Hortex: A Chaotic Sponge Hash Function

Ikhwanul Hakim Masri and Bety Hayat Susanti

***Abstract*—The hash function is one of the primitive cryptography techniques that is still widely used today, owing to its importance in preserving the integrity of digital data. Various hash functions, such as SHA-1, SHA-2, and SHA-3, have been standardized in the digital world to solve data integrity challenges. However, vulnerabilities have been discovered in some hash function standards through cryptanalysis, making it necessary to develop alternative hash function designs. The proposed hash algorithm, which was named Hortex, is an alternative hash function based on a chaotic sponge that generates an output length of 128 bits. The Hortex algorithm was tested using Cryptographic Randomness Testing (CRT) to evaluate its randomness properties, statistical analysis to examine its confusion and diffusion characteristics, and attacked using Yuval's birthday attack and brute-force attacks theoretically to evaluate its security. The results of the test show that the Hortex algorithm passes the CRT test, indicating that it has good randomness properties. The results of the statistical analysis indicate that Hortex exhibits good confusion and diffusion properties.The results of the theoretical attack on Hortex show that Hortex is resistant to Yuval's birthday attack and has a complexity of 2^{128} for the preimage and second preimage resistance and 2^{64} for collision resistance.**

***Index Terms*—chaos function, chaotic sponge, collision attack, cryptographic randomness testing, hash function, sponge construction.**

I. Introduction

INFORMATION SECURITY has become crucial in the era of Industry 4.0, particularly due to the increasing threats of various cyber attacks [1]. At its core, information security revolves around three fundamental principles, that is, Confidentiality, Integrity, and Availability (CIA) [2]. These principles are essential for protecting sensitive data and ensuring that it remains secure from unauthorized access and tampering. An effective way to achieve these services is through the use of cryptographic techniques, particularly hash functions. Hash functions play a key role in providing data integrity and authentication, both of which are vital to safeguarding information in the digital landscape today [3].

Hash functions are commonly constructed in various ways, and one popular method is based on Merkle-Damgard (MD) construction. Some MD-constructed hash functions include MD4, MD5, SHA-1, SHA-2, and RIPEMD [4]. In 2004 and 2005, X. Wang successfully executed collision attacks on the MD5 and SHA-1 hash function standards, leading to the establishment of new hash function standards [5], [6]. In 2008, SHA-2, which replaced SHA-1 as the standard hash function, was also subjected to a collision attack by [7].

Manuscript received August 30, 2024; revised January 24, 2025.

This research is supported by Politeknik Siber dan Sandi Negara, Bogor, Jawa Barat, Indonesia.

I. H. Masri is a Research Assistant at Badan Siber dan Sandi Negara, South Jakarta 12550, Indonesia (e-mail: ikhwanul.hakim@bssn.go.id)

B. H. Susanti is an Associate Professor in the Department of Cryptographic Engineering, Politeknik Siber dan Sandi Negara, Bogor, 16120, Indonesia (corresponding author, e-mail: bety.hayat@poltekssn.ac.id, bety.hayat@bssn.go.id)

Since the discovery of collisions in the MD-based hash function standard, numerous researchers have endeavored to find and develop new construction models to replace MD. Some of these models include tree-based constructions, wide-pipe designs, the hash iterative framework (HAIFA), and the sponge function [8]. Among these construction models, sponge-based construction has garnered the primary attention due to the latest hash function standards, SHA-3 or KECCAK, which employ the sponge function as their foundational structure [9]. Since its introduction, SHA-3 has been the subject of various cryptanalysis attacks [10], [11], [12]. Although SHA-3 has not been completely compromised, research on the design of alternative, more secure hash functions remains crucial [13].

Several researchers have designed hash functions using sponge construction, such as Titanium and RM70 [14], [15]. Furthermore, research on sponge-based hash function designs has become more diverse, including the incorporation of chaotic functions. The involvement of chaotic functions is due to the fact that they produce random output while simultaneously being deterministic and highly sensitive to input changes. This property is well suited for building hash functions [16]. [17] designed a chaotic sponge-based hash function using chaos functions based on DNA code sequences. On the other hand, [18] designed a chaotic sponge-based hash function using neural network structures. [19] also designed a chaotic sponge-based hash function with a two-dimensional structured chaos function.

The selection of a chaotic function for designing a hash function is typically based on the chaotic region of the chaotic function, as determined by bifurcation diagrams and Lyapunov exponents [20]. The chaotic region is defined using bifurcation diagrams and Lyapunov exponents. This chaotic region is used as a tool to manipulate the randomness in the structure of the designed hash function. Chaotic functions are generally categorized based on their dimensionality, ranging from 1D, 2D, 3D, to nD chaotic maps. Among various types of chaotic functions, 1D chaotic maps are the simplest to implement and are therefore popular for cryptographic applications. One of the simplest examples is the utilization of the logistic map as an encryption function in images [21]. Higher-dimensional chaotic functions, such as 2D and 3D chaotic maps, offer higher security levels but involve a more complex implementation. The same principle applies in general to nD chaotic maps. Nonetheless, some research has explored the use of chaotic functions in cryptography, including variants of 2D chaotic maps and even higher-dimensional variants [22], [23], [24].

The aim of this study is to design a chaotic sponge-based hash function called Hortex. The sponge function is chosen as the basis for the construction to mitigate length extension attacks, which are susceptible in MD-based constructions, as observed in previous standard hash functions [8]. The selection of the chaos function as one of the building blocks

of the hash function is based on the favorable properties of the chaos functions to construct the hash functions [16]. The specific chaos function used is the 1D chaotic map known as the Enhanced Logistic Map, introduced by [25], chosen for its superior chaotic properties compared to several other 1D chaotic maps. The proposed hash function, Hortex, will be subjected to testing and evaluation to assess its randomness and security properties. Randomness will be assessed using Cryptographic Randomness Testing (CRT) [26]. The security will be evaluated by applying cryptographic tests to the collision resistance property of the hash function. This is done because collision resistance has a lower security level compared to the other two properties (preimage and second preimage resistance) [27]. Therefore, if the collision resistance property is not satisfactory, it can be inferred that the other two properties will also exhibit poor performance.

The structure of this paper is organized as follows. Section II discusses several studies that design hash functions based on chaotic functions and sponge functions. Section III addresses the prerequisite materials used in the design of hash functions. Section IV covers the description of the proposed hash function design named Hortex. Section V conducts randomness testing on Hortex using CRT. Section VI performs theoretical security testing on the collision resistance property of the Hortex algorithm. Section VII presents the conclusions drawn from the research conducted in this paper. Additionally, Appendix A provides a more detailed explanation of performing randomness tests using CRT on the Hortex algorithm.

II. Related Work

Several researchers have explored various methods and approaches for designing hash functions. One such approach involves using chaotic functions with the purpose of influencing randomness, and the key to implementing chaos functions in a design lies in how to preserve their chaotic characteristics. An example of designing a hash function using chaotic functions was done by [28], where they constructed a keyed hash function. In their hash function design, they utilized a chaotic iteration function, the domain of which is confined to integers. The advantage of this lies in the fact that, due to its limited domain within integers, there is no necessity for further transformation of the chaotic function's outcomes to preserve the chaotic characteristics upon implementation. This stands in contrast to conventional implementations of chaotic functions, wherein the results typically remain within the realm of real numbers, necessitating additional transformations for similar preservation of chaotic attributes. In a similar vein, in 2019, [29] also pursued the design of keyed hash functions based on chaotic functions. In their implementation of chaotic functions, they utilized fixed-point representations to maintain the chaotic characteristics of the chosen chaotic function. This approach differs from the one employed by Lin et al. primarily because Teh et al. chosen chaotic function operates within the real number domain, requiring special considerations to preserve its chaotic characteristics effectively. On a different note, in 2022, [25] engaged in the design of a chaotic function called the ELM function, which exhibited superior performance compared to several other chaotic functions.

Recently, alternative designs of hash functions have focused on employing the construction of a sponge function, as demonstrated by [14], which integrates a Substitution Permutation Network (SPN) with a sponge function. Furthermore, other researchers have undertaken the design of hash functions based on the sponge function, coupled with the implementation of chaos functions to enhance the randomness properties in the design, as exemplified by [17]. Alawida et al. employ coding techniques in the implementation of their chaos function to retain its chaotic characteristics in the hash function design. [19] also devised hash function designs based on the sponge and chaos function, with the distinction lies in the type of chaos function employed, specifically a 2D map. Their research aimed to expand the mapping space, or domain, of the chaos function. Another approach to constructing hash functions based on the sponge and chaos function was undertaken by [18]. Abdoun and colleagues utilized discrete chaos functions in conjunction with a neural network structure to preserve the inherent chaotic properties of the chaos function in its implementation.

