Considerations When Designing Sponge-based Extendable-Output Functions for Lightweight and Mobile Devices

Meaad Tori, David Paul, William Billlingsley
School of Science & Technology
University of New England
Armidale, Australia
email: mtori@myune.edu.au, David.Paul@une.edu.au, wbilling@une.edu.au

Abstract—**Through the study of existing lightweight cryptographic algorithms, we suggest a number of design guidelines for creating new sponge-based extendable-output functions for use in resource-constrained environments. While several such algorithms exist, some knowledge that can be generalized from studying them in aggregate has not previously been presented. The developed guidelines include consideration of the round function width, the number of rounds, the selection of round constants, the rate, the linear and non-linear layers, and the required security claim. The result of these guidelines is a set of recommendations for the design of sponge-based extendable-output functions that should allow correctly balanced security and performance in environments where compute power, available memory, and battery life may all be limited. These recommendations could be used to help design purpose-built implementations for various wireless or mobile systems.**

Keywords-lightweight cryptography; extendable-output function; security analysis; Internet of Things.

I. Introduction

The Internet of Things (IoT), sensor networks, Radio Frequency Identifiers (RFID) and smart devices are connecting the world in ways not previously imagined [1]. However, many of these devices are resource-limited, having low computational power, small amounts of memory and limited power supply, often relying on batteries. Because of this limitation, traditional standardized cryptographic algorithms, such as the Advanced Encryption Standard (AES) [2] and Secure Hash Algorithms SHA-2 [3] and SHA-3 [4], which can have high computational and memory requirements, are not appropriate for use in these lightweight devices.

The combination of having these resource-constrained devices interacting directly with the real world and not being able to protect them using traditional algorithms means new approaches are needed to ensure their security and privacy [5]. Lightweight cryptography aims to develop secure cryptographic primitives that better fit the environment of resource-constrained devices [5]. The US National Institute of Standards and Technology (NIST) is currently holding a competition to standardize lightweight cryptographic algorithms because the performance of existing standards is not acceptable [6] but the competition does not cover all cryptographic primitives. In particular, the competition does not include an extendable-output function (XOF), and even a cryptographic hash function is optional.

A cryptographic hash function is an algorithm that maps a message of any length to a fixed-size message digest. It is a one-way function that is difficult and impractical to invert, and is important for many forms of authentication, including digital signatures [7]. An XOF has similar functionality to a cryptographic hash function, but its output can be extended to any desired length, rather than a single fixed size. This can prove very useful in lightweight environments, allowing system designers to choose the length of the output required for their individual circumstances to better balance security and performance [8].

While the inclusion of a cryptographic hash function is optional in the current NIST lightweight cryptography competition, 12 of the 32 second-round candidates included such an algorithm. With the exception of SATURNIN [8], each of these candidates chose to use a sponge construction (or derivative) [9], which allows for XOFs. Of the ten finalists in the NIST competition, the Ascon [10], Photon-Beetle [11], TinyJAMBU [12] and Xoodyak [13] algorithms are the only ones that include hashing, and each of these are based on a sponge construction.

In this paper, we examine three representative candidates from the second round of the NIST lightweight cryptography competition and, combined with general insight from the other candidates and related research, present a new set of design aspects that must be taken into account to design a secure lightweight sponge-based XOF. Our analysis includes two of the finalists (Ascon and Xoodyak) and one algorithm that did not make it to the final round of the NIST competition (Gimli [14]). The inclusion of Gimli in this analysis is because some of the reasons it did not make it to the final round are pertinent to this discussion. This aggregate study leads to general guidelines that could be used to develop custom-built XOFs for lightweight environments, including wireless and mobile systems.

The remainder of this paper is organized as follows. Section II recounts literature-supported background information required to understand the analysis of the existing lightweight sponge-based XOFs presented in

Section III. Section IV then generalizes the outcomes of the analysis to present a number of design guidelines that should be considered when creating such an XOF. Finally, Section V concludes the paper and suggests future directions for this research.

II. Background Information

This section briefly outlines, in Section II-A, the requirements of cryptographic hash functions and, in Section II-B, why it is often useful to have a more general XOF instead of a fixed-sized hash. Section II-C then gives an overview of the general sponge construction, which is required to create sponge-based XOFs. This information will be important when we examine three sponge-based XOFs in Section III.

