60} + x_{0,61} + x_{2,0} + x_{2,3} + x_{2,8} + x_{2,10} + x_{2,11} + x_{2,12} + x_{2,15} + x_{2,16} + x_{2,19} + x_{2,21} + x_{2,23} + x_{2,24} + x_{2,26} + x_{2,27} + x_{2,28} + x_{2,30} + x_{2,31} + x_{2,33} + x_{2,36} + x_{2,39} + x_{2,41} + x_{2,42} + x_{2,43} + x_{2,45} + x_{2,46} + x_{2,47} + x_{2,49} + x_{2,55} + x_{2,59} + x_{2,60} + x_{2,63}$
$\mathcal{L}_6$	$x_{0,56}$
$\mathcal{L}_7$	$x_{0,0} + x_{0,2} + x_{0,3} + x_{0,7} + x_{0,9} + x_{0,10} + x_{0,24} + x_{0,25} + x_{0,26} + x_{0,27} + x_{0,29} + x_{0,30} + x_{0,31} + x_{0,32} + x_{0,34} + x_{0,35} + x_{0,38} + x_{0,40} + x_{0,42} + x_{0,43} + x_{0,44} + x_{0,58} + x_{0,62} + x_{0,63} + x_{2,0} + x_{2,1} + x_{2,3} + x_{2,4} + x_{2,5} + x_{2,7} + x_{2,10} + x_{2,11} + x_{2,13} + x_{2,16} + x_{2,17} + x_{2,19} + x_{2,23} + x_{2,24} + x_{2,25} + x_{2,27} + x_{2,30} + x_{2,32} + x_{2,33} + x_{2,34} + x_{2,39} + x_{2,40} + x_{2,42} + x_{2,43} + x_{2,44} + x_{2,45} + x_{2,49} + x_{2,51} + x_{2,52} + x_{2,54} + x_{2,55} + x_{2,56} + x_{2,57} + x_{2,58} + x_{2,63}$
$\mathcal{L}_8$	$x_{0,1}$

Table B4: Constraints given in Table B3 handles the round constant but disturbs symmetry, thus, this table illustrates the additional set of expressions (set to zero) to handle the effect of round constant and maintain the symmetry. The constraints are represented by $\mathcal{L}'_j$ to ensure 2 round linearisation for $\mathsf{SymSum}_{\mathsf{Sim}}$ with increased MAD upto 114.