III. Preliminaries

A. Hash Function

Hash function serves as a classical cryptographic algorithm that plays a crucial role in modern cryptography. A hash function takes an arbitrary-sized input message and produces a fixed-size output, typically referred to as the hash or digest, representing the compressed form of the input message. [27] state that a hash function should be computationally efficient while generating the digest or hash of an input message. Hash functions are commonly employed as a tool to ensure data integrity in modern cryptography. In addition to compression and computational efficiency, a hash function must also exhibit three other properties for strong cryptographic resilience, namely preimage resistance, second preimage resistance, and collision resistance [27]. Let X represent an input, and $H(\cdot)$ denotes the hash function. Preimage resistance is a property of a hash function, signifying that, computationally, it is difficult to derive the original input X from its hash value $H(X)$. Second preimage resistance is another property of a hash function, meaning that if both the input X and its hash value $H(X)$ are known, it remains computationally difficult to find any other input X' that results in the same hash value, i.e., $H(X) = H(X')$. Collision resistance, on the other hand, is a property of a hash function that makes it computationally difficult to find two distinct inputs $(X \neq X')$ that produce the same hash value, i.e., $H(X) = H(X')$.

B. Sponge Function

The sponge function is a function that takes input of a specific length and produces an output of arbitrary length through a repeated iteration process on a transformation or permutation function that receives a fixed-sized state. The term "sponge" was introduced by [30], because this function exhibits two phases similar to the properties of a sponge, namely the absorbing phase and the squeezing phase. The absorption phase absorbs input bits of any quantity into a state using a transformation function, and subsequently, this state enters the squeezing phase to generate output bits [30].

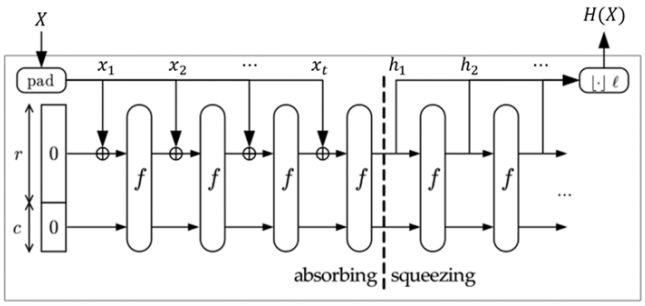

Fig. 1. Sponge Function[1]

Figure 1 illustrates the sponge function with an input X and a state of length s, where the state is the sum of the capacity (c) and rate (r). In the sponge function, the input X is divided into t blocks, denoted as $x_1, x_2, \ldots, x_t$, each of length r. During the absorbing phase, the state s absorbs x_i, followed by the application of the transformation or permutation function f onto s. In the squeezing phase, the state s is utilized as the input for the function f again, and for each iteration, the output of f is taken as the r most significant bits (*msb*), denoted as h_i. The output of the sponge function is $H(X)$, which is the result of concatenating h_i along a length of l. The padding function, pad, is applied to the input X if its length is not a multiple of r. If the output of the sponge function is of length l, and the capacity of the sponge function is c, then the security level of the sponge function against collisions is given by $\min(2^{c/2}, 2^{l/2})$ [31].

C. Chaos Function

Chaos is the aperiodic behavior observed in dynamic systems that exhibits significant sensitivity dependence on initial conditions. The concept of chaos pertains to the mapping of sequences generated under specific initial conditions and chaos parameters [22]. Chaos functions exhibit aperiodic, deterministic, and highly sensitive dependence on initial conditions. Furthermore, sequences generated by chaos functions display characteristics akin to random sequences [32]. The study of the properties of chaos functions is typically carried out by examining bifurcation diagrams and Lyapunov exponents [20].

1) Bifurcation Diagram: A bifurcation diagram illustrates the long-term output values resulting from the continuous iteration of an input parameter. The bifurcation diagram depicts the states of the long-term output within various ranges of control parameters. When a control parameter enters the chaotic region, the bifurcation diagram reveals its bifurcations. As the control parameter space becomes more chaotic, the more bifurcations are displayed in the bifurcation diagram [33].

2) Lyapunov Exponent: The sensitivity to small changes in initial conditions and control parameters as inputs plays a crucial role in chaos-based cryptography. The Lyapunov Exponent (LE) is a mathematical indicator that can be computed for different initial conditions and control parameters. A chaotic function with a high positive LE value exhibits rapid divergence of chaotic points in its iterations or over time

[1]The diagram is taken from http://sponge.noekeon.org and is available under the Creative Commons Attribution License.

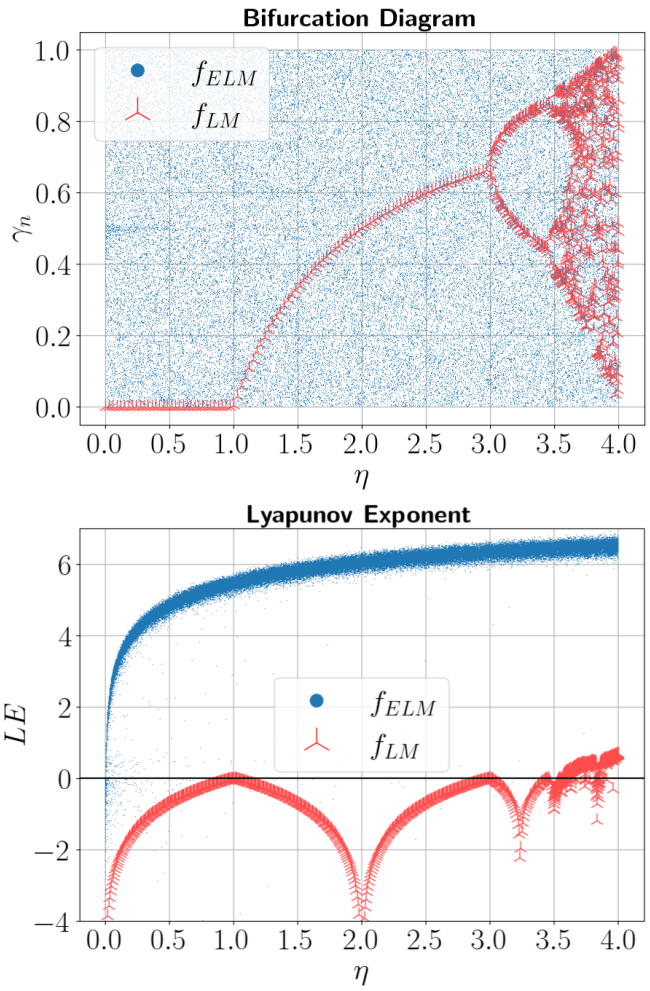

Fig. 2. Bifurcation diagram (top) and Lyapunov exponent (bottom) for the Logistic Map and ELM function with $k = 10$

compared to a function with a smaller LE value. Conversely, a negative or zero LE value indicates periodic behavior or convergence to a value within a few iterations. In other words, an increase in the LE value of a chaotic function signifies an increase in uncertainty and sensitivity in the behavior of the chaotic function [25].

3) Enhanced Logistic Map: One example of a chaotic function, as introduced by [25] in their research, is the Enhanced Logistic Map (ELM), which exhibits superior chaotic properties compared to some other chaotic functions. ELM is an extension of the chaotic Logistic Map (LM) function defined as $\gamma_{n+1} = f_{LM}(\eta, \gamma_n) = \eta\gamma_n(1 - \gamma_n)$ [34]. This is proven by the bifurcation diagram results and LE of the ELM, which show a substantial difference in chaos compared to the logistic map. The ELM is defined as follows:

$$\gamma_{n+1} = f_{ELM}(\eta, \gamma_n, k) = \left(\frac{2^k}{2 f_{LM}(\eta, \gamma_n)}\right) \qquad (1)$$

where γ_n represents the state variable in the system with $\gamma_n \in (0, 1), \forall n \geq 0, k$ and η are system parameters. Figure 2 depicts the bifurcation diagram and Lyapunov exponent of LM and ELM with $k = 10$. It is evident from Figure 2 that the bifurcation diagram of the ELM exhibits more chaotic behavior compared to the LM, attributed to a greater number of bifurcations in the bifurcation diagram of the ELM in comparison to the LM. Furthermore, in the LM, the

maximum LE value is 0.212, and the range of the chaotic region is $\eta \in$[3.57, 4]. In contrast, the maximum LE value in the ELM is 4.121, and the range of the chaotic region is $\eta \in$[0, 4]. This indicates that the chaotic nature of the ELM is superior to that of the LM, both in terms of the range of its chaotic behavior and the characteristics of chaos, as observed through the LE values.

D. Linear Regression and Simple Exponential Regression

Regression is the study of dependency [35]. Linear regression in statistics is the process of determining the best line that can represent the general trend of a set of data. The simplest form of linear regression involves two variables: $\hat{y}$ as the dependent variable and x as the independent variable. The equation for a dataset portraying linear progression is expressed as $\hat{y} = ax + b$. The values of a and b can be obtained as follows:

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=i}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$
$$b = \frac{\sum_{i=1}^{n} y_i - a\left(\sum_{i=1}^{n} x_i\right)}{n}$$

where x_i and y_i denote dependent and independent data correspondingly, and n represents the number of data points.