A. Cryptographic Hash Function

A hash function converts an arbitrarily-sized message into a message digest of some fixed length, say *d*. In order to be a cryptographic hash function, a hash function must also have the following properties [4]:

- Pre-image resistance: Given a particular message digest, it should be difficult to find a message that maps to that value.
- Second pre-image resistance: Given a particular message, it should be difficult to find a different message that has the same message digest.
- Collision resistance: It should be difficult to find any two different messages that have the same message digest.

Generic attacks on hash functions, such as brute force (which repeatedly tries different inputs until the desired message digest is found), depend only on the value of *d*, so *d* must be large enough to ensure that such an attack is computationally inefficient. In general, attacks on hash functions attempt to break (some of) the above properties of cryptographic hash functions without resorting to brute force (i.e., the attack should take fewer than 2^d steps).

B. Applications of Extandable-Output Functions

XOFs generalize hash functions by allowing an arbitrary output digest size. The computational complexity of an XOF is a combination of the computational complexity of a hash and a stream cipher [15]. Thus, the security of XOFs relies on more than just the length of the produced digest, so different security strengths can be selected. This is useful in areas where available key material might vary dramatically from one application to another, with no correlation to the required security strength [4].

For example, the ED448 digital signature standard [7] adopts the XOF SHAKE-256 [4] as its internal hash function. This significantly increases performance compared to using SHA3-512, without reducing the 256 bits of security required by the standard [7].

C. The Sponge Construction

Each of the finalists in the NIST lightweight cryptography competition that support a cryptographic hash function use a sponge construction [16], which can generally be extended to an XOF. A sponge function, as illustrated in Figure 1, is built from three components:

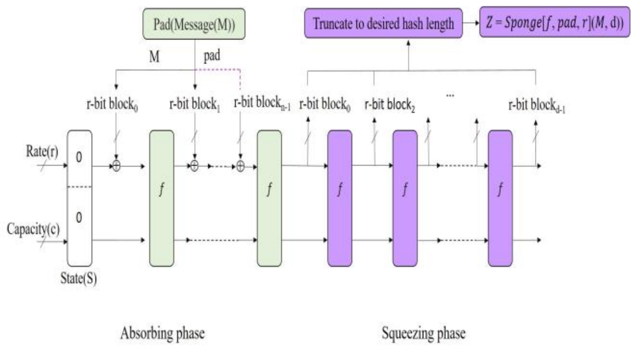

Figure 1. Hashing mode in Sponge Construction.

- A state memory, *S*, consisting of $r + c$ bits, where r is the *rate* of the sponge, and *c* its *capacity*.
- A function f: $\{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$ that transforms the state memory. It typically consists of a non-linear, a mixing, and a linear layer.
- A padding function *Pad* that appends bits to any input string to ensure its length is a multiple of *r*.

The state is initialized to zero and then, for each *r*-bit block of the padded input string, the state is updated by replacing the first *r* bits of the state with the first *r* bits of the state bitwise XORed with the *r*-bit input block. The state is then further updated by passing it through the function f, which is often a pseudorandom permutation over all possible state values. This "absorbs" all blocks of the padded input into the sponge construction's state.

The output of the sponge construction is then "squeezed out" by initially outputting the first *r*-bits of the state and then, repeatedly until enough output is generated, replacing the state *S* by $f(S)$ and outputting the first *r* bits of the result (truncating if necessary).

Assuming f is suitably difficult to invert, the following security results can be derived for a sponge construction that creates a message digest of length *d* [17] [18] [36]:

If $d \geq c$ and $c > 2r$ then:

- The construction has pre-image security of 2^{d-r}.
- The construction has second pre-image and collision security of $2^{c/2}$.
- The best pre-image attack would require a complexity of $2^{d-r} + 2^{c/2}$.

Otherwise:

- The construction has second pre-image security of 2^d.
- The construction has collision security of $2^{c/2}$.
- The best pre-image attack would require a complexity of $\{\min 2^d, \{\max 2^{d-r}, 2^{c/2}\}\}$.

The security claims of a sponge construction are typically flattened to rely purely on the capacity *c*, allowing the required security to be defined independently of the length of the output *d* [16]. Further, the sponge construction is often used with duplexing to allow the absorb and squeeze operations to alternate [19].