	Equations
$\mathcal{L}'_1$	$x_{0,0} + x_{0,1} + x_{0,15} + x_{0,16} + x_{0,17} + x_{0,18} + x_{0,20} + x_{0,21} + x_{0,22} + x_{0,23} + x_{0,25} + x_{0,26} + x_{0,29} + x_{0,31} + x_{0,32} + x_{0,33} + x_{0,34} + x_{0,48} + x_{0,52} + x_{0,53} + x_{0,54} + x_{0,56} + x_{0,57} + x_{0,61} + x_{2,0} + x_{2,1} + x_{2,3} + x_{2,6} + x_{2,7} + x_{2,9} + x_{2,13} + x_{2,14} + x_{2,15} + x_{2,17} + x_{2,20} + x_{2,22} + x_{2,23} + x_{2,24} + x_{2,29} + x_{2,30} + x_{2,32} + x_{2,33} + x_{2,34} + x_{2,35} + x_{2,39} + x_{2,41} + x_{2,42} + x_{2,44} + x_{2,45} + x_{2,46} + x_{2,47} + x_{2,48} + x_{2,53} + x_{2,54} + x_{2,55} + x_{2,57} + x_{2,58} + x_{2,59} + x_{2,61}$
$\mathcal{L}'_2$	$x_{0,3} + x_{0,5} + x_{0,7} + x_{0,8} + x_{0,9} + x_{0,14} + x_{0,18} + x_{0,19} + x_{0,20} + x_{0,22} + x_{0,26} + x_{0,28} + x_{0,30} + x_{0,31} + x_{0,32} + x_{0,35} + x_{0,37} + x_{0,38} + x_{0,43} + x_{0,44} + x_{0,45} + x_{0,46} + x_{0,48} + x_{0,49} + x_{0,50} + x_{0,51} + x_{0,52} + x_{0,55} + x_{0,58} + x_{0,60} + x_{0,61} + x_{2,0} + x_{2,3} + x_{2,8} + x_{2,10} + x_{2,11} + x_{2,12} + x_{2,15} + x_{2,16} + x_{2,19} + x_{2,21} + x_{2,23} + x_{2,24} + x_{2,26} + x_{2,27} + x_{2,28} + x_{2,30} + x_{2,31} + x_{2,33} + x_{2,36} + x_{2,39} + x_{2,41} + x_{2,42} + x_{2,43} + x_{2,45} + x_{2,46} + x_{2,47} + x_{2,49} + x_{2,55} + x_{2,59} + x_{2,60} + x_{2,63}$
$\mathcal{L}'_3$	$x_{0,1} + x_{0,3} + x_{0,4} + x_{0,9} + x_{0,10} + x_{0,11} + x_{0,12} + x_{0,14} + x_{0,15} + x_{0,16} + x_{0,17} + x_{0,18} + x_{0,21} + x_{0,24} + x_{0,26} + x_{0,27} + x_{0,32} + x_{0,33} + x_{0,35} + x_{0,36} + x_{0,37} + x_{0,42} + x_{0,46} + x_{0,47} + x_{0,48} + x_{0,50} + x_{0,54} + x_{0,56} + x_{0,58} + x_{0,59} + x_{0,60} + x_{2,1} + x_{2,4} + x_{2,6} + x_{2,7} + x_{2,8} + x_{2,10} + x_{2,11} + x_{2,12} + x_{2,14} + x_{2,20} + x_{2,24} + x_{2,25} + x_{2,28} + x_{2,29} + x_{2,32} + x_{2,37} + x_{2,39} + x_{2,40} + x_{2,41} + x_{2,44} + x_{2,45} + x_{2,48} + x_{2,50} + x_{2,52} + x_{2,53} + x_{2,55} + x_{2,56} + x_{2,57} + x_{2,59} + x_{2,60} + x_{2,62}$
$\mathcal{L}'_4$	$x_{0,2} + x_{0,4} + x_{0,6} + x_{0,7} + x_{0,8} + x_{0,12} + x_{0,14} + x_{0,15} + x_{0,20} + x_{0,21} + x_{0,22} + x_{0,23} + x_{0,25} + x_{0,26} + x_{0,27} + x_{0,28} + x_{0,29} + x_{0,32} + x_{0,34} + x_{0,36} + x_{0,37} + x_{0,42} + x_{0,44} + x_{0,46} + x_{0,47} + x_{0,48} + x_{0,53} + x_{0,57} + x_{0,58} + x_{0,59} + x_{0,61} + x_{2,0} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,6} + x_{2,7} + x_{2,9} + x_{2,12} + x_{2,15} + x_{2,17} + x_{2,18} + x_{2,19} + x_{2,21} + x_{2,22} + x_{2,23} + x_{2,25} + x_{2,31} + x_{2,35} + x_{2,36} + x_{2,39} + x_{2,40} + x_{2,43} + x_{2,48} + x_{2,50} + x_{2,51} + x_{2,52} + x_{2,55} + x_{2,56} + x_{2,59} + x_{2,61} + x_{2,63}$
$\mathcal{L}'_5$	$x_{0,0} + x_{0,4} + x_{0,6} + x_{0,7} + x_{0,12} + x_{0,13} + x_{0,14} + x_{0,15} + x_{0,17} + x_{0,18} + x_{0,19} + x_{0,20} + x_{0,21} + x_{0,24} + x_{0,27} + x_{0,29} + x_{0,30} + x_{0,34} + x_{0,36} + x_{0,38} + x_{0,39} + x_{0,40} + x_{0,45} + x_{0,49} + x_{0,50} + x_{0,51} + x_{0,53} + x_{0,57} + x_{0,59} + x_{0,61} + x_{0,62} + x_{2,1} + x_{2,4} + x_{2,7} + x_{2,9} + x_{2,10} + x_{2,11} + x_{2,13} + x_{2,14} + x_{2,15} + x_{2,17} + x_{2,23} + x_{2,27} + x_{2,28} + x_{2,31} + x_{2,32} + x_{2,35} + x_{2,40} + x_{2,42} + x_{2,43} + x_{2,44} + x_{2,47} + x_{2,48} + x_{2,51} + x_{2,53} + x_{2,55} + x_{2,56} + x_{2,58} + x_{2,59} + x_{2,60} + x_{2,62} + x_{2,63}$
$\mathcal{L}'_6$	$x_{0,24}$
$\mathcal{L}'_7$	$x_{0,0} + x_{0,1} + x_{0,3} + x_{0,4} + x_{0,7} + x_{0,9} + x_{0,11} + x_{0,12} + x_{0,13} + x_{0,27} + x_{0,31} + x_{0,32} + x_{0,32} + x_{0,34} + x_{0,35} + x_{0,39} + x_{0,41} + x_{0,42} + x_{0,56} + x_{0,57} + x_{0,58} + x_{0,59} + x_{0,61} + x_{0,62} + x_{2,0} + x_{2,1} + x_{2,2} + x_{2,7} + x_{2,8} + x_{2,10} + x_{2,11} + x_{2,12} + x_{2,13} + x_{2,17} + x_{2,19} + x_{2,20} + x_{2,22} + x_{2,23} + x_{2,24} + x_{2,25} + x_{2,26} + x_{2,31} + x_{2,32} + x_{2,33} + x_{2,35} + x_{2,36} + x_{2,37} + x_{2,39} + x_{2,42} + x_{2,43} + x_{2,45} + x_{2,48} + x_{2,49} + x_{2,51} + x_{2,55} + x_{2,56} + x_{2,57} + x_{2,59} + x_{2,62}$
$\mathcal{L}'_8$	$x_{0,33}$

Appendix C Degree Difference in Bash-f

The following Theorem examines the maximum degree term which is influenced by the round constant.

Theorem 3. *In a Bash-f permutation consisting of n_r rounds, the highest degree of a monomial that includes a round constant is $d^{\circ}F^{n_r} - 2$.*

Proof. The basic idea of this proof is similar to Keccak. We provide the proof in for the sake of completeness. Consider the following round function $\mathcal{F} : \mathbb{F}_n^2 \rightarrow \mathbb{F}_n^2$ and $\mathcal{F} = \mathcal{C} \circ \mathcal{L}_1 \circ \mathcal{N} \circ \mathcal{L}$, where $\mathcal{C}$ denotes round-constant addition, $\mathcal{N}$ is the non-linear component, and $\mathcal{L}$ and $\mathcal{L}_1$ are the linear component. Without sacrificing generality, we can think of the component operations in the following order: $\mathcal{F} = \mathcal{C} \circ \mathcal{N}^{'} \circ \mathcal{L}$, where $\mathcal{N}^{'} = \mathcal{L}_1 \circ \mathcal{N}$. The algebraic degree and arrangement of operations in bash closely resemble the permutation in Keccak-p. Hence, by applying the principles presented in [SKC17], we can prove this. □