Conversely, in the context of exponential growth evident within a dataset, the corresponding formulation takes the form of $\hat{y} = ab^x$. The equation can be rewritten as

$$\hat{y} = ab^x$$
$$\ln \hat{y} = \ln\left(ab^x\right)$$
$$\ln \hat{y} = \ln a + x \ln b$$

Let $\hat{Y} = \ln \hat{y}$, $\hat{a} = \ln b$, and $\hat{b} = \ln a$, then the previous equation can be expressed as

$$\hat{Y} = \hat{a}x + \hat{b}$$

Thus, we obtain

$$\hat{a} = \frac{n\Sigma_{i=1}^{n} x_i Y_i - \left(\Sigma_{i=1}^{n} x_i\right)\left(\Sigma_{i=1}^{n} Y_i\right)}{n\Sigma_{i=1}^{n} x_i^2 - \left(\Sigma_{i=1}^{n} x_i\right)^2}$$
$$\ln b = \frac{n\sum_{i=1}^{n} x_i \ln y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} \ln y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$
$$b = \exp\left(\frac{n\sum\limits_{i=1}^{n} x_i \ln y_i - \left(\sum\limits_{i=1}^{n} x_i\right)\left(\sum\limits_{i=1}^{n} \ln y_i\right)}{n\sum\limits_{i=1}^{n} x_i^2 - \left(\sum\limits_{i=1}^{n} x_i\right)^2}\right) \quad (2)$$

and

$$\hat{b} = \frac{\Sigma_{i=1}^{n} Y_i - \hat{a}\left(\Sigma_{i=1}^{n} x_i\right)}{n}$$
$$\ln a = \frac{\sum_{i=1}^{n} \ln y_i - \ln b\left(\sum_{i=1}^{n} x_i\right)}{n}$$
$$a = \exp\left(\frac{\sum_{i=1}^{n} \ln y_i - \ln b\left(\sum_{i=1}^{n} x_i\right)}{n}\right) \quad (3)$$

Consequently, if given data exhibits exponential growth, the relationship describing this growth can be formulated as $y = ab^x$ where a can be obtained using the formula in Equation (3), and b can be obtained using the formula in Equation (2).

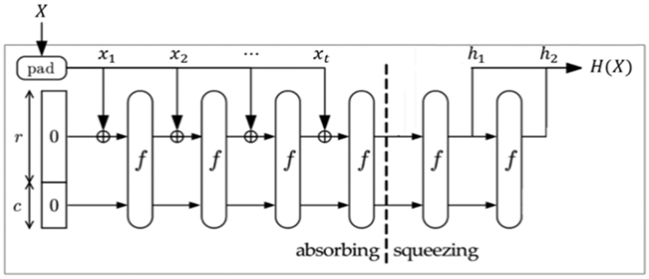

Fig. 3. Sponge Function Scheme in Hortex

Algorithm 1 Hortex

Input: Message X with an arbitrary length.
Output: $H(X)$, the 128-bit hash value of message X.

1: Let $r = 64$ and $c = 192$ represent the parameters used in the sponge function.
2: **if** $r \nmid len(X)$ **then** ▷ Initialization phase
3: $X = X \parallel pad(X)$
4: Divide X into t blocks of 64-bit denoted as $x_1, x_2, ..., x_t$.
5: $s_0 = 0^r \parallel 0^c$
6: **for** $i = 1$ to t **do** ▷ Absorbing phase
7: $s_i = f(s_{i-1} \oplus (x_i || 0^c))$
8: **for** $j = 1$ to 2 **do** ▷ Squeezing phase
9: $s_{t+j} = f(s_{t+j-1})$
10: $h_j = r$ *msb* of s_{t+j}
11: $H(X) = h_1 \parallel h_2$.

IV. The Design of Hortex

Hortex is a hash function algorithm based on chaos functions and the sponge function proposed in this paper. Hortex utilizes the sponge function as its fundamental construction. It takes an input message X of arbitrary size and produces an output $H(X)$ with a size of 128 bits. The algorithm maintains a 256-bit state s, which is composed of a 64-bit rate (r) and a 192-bit capacity (c). Figure 3 depicts the Hortex algorithm's sponge function scheme.

A. Hortex Algorithm Description

As depicted in Figure 3, there are three phases in the generation of the hash value of the Hortex algorithm, commencing with the initialization phase, followed by the absorbing phase, and concluding with the squeezing phase. The Hortex algorithm is described in Algorithm 1. In the initialization phase, the initial state value (s_0) is an empty string, or $s_0 = 0^r \parallel 0^c$, where r and c represent the rate and capacity. The input message X, which can be of an arbitrary size, may have a length that is not a multiple of the rate. Therefore, padding is required for the initial message. Padding is performed by appending a '`1`' bit followed by '`0`' bits to fill the deficiency needed to obtain a message with a multiple of r. The input message, after padding (if necessary), is then divided into blocks x_i of size r, comprising t blocks that will be processed within the transformation function f during the absorbing phase. The padded message is denoted as $X \parallel pad(X)$.

In the absorbing phase, the process involves the extraction of message blocks x_i, each of size r bits or 64 bits, to update the value of a 256-bit state (s_0). This is achieved

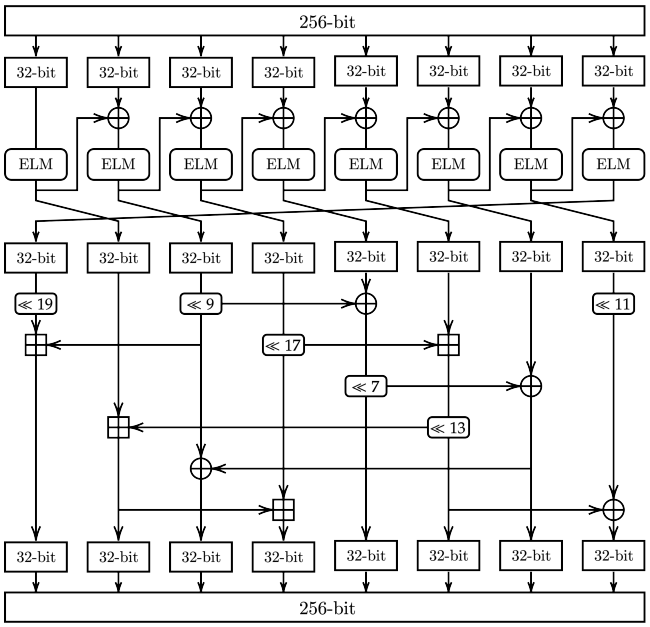

Fig. 4. The scheme of the transformation function f in the Hortex algorithm

Algorithm 2 f function

Input: 256-bit string x.
Output: 256-bit string y.

Divide x into 8 blocks of 32-bit denoted as $x_1, x_2, \ldots, x_8$.
Let $v_1, v_2, \ldots, v_8$ denote binary strings consisting of zeros.
for $i = 1$ to 8 **do** ▷ Chaos ELM algorithm
  if $i = 8$ **then**
    $v_1 = ELM(x_i \oplus v_i)$
  else
    $v_{i+1} = ELM(x_i \oplus v_i)$
Do the following in sequence: ▷ ARX function
$v_1 = (v_1 \ll 19) \boxplus (v_3 \ll 9)$
$v_5 = (v_5 \oplus (v_3 \ll 9)) \ll 7$
$v_6 = (v_6 \oplus (v_4 \ll 17)) \ll 13$
$v_7 = v_7 \boxplus v_5$
$v_8 = (v_8 \ll 11) \oplus v_6$
$v_2 = v_2 \boxplus v_6$
$v_3 = (v_3 \ll 9) \oplus v_7$
$v_4 = (v_4 \ll 17) \boxplus v_2$
$y = v_1 \parallel v_2 \parallel \ldots \parallel v_8$

by performing an XOR operation between the 64 *msb* of the state and the message block x_i. The updated state is then inputted into the transformation function f to obtain the updated state value (s_i). The transformation function f comprises two components: the chaos ELM algorithm and the ARX function (addition rotation XOR).

The value of the state after the absorbent phase is denoted as s_t. The state s_t is then processed in the squeezing phase twice to obtain the output of the hash function Hortex. In the squeezing phase, the state s_t is fed back into the transformation function f, yielding s_{t+1}, and the r *msb* of s_{t+1} is stored as h_1. Subsequently, s_{t+1} is also fed back into f to generate s_{t+2}, and the r *msb* of s_{t+2} is stored as h_2. The final hash value of the Hortex algorithm is expressed as $H(X) = h_1 \parallel h_2$.

In the absorbing and squeezing phases, there exists a transformation function f, whose schematic is illustrated in Figure 4. The transformation function f takes an input of 256-bit size and produces an output of 256-bit size. Algorithm 2 specifies the steps for the transformation function f in the Hortex algorithm. A 256-bit block is divided into 8×32-bit blocks in the function f, and each of them is processed in the ELM algorithm. The output of the ELM algorithm is stored in the adjacent block, which is also utilized to provide feedback to the preceding ELM block. Subsequently, the output of the ELM algorithm undergoes an ARX function within each block, and the outcomes of these ARX functions are then concatenated to form the updated state value.