TABLE I. COMPARSION OF ASCON, GIMLI AND XOODYAK XOFS

Algorithm	Number of Rounds	State Size (bits)	Rate (bits)	Capacity (bits)
Ascon	12	320	64	256
Gimli	24	384	128	256
Xoodyak	12	384	130	254

III. EXISTING SPONGE-BASED XOFS

This section examines Ascon-XOF (in Section III-A), Gimli-XOF (in Section III-B) and Xoodyak-XOF (in Section III-C) in order to understand and generalize design decisions for creating sponge-based XOFs. A comparison of the three algorithms is presented in Table I. Lessons learned from these algorithms will be presented as a set of design considerations in Section IV.

A. Ascon-XOF

Ascon-XOF [10] uses a 12-round permutation based on a sponge construction with a state size of 320 bits, consisting of five 64-bit words. It uses a 64-bit rate and 256-bit capacity. The substitution layer is identical to the Keccak χ mapping [20] and an adaptation of the $\sum$ function of SHA-2 [21] is used to provide diffusion.

Ascon-XOF has received significant third-party analysis (e.g., [22]). A summary is provided in Table II. The pre-image attacks target the low algebraic degree of the reduced round version of Ascon-XOF; the search for pre-images can be speed up for low degree functions. Ascon-XOF has fast diffusion because it applies its linear layer to every five words. It also has a strong word structure with a good choice of round constants, which makes it challenging to apply a cube attack [23] effectively. However, since each output bit depends on only three input bits, of which two are non-linear, consecutive dependent bits can lead to the derivation of linear equations that can be solved to break the system. Even the original Ascon specification [10] admits that the Ascon permutation is not ideal in terms of differential and linear properties [22] [24]. However, it has been shown that Ascon-XOF has a good security margin against collision attacks [22]. Currently, even with the use of all 320 bits of the state in a semi-free-start collision, only four out of Ascon-XOF's twelve rounds can effectively be broken.

B. Gimli-XOF

Gimli-XOF [14] uses a 24 round permutation based on sponge construction with a state size of 384 bits, represented as a 3×4 matrix of 32-bit words. It uses a 128-bit rate and 256-bit capacity. The non-linear layer operates on the column level. The linear layer operates on the row level and applies one of two swap operations, a small swap, or a big swap.

Gimli has a slow diffusion compared to Ascon because its small and big swap operations only apply to the first row, and not in every round. This makes it easier to analyze multiple rounds of Gimli-XOF. Table III demonstrates Gimli's lower diffusion compared to Ascon.

TABLE II. SUMMARY OF ATTACKS ON ASCON-XOF

Attack	Method	Round	Time Complexity	Ref
Pre-image	cube	2	2^{39}	[22]
Pre-image	Algebraic	6	$2^{63.3}$	[22]
SFS collision	Differential	4	Practical	[22]
Collision	Differential	2	Practical-2^{15}	[25]

TABLE III. UPPER BOUND FOR THE ALGEBRAIC DEGREE OF DIFFUSION AFTER DIFFERENT NUMBERS OF ROUNDS FOR ASCON AND GIMLI

Round	1	2	3	4	5	6	7	8	9
Ascon	2	4	8	16	32	64	128	256	298
Gimli	2	4	8	16	29	52	95	163	266

A divide-and-conquer technique, which applies an exhaustive search to a divided message space, allows theoretical pre-image attacks on up to five of the nine rounds of Gimli-XOF [26]. By exploiting Gimli's weak diffusion, equations that represent the bit dependencies in Gimli-XOF's rate can be constructed and solved. This did require fixing the block size to 128 bits, and ignoring the padding rule, but does give a practical attack on a reduced-round version of Gimli-XOF.

The slow diffusion of Gimli's state means that the swap operations only affect 256 of the 384 bits of Gimli's state, and this does not even occur each round. This can be exploited to construct equations that can be practically solved with a SAT solver [27].

C. Xoodyak-XOF

Xoodyak-XOF [13] uses a 12-round permutation based on a sponge construction with a state size of 384 bits, consisting of three planes of 128 bits. It uses a 130-bit rate and 254-bit capacity (reduced by two for internal reasons [13]). It is based on the Xoodoo Permutation [28] and uses a column parity mixer [29]; this provides good diffusion and is suitable for modes that do not need inverses, such as sponge constructions. The non-linear layer uses a shift-invariant mapping based on the parity of three bits and implements bitwise boolean operations. The narrowing of the non-linear layer from five bits to three bits increases Xoodyak's resistance to cube attacks [13].