The ELM algorithm processes 32-bit inputs and produces a 32-bit output. The ELM algorithm utilizes the chaos function ELM introduced by [25] to impart a randomness effect on the Hortex algorithm, as illustrated in Figure 5. The steps of the ELM algorithm are described in Algorithm 3. Parameters γ_0, η, and k are obtained by dividing a 32-bit input into 12 *msb*, followed by the next 16 bits after 12 *msb*, and the remaining 4 *lsb* denoted as x_l, x_m, and x_r, respectively. The variables x_l, x_m, and x_r sequentially determine the values of γ_0, η, and k. Due to the constraint $\gamma_0 \in [0, 1]$ and the size of x_l being 12 bits, a numerical range partition is created with equal intervals of 2^{12} within the range [0,1]. The decimal representation of x_l determines the partition position chosen as γ_0 within the range [0,1]. For example, if the equal interval partition in the range [0,1] yields a sequence of numbers $(a_0, a_1, ..., a_d, ..., a_{2^{12}-1})$, and the decimal representation of x_l is d, then $\gamma_0 = a_d$. The same process is applied to η and k, with the chosen range for η being [2,4], divided into 2^{16} partitions (x_m having a length of 16 bits), and the range for k being [10.01,11.01], divided into 2^4 partitions (x_r having a length of 4 bits). The decimal values of x_m and x_r determine the partition positions chosen as the values for η and k, respectively.

Using Equation 1, the chaos function ELM will undergo n iterations, where $n = \lfloor 6\gamma_0 \rfloor$. The real numbers resulting from the chaos ELM function will be converted into a binary string using a single precision floating point in the IEEE 754 standard, denoted as $binary32(\cdot)$ [36]. The output of the ELM algorithm is the XOR result of the iteration at step $(n+1)$ multiplied by 10^{10}, processed through the $binary32(\cdot)$ function, then left-rotated by 17 bits, and the iteration at step $(n+2)$ processed through the $binary32(\cdot)$ function.

B. Design Rationale

Algorithm 3 contains the utilization or implementation of the chaos function in the Hortex algorithm. This algorithm is an adoption and modification of research that generalizes the implementation of the chaos function in primitive cryptography, including hash functions, by [37]. The adoption and modification are aimed at meeting the design requirements of the hash function, namely having good randomness and collision resistance properties.

The ELM algorithm, as depicted in Algorithm 3, is designed in such a way as to have dependencies on the received input, which is a 32-bit string. This dependency is not only about determining the initial conditions of the

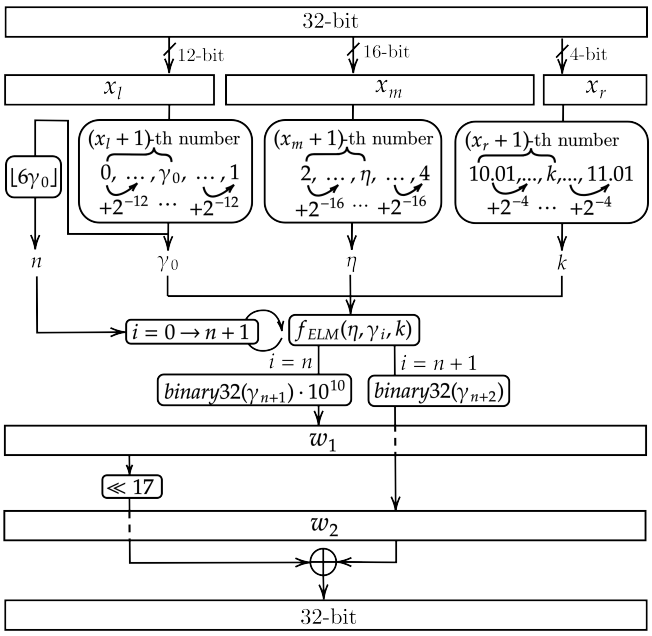

Fig. 5. The ELM Scheme in the Hortex Algorithm

variable parameters of ELM (γ_0) but also determining the system parameters of ELM (η) each time the ELM algorithm is called. This is done because the system parameters of ELM have a sufficiently large chaotic characteristic range, that is, in the range $[0, 4]$. Therefore, it is better to utilize the entire range to obtain more diverse chaotic values. Furthermore, the range selected for the system parameter is $\eta \in [2, 4]$ from the overall range $[0, 4]$. This is done to achieve a high chaotic property because, in this range, the LE value is greater than 4, which is considered significant. Then, instead of the integer value $k = 10$, as suggested by [25], the control parameter of the system k is chosen to be in the range $[10.01, 11.01]$. This is done to ensure that the output of the chaos function ELM is nonzero even when the received input is an empty string, and it establishes a dependency on the value of k with the input.

Equation (1) utilized in the ELM algorithm is executed for a total of $(n + 2)$ iterations, where $n = \lfloor 6\gamma_0 \rfloor$. Due to the highly chaotic characteristics of the employed chaos ELM function, the number of iterations performed need not be excessively large. Opting for a relatively small number of iterations is chosen to expedite the algorithm execution process. The conversion of real numbers to binary strings using IEEE 754 single-precision in the ELM algorithm is necessitated by the computer system's storage of real numbers in the IEEE 754 standard, eliminating the need for additional conversion overhead and concurrently saving execution time. The selection of the IEEE 754 standard is also motivated by its ability to preserve the chaotic characteristics of real values generated by the chaos function during conversion to binary strings.

The transformation function f employed in the Hortex algorithm comprises two main components: the chaos ELM algorithm, as illustrated in Algorithm 3, and the ARX function. The incorporation of the chaos function is evident due to its chaotic nature, making it suitable for constructing a hash function. Additionally, the chaos ELM function that is

Algorithm 3 ELM

Input: 32-bit string x
Output: 32-bit string y

1: Partition x sequentially into blocks of sizes 12 bits, 16 bits, and 4 bits, denoted as x_l, x_m and x_r, respectively.
2: Calculate $\gamma_0 = x_l \cdot \frac{1}{2^{12}}$, $\eta = \left(x_m \cdot \frac{2}{2^{16}}\right) + 2$, $k = \left(x_r \cdot \frac{1}{2^4}\right) + 10.01$, and $n = \lfloor 6\gamma_0 \rfloor$
3: **for** $i = 0$ to $n + 1$ **do**
4: $\quad \gamma_{i+1} = f_{ELM}(\eta, \gamma_i, k)$ ▷ Refer to Equation (1)
5: $\quad$ **if** $i = n$ **then**
6: $\quad\quad w_1 = binary32(\gamma_{n+1} \cdot 10^{10})$
7: $\quad$ **else if** $i = n + 1$ **then**
8: $\quad\quad w_2 = binary32(\gamma_{n+2})$
9: $y = (w_1 \ll 17) \oplus w_2$

used in the ELM algorithm is chosen for its superior chaotic performance compared to other chaos functions [16], [22], [25]. In addition to its simple and fast structure, as mentioned in [38], the ARX function is utilized to disseminate the chaotic characteristics emitted by each ELM algorithm throughout the state. Furthermore, a feedback mechanism is applied in the ELM algorithm within the transformation function f. This is done to ensure that the outputs of each ELM algorithm block differ even when all input values to the ELM algorithm blocks are identical.

The selection of the sponge construction for the function is motivated by its advantages over other hash function constructions, such as simplicity, flexibility, and variable-length output [39]. The simplicity of the general scheme of the sponge function is straightforward, comprising only two phases: absorbing and squeezing, which involve a transformation function f. The flexibility of the sponge function is a trade-off related to the security level and computational speed. The sponge function can achieve high-security levels by sacrificing computational speed, or vice versa. This can be easily controlled through parameters such as state, rate, and capacity defined in the sponge function. The advantage of variable-length output in the sponge function lies in its ability to produce output as needed without altering the predefined parameters of state, rate, and capacity. In other words, a single sponge function can be configured to generate different output lengths.

Furthermore, observe in Figure 3, there is a gap at the beginning of the squeezing phase, one state before producing h_1, which differs from Figure 1. This is done to ensure that the transformation function f is executed twice (i.e., f is applied for two rounds) in a single execution. Thus, it is ensured that each time the Hortex algorithm is executed, the function f two-round is executed exactly once. This is done to minimize the execution of the function f as much as possible to save execution time, so that not every execution of the function f will be run twice in each block, but only in the transition from the absorbing phase to the squeezing phase. The reason for performing the function f two-round is because the function f one-round does not have sufficiently good randomness characteristics, so there is concern that it may perform poorly on the overall Hortex algorithm. However, the function f two-round has good randomness characteristics, so by running the function f two-round in

the Hortex algorithm, it will have a positive impact on the overall Hortex algorithm (refer to Section V-A2 for a more detailed explanation).

The security level of the hash function constructed using the sponge function against collision resistance is given by $\min(2^{c/2}, 2^{n/2})$, where n is the output length and c is the capacity length [31]. The choice of parameters $r = 64$, $c = 192$, and $n = 128$ in the sponge function used in the Hortex algorithm ensures that the security against collision resistance is not dependent on internal collisions $(2^{c/2})$, but rather on full collisions $(2^{n/2})$, such as $n < c$, resulting in $\min(2^{c/2}, 2^{n/2}) = 2^{n/2}$. Thus, the maximum security level achievable by the Hortex algorithm against collision resistance is 2^{64}, and against preimage and second-preimage resistance is 2^{128}.

C. Test Vector

A test vector was executed for each intermediate step in the Hortex hashing process. A 96-bit message ($X =$ `abcd1234bcd4517aabc2efd2`) underwent hashing to produce an 128-bit hash value. Table I presents the test vector for the Hortex algorithm. The first string highlighted in red corresponds to h_1 generated during the squeezing phase, while the second string highlighted in red represents h_2. It can be seen that the hash value of the message X is a concatenation of h_1 and h_2, denoted as $h_1 \parallel h_2$, according to the description of the Hortex algorithm.

V. Randomness Evaluation

Randomness testing of the Hortex algorithm is carried out on both the general algorithm and its constituent components. The randomness assessment of the Hortex algorithm employs the CRT proposed by [26] in their four tests: Strict Avalanche Criterion (SAC) Test, Linear Span Test, Collision Test, and Coverage Test. The test is conducted using the chi-squared goodness-of-fit test with 2^{20} independent input samples, each consisting of a 32-bit string, with a significance level of $\alpha = 0.01$. Each test is repeated five times to ensure statistically independent insights, minimizing the influence of any single outlier or anomaly on the overall results. By conducting multiple iterations, the reliability and consistency of the findings are enhanced, allowing for a more robust evaluation of the algorithm's performance. This approach also reduces the risk of bias and provides a clearer understanding of the underlying trends and patterns. The evaluation of the constituent components of the Hortex algorithm includes the SAC test on the transformation function f and the ELM algorithm. For a more detailed description of the steps undertaken in conducting CRT on the Hortex algorithm, please refer to Appendix A.

A. SAC Test

The objective of the SAC test is to assess whether an algorithm satisfies the SAC property. The SAC property is defined so that a change in one input bit results in a change in the output bit with a probability of 0.5. In other words, SAC evaluates whether an algorithm exhibits a correlation between input and output bits.

There are two conditions that must be met to determine whether an algorithm passes the SAC test. The **first condition** is that $p_{value} \geq \alpha = 0.01$ in the approach using the χ^2 goodness of fit test with degrees of freedom $df = 4$, and the **second condition** is that the entry of the SAC matrix should not fall outside the interval[2] [521783.48 , 526792.52]. The second condition is used to determine whether the results of the first condition are coincidental or not, assuming that the correct procedures are followed.

1) SAC Test on Hortex Algorithm: The results of the SAC test on the Hortex algorithm are presented in Table II. It can be seen that the p_{value} of Hortex exceeds its significance level and that there are no entries in the SAC matrix that fall outside the interval. This satisfies the two conditions required to declare that the SAC test is passed. The five successful tests in Hortex indicate that the SAC test results are not coincidental, demonstrating that Hortex possesses the random mapping property in terms of the cryptographic nature of SAC.

2) SAC Test on f function: The results of the SAC test on the one-round f function and the two-round f function are presented in Table III. It can be seen that the p_{value} for the one-round function f is significantly lower than its significance level, and there are no entries in the SAC matrix outside the interval. Thus, it does not satisfy the first condition required to pass the SAC test. The results of the five tests for the one-round f function consistently demonstrate similar results, indicating that the one-round f function does not pass the SAC test.

However, the results of the SAC test for the two-round function f show the opposite. The p_{value} for the two-round f function is greater than its significance level, and there are no entries in the SAC matrix outside the interval. This satisfies both conditions required to be considered as passing the SAC test. Due to the consistent results of the five tests conducted on the two-round f function, it can be concluded that the two-round f function passes the SAC test.

Based on the results obtained on the randomness of the f function for both the one-round f function and the two-round f function, it is evident that the two-round f function exhibits superior randomness characteristics. This observation forms the basis for the design choice in the Hortex algorithm, where there is a gap in the transition between the absorption and squeezing phases. As a consequence of this gap, the Hortex algorithm executes the f function twice, equivalent to running the two-round f function, ensuring the execution of a function with superior randomness properties.

3) SAC Test on ELM algorithm: The results of the SAC test on the ELM algorithm are presented in Table IV. It is observed that the p_{value} of the ELM algorithm exceeds the significance level and that there are no entries in the SAC matrix outside the interval. This satisfies the two conditions required to declare that the SAC test is passed. The results of the five tests on the ELM algorithm also demonstrate similar results, indicating that the ELM algorithm passes the SAC test. Consequently, the ELM algorithm exhibits favorable random mapping properties, as evidenced by its

[2]The actual interval stated by [26] is $[2^{19}-5009\ ,\ 2^{19}+5009]$, but based on unpublished observations by Riyanti in her thesis at the National Crypto and Cyber Polytechnic in 2016, the more accurate interval is [521783.48 , 526792.52]

TABLE I
TEST VECTOR FOR HORTEX

Initialization Phase		
$X =$ abcd1234bcd4517aabc2efd2		
$X \parallel pad(X) =$ abcd1234bcd4517aabc2efd280000000		
$x_1 =$ abcd1234bcd4517a		
$x_2 =$ abc2efd280000000		
Absorbing Phase		
abcd1234bcd4517a0000000000000000 00000000000000000000000000000000	f	28bec327f35f3fb20b79a72bf9fe3f98 96e278d6e44dd7bae3a3ad9e6fa8939f
837c2cf5735f3fb20b79a72bf9fe3f98 96e278d6e44dd7bae3a3ad9e6fa8939f	f	1145ec5e14a2821f6f8f46ea91fc8bfa 07b6ad379379cc0ceb2476fd11281c28
Squeezing Phase		
1145ec5e14a2821f6f8f46ea91fc8bfa 07b6ad379379cc0ceb2476fd11281c28	f	8ac693947ff82936bf3facc68b69d5ce f4e870c81b8bdabd0a82dbd49c3f058f
8ac693947ff82936bf3facc68b69d5ce f4e870c81b8bdabd0a82dbd49c3f058f	f	e1d54a1b837b31984bdd6779974d63d9 f9fbdcf5e02907d001d3d7d65381e70a
Hash Value		
$Hortex(X) =$ 8ac693947ff82936e1d54a1b837b3198		

TABLE II
THE OUTPUT OF FIVE SAC TESTS ON HORTEX ALGORITHM

SAC test	First condition		Second condition	
	χ^2	p_{value}	inside interval	outside interval
I	6.2234	0.1830	4096	0
II	10.0770	0.0391	4096	0
III	7.1764	0.1268	4096	0
IV	9.8815	0.0424	4096	0
V	7.0501	0.1332	4096	0

cryptographic SAC characteristics.

B. Linear Span Test

Nonlinearity is a fundamental design aspect of primitive cryptography, including hash functions. To assess the randomness of the hash function based on non-linearity, a linear span test is employed. An algorithm is deemed to pass the linear span test when it satisfies condition $p_{value} \geq \alpha = 0.01$ in the approach using the χ^2 goodness of fit test with degrees of freedom $df = 2$. The results of the linear span test for the Hortex algorithm are presented in Table V. It can be seen that all five tests conducted have a p_{value} greater than the specified significance level, indicating that the Hortex algorithm passes the linear span test. Therefore, it can be concluded that the Hortex algorithm exhibits a random mapping concerning the non-linearity properties of cryptographic nature.

C. Collision Test

The objective of the collision test is to evaluate the randomness of an algorithm by considering its collision resistance properties. The conducted collision test focuses on collisions occurring in a subset of output bits (near collisions). In other words, this test aims to assess the resistance of an algorithm to near collisions. The collision test involves modifying all possible 12-bit combinations in the input message and observing 16 *msb* of the output to detect collisions. An algorithm is deemed to pass the collision test when it satisfies the condition $p_{value} \geq \alpha = 0.01$ in the approach using the χ^2 goodness-of-fit test with degrees of freedom $df = 4$.

The results of this collision test are presented in Table VI. It can be observed that all five tests conducted have p_{value} greater than the specified significance level, indicating that the Hortex algorithm passes the collision test. Therefore, it can be concluded that the Hortex algorithm exhibits random mapping characteristics concerning the cryptographic properties of near collisions.

D. Coverage Test

The objective of test coverage is to evaluate the randomness of an algorithm by considering the distribution of the output set relative to the input set (coverage). In order for a hash function to be deemed random, it must have an expected coverage of 63% of the input set. Coverage testing is performed by modifying all possible 12 *msb* of the input to observe the corresponding 12 *msb* in the resulting output.An algorithm is deemed to pass the coverage test when it satisfies the condition $p_{value} \geq \alpha = 0.01$ in the approach using the χ^2 goodness-of-fit test with degrees of freedom $df = 4$.