A deep-learning pre-image attack has been proposed on Xoodyak-Hash [30], though only with a fixed message size of 32 bits and with adjusted squeeze rates, hash lengths, or round numbers. The first model increases the squeezing rate to 384 bits, rather than the original 128 bits, representing the entire state. The second model increases the hash length to 384 bits, rather than the original 256. The third model is identical to Xoodyak-Hash, but they reduce the number of rounds to just one. Xoodyak-Hash is proven to be strong enough to resist these attacks, as they only have any success on at most one round [30], though it has been demonstrated that reducing the capacity of Xoodyak down to 128 bits helps make pre-image attacks over a small number of rounds possible [30].

IV. Design Consideration for Lightweight Sponge-based XOFs

The main goal when designing a lightweight XOF should be to provide the best trade-off between security and performance in both hardware and software. While the sponge construction gives a general framework that can work well in a resource-constrained environment, choices related to a particular implementation can greatly affect the overall result. In this section, we discuss the main choices that need to be made, including recommendations and considerations, when developing a sponge-based XOF.

A. Round Function Width

In general, a wider round function (i.e., one that maps more bits) offers improved security over a narrow one, though typically has performance and cost implications [29]. Wider round functions may require more circuitry or more complex software implementations which may not be appropriate in lightweight environments. Implementing widths that are a multiple of 32 or 64 bits, such as Ascon (320-bit state), Gimli (384-bit state) and Xoodyak (384-bit state), can allow vectorization on some platforms to allow parallel computation on different blocks of the state. In contrast Keccack [15], which SHA-3 is based on, uses permutations that are a multiple of 25 bits, which can severely impact performance on lightweight devices since vectorization cannot be used [14].

B. Number Of Rounds

A round function is typically used multiple times per round, potentially with some different parameters (e.g., round constants). A high number of rounds reduces performance but can improve security [31]. For lightweight algorithms it is best to select the minimum number of rounds for which there are no shortcut attacks that have a higher success probability than generic attacks such as brute force. Linear, differential and truncated differential [32] attacks exploit constructed propagation in n rounds, then attack later rounds of the primitive. If an attack is successful on n rounds, the designer should double the number of rounds to increase security resistance [33].

C. Selection Of Round Constants

Good rounds constants eliminate symmetries in iterative primitives [33]. Round constants should be different for each round, independent of the non-linear layer and defined by a specific rule to avoid slide, rotational, self-similarity, and similar attacks [34].

To see how important round constants are, consider Gimli. Gimli's use of round constants only each four rounds and having them affect just one 32-bit word of the state, led to the construction of a distinguisher for the full Gimli permutation [27]. Instead, some constant rotation should be applied each round to help provide fast diffusion [13].

Round constants also have an implication for performance. For example, Ascon's choice of round constants allows pipelining, while still ensuring that differential attacks are impossible [10].

D. Rate

In general, a low rate is less susceptible to pre-image and differential cryptanalysis attacks [31]. In Ascon-XOF, a rate of 64 bits is used. Increasing the rate would decrease the capacity, reducing robustness of the primitive [10] and increasing the likelihood of rebound attacks [36]. Recent collision attacks on Ascon [22] [25], Gimli [26] [27] and Xoodyak [30] are constrained to at most six round reduced versions of these algorithms, primarily because each uses a small rate.

E. Non-linear Layer

The non-linear layer is essentially an s-box, which is a vectorial Boolean function that performs substitution. It is mainly responsible for creating confusion (measured using Shannon's entropy) in the cryptographic primitive [37]. There is a trade-off between the size of s-boxes and the security they provide; a small differential probability, high algebraic degree and significant non-linearity reduce the number of rounds needed to secure the primitive, but often require larger, less efficient sizes [38]. In resource-constrained devices, designers are left with the choice of using smaller optimal s-boxes (perhaps combined to give a virtual larger s-box) or using a larger sub-optimal s-box that gives moderate performance [38]. In lightweight cryptographic primitives, smaller 4×4 s-boxes are the most commonly used [38].

F. Linear Layer

The design of a linear layer specifies how the non-linear and mixing layers are combined and affects propagation of the function [39]. For lightweight cryptography, a bit-oriented design should be used to improve efficiency. Further, the linear layer should be carefully considered to ease the derivation of algebraic, diffusion and correlation propagation [29]. This is achievable by ensuring weak alignment of the primitive [40], as this reduces susceptibility to truncated differential, saturation and trail clustering efforts related to differential or linear cryptanalysis. Ascon-XOF achieves this using a similar approach to SHA-2, using variant rotation constants for each word without decreasing performance [10]. On the other hand, Xoodyak uses column parity mixers, similar to Keccak [29], which are lightweight and offer weak alignment [39]. Further, using an odd number of rows makes a column parity mixer invertible, which gives immediate full diffusion in the backward direction [29].