The results of the coverage test on the Hortex algorithm are presented in Table VII. It can be seen that all five tests conducted have p_{value} greater than the specified significance level, indicating that the Hortex algorithm passes the coverage test. Therefore, it can be concluded that the Hortex algorithm has a coverage of at least 63%, demonstrating its random mapping property.

VI. SECURITY EVALUATION

Three security evaluations will be performed on the Hortex algorithm. The first involves a theoretical security assessment to determine the overall security level of the Hortex. This assessment will provide insight into the fundamental strengths and potential vulnerabilities of the algorithm.

The second evaluation focuses on assessing the quality of the confusion and diffusion properties of the Hortex algorithm. This is carried out because, as highlighted in [40], confusion and diffusion are essential properties of a secure cipher in cryptography. Hence, it is crucial to evaluate how well the Hortex algorithm exhibits these characteristics. Theoretically, a good hash function must demonstrate sensitivity to minor changes in the input message. Specifically, any small alteration in the input should result in a 50% difference

TABLE III
THE OUTPUT OF FIVE SAC TESTS ON THE ONE-ROUND AND TWO-ROUND f FUNCTION

SAC test	One-round f function				Two-round f function			
	First condition		Second condition		First condition		Second condition	
	χ^2	p_{value}	inside interval	outside interval	χ^2	p_{value}	inside interval	outside interval
I	65.4	2.0×10^{-13}	65535	0	1.35	0.851	65535	0
II	207	7.2×10^{-44}	65535	0	1.21	0.874	65535	0
III	64.0	4.1×10^{-13}	65535	0	3.43	0.488	65535	0
IV	240	5.7×10^{-51}	65535	0	1.39	0.845	65535	0
V	29.9	5.0×10^{-6}	65535	0	2.41	0.660	65535	0

TABLE IV
THE OUTPUT OF FIVE SAC TESTS ON THE ELM ALGORITHM

SAC test	First condition		Second condition	
	χ^2	p_{value}	inside interval	outside interval
I	2.9125	0.5725	1024	0
II	4.6550	0.3245	1024	0
III	0.2088	0.9949	1024	0
IV	2.8164	0.5889	1024	0
V	5.3129	0.2566	1024	0

TABLE V
THE OUTPUT OF FIVE LINEAR SPAN TESTS ON HORTEX ALGORITHM

Linear span test	χ^2	p_{value}
I	0.6865	0.7094
II	2.1382	0.3433
III	0.0741	0.9636
IV	3.8334	0.1470
V	1.4493	0.4844

TABLE VI
THE OUTPUT OF FIVE COLLISION TESTS ON HORTEX ALGORITHM

Collision test	χ^2	p_{value}
I	3.2305	0.5200
II	9.5165	0.0494
III	5.9227	0.2049
IV	2.5306	0.6391
V	4.5009	0.3424

TABLE VII
THE OUTPUT OF FIVE COVERAGE TESTS ON HORTEX ALGORITHM

Coverage test	χ^2	p_{value}
I	3.2310	0.5199
II	3.6043	0.4621
III	9.5184	0.0493
IV	13.1197	0.0107
V	6.4254	0.1695

in the output. Ensuring this behavior is vital for the security and reliability of the hashing process.

The third evaluation focuses on analyzing the collision resistance through an attack algorithm. To achieve this, Yuval's birthday attack will be theoretically employed due to limitations in the available hardware, which prevent a practical implementation. The emphasis on attacking collision resistance stems from the fact that this property generally exhibits a lower security threshold compared to preimage and second preimage resistance in hash functions. If an adversary can breach the collision resistance of a hash function, it indicates a significant compromise of the function's security, even if the preimage and second preimage resistance remain intact. By targeting the collision resistance properties, we aim to highlight the potential weaknesses and overall reliability of the Hortex algorithm under theoretical attack scenarios. This comprehensive evaluation will enhance our understanding of Hortex's resilience against cryptographic attacks.

A. Theoretical Security

The theoretical security analysis of the Hortex algorithm aims to evaluate the algorithm's resilience against brute-force attacks by focusing on three fundamental security properties: preimage resistance, second preimage resistance, and collision resistance. The Hortex algorithm generates a 128-bit hash output and is built using a sponge function, which operates with the following parameters: a state (s) of 256 bits, a rate (r) of 64 bits, and a capacity (c) of 192 bits.

Given that the Hortex algorithm is based on sponge construction, its security against brute-force attacks can be assessed through these properties. For collision resistance, the security is determined by $\min(2^{c/2}, 2^{n/2})$ [31], where n is the output length of the algorithm (128 bits). For preimage and second preimage resistance, the security level is 2^n. Since $n = 128$ and $c = 192$, the result of $\min(2^{c/2}, 2^{n/2})$ is $2^{n/2}$, resulting in a security level of 2^{64} for collision resistance. Meanwhile, the security levels for the preimage and second preimage resistance are both 2^{128}.

Thus, the maximum security level the Hortex algorithm achieves is 2^{64} against collision attacks and 2^{128} against preimage and second preimage attacks, reflecting its robustness across these critical cryptographic properties.

B. Statistical Analysis of Confusion and Diffusion

To evaluate the confusion and diffusion properties, the following experiment was conducted: First, a random input value was selected, and its corresponding output value was computed. Then, one bit of the initial input value was randomly altered, and the output value was recalculated. The two resulting output values were compared, and the number of differing bits was counted and recorded as B_i [41]. For this experiment, a 32-bit input length was used, and the process was repeated N times. This experiment was applied to the Hortex algorithm and its components, specifically the f-function and the ELM algorithm. Notably, for the f-function a 256-bit input was utilized.

TABLE VIII
STATISTICAL ANALYSIS OF CONFUSION AND DIFFUSION ON HORTEX ALGORITHM

Parameter	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	Mean
B_{min}	53	53	53	53	53
B_{max}	75	75	75	75	75
$\overline{B}$	63.26	63.35	63.38	63.38	63.34
$P(\%)$	49.42	49.49	49.52	49.51	49.48
ΔB	4.86	4.96	4.95	4.96	4.93
$\Delta P(\%)$	3.80	3.87	3.87	3.87	3.85

TABLE IX
STATISTICAL ANALYSIS OF CONFUSION AND DIFFUSION ON f-FUNCTION

Parameter	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	Mean
B_{min}	107	107	107	107	107
B_{max}	149	149	149	149	149
$\overline{B}$	127.35	127.41	127.50	127.48	127.44
$P(\%)$	49.75	49.77	49.81	49.80	49.78
ΔB	9.03	9.13	9.29	9.28	9.18
$\Delta P(\%)$	3.53	3.57	3.63	3.63	3.59

The evaluation of confusion and diffusion properties in cryptography is typically assessed using six statistical measures:

- Minimum number of bits changed:

$$B_{min} = min\{B_1, B_2, \ldots, B_N\}$$

- Maximum number of bits changed:

$$B_{max} = max\{B_1, B_2, \ldots, B_N\}$$

- Mean number of bits changed:

$$\overline{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$$

- Mean changed probability:

$$P = \frac{\overline{B}}{l} \times 100\%$$

- Standard deviation of the changed bit number:

$$\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=i}^{N}\left(B_i - \overline{B}\right)^2}$$

- Standard deviation of the changed probability:

$$\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=i}^{N}\left(\frac{B_i}{l} - P\right)^2} \times 100\%$$

where B_i denotes the number of bits changed in the i^{th} test, N represents the number of experiments carried out and l corresponds to the length of the output value.

Tables VIII, IX, and X present the results of experiments conducted on the Hortex algorithm, the f-function, and the ELM algorithm, respectively, across various N (number of tries). Based on the data obtained, the Hortex algorithm, the f function and the ELM algorithm demonstrated good performance, with output changes consistently around 50%. Furthermore, the standard deviation of the changed bit number (ΔB) and the standard deviation of the changed probability (ΔP) for each result were relatively small, indicating minimal deviation from the expected value (a 50% change). These low standard deviation values further suggest that the algorithms exhibit good confusion and diffusion properties. Hence, it can be concluded that the Hortex algorithm, the f-function, and the ELM algorithm possess well-defined confusion and diffusion characteristics, meeting the criteria for secure cryptographic functions.

TABLE X
STATISTICAL ANALYSIS OF CONFUSION AND DIFFUSION ON ELM ALGORITHM

Parameter	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	Mean
B_{min}	11	11	11	11	11
B_{max}	23	23	23	23	23
$\overline{B}$	16.18	16.31	16.34	16.35	16.30
$P(\%)$	50.57	50.97	51.07	51.09	50.92
ΔB	2.81	2.72	2.75	2.76	2.76
$\Delta P(\%)$	8.77	8.50	8.59	8.61	8.62

Algorithm 4 Yuval's birthday attack

Input: Original message X_1, fake message X_2, and n-bit hash function $H(\cdot)$.