G. Security Cliam

For sponge-based XOFs, the security claim is typically flattened to only rely on the capacity *c*, though the length of the digest used must be long enough to make generic attacks implausible (i.e., at least 128-bits). It is recommended to use a 255-bit capacity to give a strength of 128 bits [41].

V. Conclusion And Future Work

This paper examined three sponge-based XOF implementations (Ascon-XOF, Gimli-XOF and Xoodyak-XOF) to inform some design considerations that need to be taken into account when designing a secure lightweight

XOF. While exact considerations depend on the environment in which the XOF will operate, the following general guidelines were presented:

- Wider round functions offer improved security, though often at the expense of performance.
- Many platforms that support vectorization perform best when the round function width is a multiple of 32 or 64 bits.
- The number of rounds used should be minimized to improve performance while giving an adequate level of security; if there is a known attack on n rounds, doubling the number of rounds should give adequate security in many cases.
- Round constants should be different for each round and should eliminate symmetries in the primitive.
- Low absorbing and squeezing rates are preferable; 64 bits seems to be a good rate size.
- The non-linear layer should consist of smaller (e.g., 4×4) optimal s-boxes or larger sub-optimal s-boxes to balance performance and security.
- A bit-oriented design should be used for the linear layer and should ensure weak alignment of the primitive. The use of column parity mixers is recommended.
- A capacity of at least 255 bits and a digest length of at least 128 bits should be used to provide a security strength of 128 bits.

These guidelines help ensure the creation of a performant and secure lightweight sponge-based XOF, suitable for use in low-resource environments such as mobile systems. Future work will consider automatic security analysis of XOFs created using these guidelines with the goal of using evolutionary computation to design XOFs that are fit for purpose in a wide range of resource-constrained environments.

REFERENCES

[1] K. McKay, L. Bassham, M. Sönmez Turan, and N. Mouha, “Report on lightweight cryptography,” tech. rep., National Institute of Standards and Technology, 2016.

[2] J. Daemen and V. Rijmen, “AES and the wide trail design strategy,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2332, no. April 2002, pp. 108–109, 2002.

[3] W. Penard and T. van Werkhoven, “On the secure hash algorithm family,” *Cryptography in context*, pp. 1–18, 2008.

[4] M. J. Dworkin et al., “SHA-3 standard: Permutation-based hash and extendable-output functions,” tech. rep., National Institute of Standards and Technology, 2015.

[5] S. P. Jadhav, “Towards light weight cryptography schemes for resource constraint devices in IoT,” *Journal of Mobile Multimedia*, pp. 91–110, 2019.

[6] National Institute of Standards and Technology, “Lightweight Cryptography,” tech. rep., National Institute of Standards and Technology, 2022.

[7] National Institute of Standards and Technology, “Digital Signature Standard (DSS),” tech. rep., National Institute of Standards and Technology, 2013.

[8] A. Canteaut et al., “Saturnin: A suite of lightweight symmetric algorithms for post-quantum security,” *IACR Transactions on Symmetric Cryptology*, vol. 2020, Special Issue 1, pp. 160–207, 2020.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Cryptographic sponges,” tech. rep., Team Keccak, 2011.

[10] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, “Ascon v1.2,” *Submission to the CAESAR Competition*, 2016.

[11] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, “Beetle family of lightweight and secure authenticated encryption ciphers,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 2, pp. 218–241, 2018

[12] H. Wu and T. Huang, “TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms,” *Submission to Lightweight Cryptography Competition*, March, 2021.

[13] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, “Xoodyak, a lightweight cryptographic scheme,” in *IACR Transactions on Symmetric Cryptology*, vol. S1, pp. 60–87, 2020.

[14] D. J. Bernstein et al., “Gimli: a cross-platform permutation,” in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 299–320, Springer, 2017.

[15] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Keccak sponge function family main document,” *Submission to NIST (Round 2)*, vol. 3, no. 30, pp. 320–337, 2009.

[16] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Sponge functions,” in *ECRYPT hash workshop*, vol. 9, 2007.

[17] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, “On the indifferentiability of the sponge construction,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 181– 197, Springer, 2008.

[18] J. Guo, T. Peyrin, and A. Poschmann, “The PHOTON family of lightweight hash functions,” in *Annual Cryptology Conference*, pp. 222– 239, Springer, 2011.

[19] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, “Duplexing the sponge: single-pass authenticated encryption and other applications,” in *International Workshop on Selected Areas in Cryptography*, pp. 320– 337, Springer, 2011.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, “Keccak,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 313–314, Springer, 2013.

[21] National Institute of Standards and Technology, “Secure Hash Standard (SHS) Publication,” *tech. rep. March 2012, National Institute of Standards and Technology*, 2012.

[22] C. Dobraunig and F. Mendel, “Preliminary Analysis of Ascon-Xof and Ascon-Hash,” *Tech. Rep.* 2019.

[23] I. Dinur and A. Shamir, “Cube attacks on tweakable black box polynomials,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 278–299, Springer, 2009.

[24] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, “Cryptanalysis of Ascon,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9048, pp. 371–387, 2015.

[25] R. Zong, X. Dong, and X. Wang, “Collision Attacks on Round-Reduced Gimli-Hash/Ascon-Xof/Ascon-Hash,” *Cryptology ePrint Archive*, 2019.

[26] F. Liu, T. Isobe, and W. Meier, “Exploiting Weak Diffusion of Gimli: Improved Distinguishers and Preimage Attacks,” *IACR Transactions on Symmetric Cryptology*, pp. 185–216, 2021.

[27] A. Flórez Gutiérrez, G. Leurent, M. Naya-Plasencia, L. Perrin, A. Schrottenloher, and F. Sibleyras, “New results on Gimli: full-permutation distinguishers and improved collisions,” in *International Conference on the Theory and*

Application of Cryptology and Information Security, pp. 33–63, Springer, 2020

[28] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, "Xoodoo Cookbook," *Cryptology ePrint Archive*, 2018.

[29] K. Stoffelen and J. Daemen, "Column Parity Mixers," *IACR Transactions on Symmetric Cryptology*, vol. 1, pp. 126–159, 2018.

[30] G. Liu, J. Lu, H. Li, P. Tang, and W. Qiu, "Preimage Attacks Against Lightweight Scheme Xoodyak Based on Deep Learning," in *Future of Information and Communication Conference*, pp. 637–648, Springer, 2021.

[31] L. Song, G. Liao, and J. Guo, "Non-full sbox linearization: applications to collision attacks on round-reduced Keccak," in *Annual International Cryptology Conference*, pp. 428–451, Springer, 2017.

[32] L. R. Kundsen, "Truncate and higher order differentials ," in *International Workshop on Fast Software Encryption*, pp. 196-211, Springer, 1994.

[33] J. Daemen and V. Rijmen, "The design of Rijindal, vol 2.2 Springer, 2002.

[34] C. Beierle, A. Canteaut, G. Leander, and Y. Rotella, "Proving resistance against invariant attacks: How to choose the round constants," in *Annual International Cryptology Conference*, pp.647-678, Springer, 2017.

[35] M. Eichlseder, "Differential Cryptanalysis of Symmetric Primitives," *Ausgezeichnete Informatikdissertationen*, 2019.

[36] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: the design space of lightweight cryptographic hashing," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2041– 2053, 2012.

[37] I. Haitner, T. Holenstein, O. Reingold, S. Vadhan, and H. Wee, "Universal one-way hash functions via inaccessible entropy," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 616–637, Springer, 2010.

[38] S. Picek, L. Mariot, B. Yang, D. Jakobovic, and N. Mentens, "Design of S-boxes defined with cellular automata rules," *ACM International Conference on Computing Frontiers 2017, CF 2017*, pp. 409–414, 2017.

[39] N. Bordes, J. Daemen, D. Kuijsters, and G. V. Assche, "Thinking Outside the Superbox," in *Annual International Cryptology Conference*, pp. 337–367, Springer, 2021.

[40] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche , *"On alignment in Keccak,"* ECRYPT II Hash Workshop, pp. 1-18, 2011.

[41] G. Bertoni et al., "Kangaroo twelve: Fast Hashing Based on Keccak-p," in *International Conference on Applied Cryptography and Network Security*, pp. 400-418, Springer, 2018.