Output: X_1', X_2', which are the results of minor modifications to X_1 and X_2 with $H(X_1') = H(X_2')$.

1: Generate $k = 2^{n/2}$ minor modifications X_1' of X_1.
2: **for all** X_1' **do**
3: Compute and store $H(X_1')$ (grouped with corresponding message i.e. X_1')
4: `Collision=False`
5: **repeat**
6: Generate minor modifications X_2' of X_2 and compute $H(X_2')$
7: **if** $H(X_2')$ is equal to any $H(X_1')$ **then**
8: `Collision=True`
9: **until** `Collision`

C. Collision Attack

Collision resistance is a property of a hash function that makes it extremely difficult to find two distinct messages that produce the same hash output [42]. A collision attack known as the Yuval's birthday attack will be employed to assess the collision resistance property of the Hortex algorithm. The Yuval's birthday attack utilizes the fourth birthday problem approach with the probability (P) of finding a collision given by

$$P \approx 1 - e^{-k^2/N} \tag{4}$$

where k represents the number of generated elements and N represents the population size [43]. In general, Yuval's birthday attack is illustrated in Algorithm 4 [44], [27]. In essence, Yuval's birthday attack attempts to identify the identical value (collision) between two sets. The first set is the set of minor modifications of the original message, and the second set is the set of minor modifications of the fake message.

Yuval's birthday attack has a complexity of $O(2^{n/2})$, where n is the length of the output of the hash function, in this case, the Hortex algorithm. Yuval's birthday attack introduces minor modifications totaling $2^{n/2}$ or, in the case of

TABLE XI
OBSERVATION OF EXECUTION TIME FOR t-BIT MODIFICATIONS

The number of bits modified (t_i)	Execution time in ms (y_i)	The number of bits modified (t_i)	Execution time in ms (y_i)
15	3726	21	240253
16	7491	22	488274
17	15075	23	1040332
18	30183	24	1918816
19	60682	25	6618324
20	128465		

the Hortex algorithm, equivalent to $2^{128/2} = 2^{64}$. The device employed for Yuval's birthday attack is an Intel Core i7-10700 CPU @ 2.90GHz with 16 GB of RAM. A theoretical approach will be used to determine the feasibility of this collision attack on the basis of computational requirements.

The primary consideration in carrying out this attack is the utilization of the required memory. Software running on a computer system during execution requires temporary storage space. The temporary storage used is Random Access Memory (RAM). The software implementing the source code for Yuval's birthday attack will require memory allocation in the RAM of a computer system. Since the Hortex algorithm has an output size of 128 bits, or equivalently, 16 bytes of memory, each output of the Hortex algorithm during execution requires a memory allocation of 16 bytes in the RAM of a computer system.

In its implementation, Yuval's birthday attack can be approached in two ways: (a) storing both sets of hash values for original and fake messages, or (b) storing only one of the sets. If approach (a) is adopted, the total memory used in 2^t modifications is $2(2^t \cdot 16 \text{ bytes}) = 2^{t+5}$ bytes, or equivalently, $2^{-30} \cdot 2^{t+5} = 2^{t-25}$ GB, with t being the number of bits modified. Then, if approach (b) is chosen, the total required memory is 2^{t-26} GB. Thus, the amount of RAM required for a computer system under approach (a) is 2^{t-25} GB RAM, whereas under approach (b), it is 2^{t-26} GB RAM. In the computation of Yuval's birthday attack on the Hortex algorithm, the worst-case scenario will be considered for the algorithm designer or the best-case scenario for the attacker. Therefore, it will be assumed that the attacker adopts approach (a), requiring a total memory of 2^{t-25} GB RAM for the t bits of modification performed.

In addition to memory usage, the next consideration is the execution time required to complete the execution of Yuval's birthday attack algorithm. An empirical approach will be taken on a smaller number of bit modifications in Yuval's birthday attack to estimate the execution time for other numbers of bit modifications. The empirical approach will be applied to a number of bits, denoted as t_i, where $15 \leq t_i \leq 25$, and the recorded execution time is measured in milliseconds (ms). The results of the empirical approach are presented in Table XI.

Based on the data in Table XI, simple exponential regression will be employed to obtain the exponential equation relating the modified bits (t) to the execution time ($\hat{y}$), expressed as follows: $\hat{y} = a \cdot b^t$. With a data set comprising 11 data points, the values of a and b, calculated using Equations (3) and (2), are found to be 2.055187 and 0.070922, respectively. Thus, the relationship between the number of modified bits and the execution time is given by:

$$\hat{y} = 0.070922 \cdot 2.055187^t \tag{5}$$

where t is the number of modified bits and $\hat{y}$ is the execution time in ms. The correlation coefficient of equation $\hat{y}$ is 0.94, indicating that equation $\hat{y}$ can effectively predict the execution time required for the given number of modified bits in Yuval's birthday attack.

TABLE XII
PREDICTION OF MEMORY, TIME, AND PROBABILITY IN YUVAL'S BIRTHDAY ATTACK ON THE HORTEX ALGORITHM.

t-bit	Memory	Time	Probability
25	1 GB	1.3 hours	$3.30 \cdot 10^{-24}$
26	2 GB	2.6 hours	$1.32 \cdot 10^{-23}$
27	4 GB	5.5 hours	$5.29 \cdot 10^{-23}$
28	8 GB	11.3 hours	$2.11 \cdot 10^{-23}$
29	16 GB	23.2 hours	$8.47 \cdot 10^{-22}$
30	32 GB	47.8 hours	$3.38 \cdot 10^{-21}$
⋮	⋮	⋮	⋮
55	1073 PB	362704 years	$3.81 \cdot 10^{-6}$
56	2147 PB	744131 years	$1.52 \cdot 10^{-5}$
57	4294 PB	1.5 million years	$6.10 \cdot 10^{-5}$
58	8589 PB	3.1 million years	$2.44 \cdot 10^{-4}$
59	17179 PB	6.4 million years	$9.76 \cdot 10^{-4}$
60	34359 PB	13.2 million years	$3.89 \cdot 10^{-3}$
61	68719 PB	27.2 million years	0.0155
62	$1 \cdot 10^6$ PB	56 million years	0.0605
63	$2 \cdot 10^6$ PB	115 million years	0.221
64	$5 \cdot 10^6$ PB	236 million years	0.632

Next, we will examine the probability of collision in Yuval's birthday attack for t-bit modifications. Due to the use of the fourth birthday problem approach in Yuval's birthday attack, the probability of collision can be calculated using Equation (4). In Equation (4) for the Hortex Algorithm $N = 2^{128}$ and $k = 2^t$, the probability of collision in Yuval's birthday attack for t-bit modifications is given by

$$P \approx 1 - e^{-2^{2t-128}} \tag{6}$$

where t is the number of modified bits and P is the probability of collision. Given the derived formulations for memory and execution time as per Equation (5) and the requisite probability in the context of Yuval's birthday attack for t-bit modifications, as expressed in Equation (6), predictions can be made regarding the memory, time, and probability for the scenario where $t = 64$ bits or for the actual Yuval's birthday attack on the Hortex Algorithm. The predicted values for memory, time, and probability are presented in Table XII.

Based on results obtained from theoretical testing of Yuval's attack, the most feasible configuration for an executable attack within reasonable time and memory limits is the 29-bit modification of Yuval's birthday attack. While the execution time and memory requirements are within achievable bounds, the probability of a successful collision is extremely low, quantified as $8.47 \cdot 10^{-22}$. Consequently, to achieve a collision, approximately 10^{21} iterations of collision attacks would be required, necessitating a high volume of repetitive attempts specifically under the 29-bit modification constraint.

In contrast, executing a full-scale Yuval's birthday attack on the Hortex Algorithm ($t = 64$) would demand an immense amount of resources. A device with at least $5 \cdot 10^6$ petabytes (PB) of RAM would be required, yet no such

device currently exists. Additionally, the estimated execution time for this attack is approximately 236 million years. Given these practical limitations, it is effectively impossible to execute this collision attack on the Hortex Algorithm, ensuring its security against Yuval's birthday attack under current resource constraints.

VII. Conclusion

A new algorithm, named Hortex, has been proposed as a hash function that processes inputs of arbitrary length and generates a 128-bit hash value. Hortex is built using a sponge construction with a transformation function f, which integrates the chaotic ELM algorithm and an ARX function. In this design, the sponge function operates in a 256-bit state with a 64-bit rate (r) and a 192-bit capacity (c), ensuring an efficient balance between throughput and security.

Hortex stands out due to its strong cryptographic properties, such as the Strict Avalanche Criterion (SAC), linear span, collision resistance, and coverage. These properties are reinforced by the fact that Hortex successfully passes the Cryptographic Randomness Test (CRT). Furthermore, the two-round transformation function of the algorithm, f, combined with the ELM algorithm, also passes the SAC test, further validating the randomness and unpredictability of its output.

From a theoretical perspective, the Hortex algorithm is secure against attacks such as Yuval's birthday attack due to the infeasibly high computational and time resources required to execute such attacks. Additionally, Hortex demonstrates strong confusion and diffusion properties, which are fundamental criteria for secure cryptographic functions. Furthermore, the main components of the Hortex algorithm, namely the f-function and the ELM algorithm, also exhibit robust confusion and diffusion properties. This reinforces the conclusion that Hortex inherently possesses these essential cryptographic characteristics. In terms of collision resistance, Hortex achieves security up to 2^{164} operations, while its preimage and second-preimage resistance provide security levels of 2^{128}. These features make Hortex a strong candidate for sponge-based hash functions, delivering both efficiency and robust security for a wide range of cryptographic applications.

Appendix A
CRT steps on Hortex

To better understand how the results of the randomness evaluation in Section V were obtained, the steps involved in conducting the CRT on Hortex will be elucidated. The presentation of these steps will cover a subset of the tests performed, providing sufficient information to understand the execution of the randomness evaluation. However, prior to that, it is pertinent to briefly discuss the χ^2 goodness-of-fit test used in the CRT process.

A. χ^2 Goodness of Fit Test

The χ^2 Goodness of Fit Test is a tool used to determine whether the data from a population conforms to the hypothesized population distribution of a specific set of data [45]. This test evaluates a population based on the degree to which the observed frequency (o_i) of events aligns with the expected frequency (e_i) of events predicted by a hypothesized population distribution. The observed frequency of events in the data is referred to as the *observed frequency*, while the *expected frequency* is the anticipated frequency of events in a population based on the hypothesized distribution.

The steps in conducting the χ^2 Goodness of Fit Test are as follows: 1.

1) Establish the initial hypotheses $\mathbf{H}_0$ and $\mathbf{H}_1$.
   $\mathbf{H}_0$: The observed frequencies are equal to the expected frequencies ($o_i = e_i$).
   $\mathbf{H}_1$: The observed frequencies are not equal to the expected frequencies ($o_i \neq e_i$).
2) Determine the significance level α. The significance level α represents the probability of committing an error by rejecting the null hypothesis H_0 when it is true. A significance level of $\alpha = 0.5$ indicates a 5% probability of making a decision to reject H_0 even when it is true. The predetermined significance level α becomes a critical value for drawing conclusions related to observations.
3) Determine the degrees of freedom (df), where the degrees of freedom are the number of classes minus one.
4) Calculate the χ^2 values as follows:
$$e_i = v \cdot p_i \tag{7}$$
$$\chi^2 = \sum_{i=1}^{l} \frac{(o_i - e_i)^2}{e_i} \tag{8}$$
where v represents the number of experiments conducted, p_i denotes the probability in the i-th class, and l represents the number of classes.
5) Make decisions based on the adequacy or inadequacy of the fit between the o_i and e_i. Decisions are made by comparing the p_{value} from the calculated χ^2 with the significance level α. The null hypothesis H_0 is accepted if $p_{value} \geq \alpha$. Conversely, if $p_{value} < \alpha$, the null hypothesis H_0 is rejected, and the alternative hypothesis H_1 is accepted. The p_{value} can be calculated as follows:
$$p_{value} = \frac{\gamma\left(\frac{df}{2}, \frac{\chi^2}{2}\right)}{\Gamma\left(\frac{df}{2}\right)} \tag{9}$$
where γ and Γ represent the lower incomplete gamma function and the gamma function, respectively. p_{value} is often referred to as the Cumulative Distribution Function (CDF) of χ^2 with degree of freedom df.

B. Steps of SAC Test on Hortex

The probabilities of the desired expected frequency to assess randomness, using the SAC test for test subjects, are illustrated in Table XIII as computed by [26]. The SAC test requires an SAC matrix of dimensions $m \times n$, where m and n represent the input and output of the test subjects. Given that the test subject is Hortex, an SAC matrix of size 32×128 will be generated (with a 32-bit input size). The significance level used in the SAC test is $\alpha = 0.01$.

The steps for conducting the SAC test on Hortex, specifically for the first SAC test as shown in Table II, are as follows: 1.

TABLE XIII
RANGES AND PROBABILITIES OF SAC TEST FOR 2^{20} TRIALS

Class	Range	Probability
1	0 - 523857	0.200224
2	523858 - 524158	0.199937
3	524159 - 524417	0.199677
4	524418 - 524718	0.199937
5	524719 - 1048576	0.200224

1) Initialize the SAC matrix $M = \{m^{ij}\}$ with $m^{ij} = 0$ for $i = 1, 2, \ldots, 32$ and $j = 1, 2, \ldots, 128$.
2) Generate independent random input values X_a for a total of 2^{20} with $1 \leq a \leq 2^{20}$, displayed below:

a	1	2	$\cdots$	2^{20}
X_a	c2b6b723	b9c6ee82	$\cdots$	ab974c45

3) Select one message sample from the 2^{20} independent random input samples X_a. Let X_1 be chosen, then calculate its hash value, $Y_1 = H(X_1)$.
$Y_1 = H(X_1) = H(\texttt{c2b6b723}) = \texttt{ea81e48754e252403c00d90a76c7b308}$.
4) Subsequently, for each bit position i of X_1, perform a bit flip and store it as X_1^i, then calculate its hash value and store it as Y_1^i. The values of the modified message X_1^i and the hash of the modified message bits Y_1^i are presented in Table XIV.
5) Perform an XOR operation between Y_1 and Y_1^i, resulting in $Z_1^i = Y_1 \oplus Y_1^i$. The value of Z_1^i can be expressed as a 128-bit bitstream, or $Z_1^i = (Z_1^{i1}, Z_1^{i2}, \ldots, Z_1^{i128})$. After the entire XOR operation is completed, the matrix $Z_1 = \{Z_1^{ij}\}$ is obtained with $1 \leq i \leq 32, 1 \leq j \leq 128$. The set of rows of matrix Z_1, namely Z_1^i, can be observed in Table XIV.
6) Update the SAC matrix M, using the formula $M_{old} = M_{new} + Z_1$.
7) Repeat steps 3 to 6 for the remaining $2^{20} - 1$ samples of other input messages.
8) Group each entry of the final SAC matrix M into class i based on its range and calculate the observed frequency denoted as o_i, where $i = 1, 2, \ldots, 5$.
9) Calculate the expected frequency e_i using Equation (7) with the number of experiments conducted, $v = n \cdot m = 32 \cdot 128 = 4096$, and p_i as the probability of expected frequency in class i according to Table XIII. The observed frequency o_i and the expected frequency e_i resulting from the SAC test in the first test of the Hortex algorithm are presented in Table XV.
10) Obtain the χ^2 value using Equation (8) based on the values of o_i and e_i in Table XV and the p_{value} using Equation (9) with $df = 4$. The obtained χ^2 value is 6.2234, and the p_{value} is 0.1830, as shown in Table II.
11) Conduct testing with the expected interval approach on the results of the SAC matrix obtained. Verify whether there are entries in the SAC matrix M that fall outside the interval [521783.48, 526792.52][2]. The outcome of the verification of entries in the SAC matrix, whether they fall within the specified interval in the first SAC test, can be observed in Table II.
12) Determine whether the null hypothesis $\mathbf{H}_0$ is accepted or rejected based on p_{value} and α. It is known from the previously obtained p_{value} that $p_{value} \geq \alpha$, indicating the acceptance of the null hypothesis $\mathbf{H}_0$. Furthermore, it is also known that there are no entries in the SAC matrix outside the specified interval. Due to the acceptance of the null hypothesis and the absence of SAC matrix entries beyond the specified interval, Hortex passes the SAC test (for the first trial).

TABLE XIV
THE RESULTS OF X_i, Y_1^i, AND Z_i^i FROM THE FIRST SAC TEST ON HORTEX.

i	X_1^i	Y_1^i	Z_1^i
1	c2b6b722	4fa38ebbc22b829f e58eb1e8d21ee404	a5226a3c96c9d0df d98e68e2a4d9570c
2	c2b6b721	06f4045f22e057fe 27d347ac5302e208	ec75e0d8760205be 1bd39ea625c55100
⋮	⋮	⋮	⋮
32	42b6b723	a2450324438a71ba ed21871e723fae2c	48c4e7a3176823fa d1215e1404f81d24

TABLE XV
THE RESULTS OF o_i AND e_i FROM THE FIRST SAC TEST ON HORTEX.

Class	Range	o_i	e_i
1	0 - 523857	825	820.117504
2	523858 - 524158	843	818.941952
3	524159 - 524417	777	817.876992
4	524418 - 524718	862	818.941952
5	524719 - 1048576	789	820.117504

Perform the aforementioned steps an additional four times, resulting in a total of five SAC tests. This is done to ensure that the outcomes of the SAC tests are unbiased and not mere chance occurrences. The results of the five SAC tests conducted on Hortex all passed, as indicated in Table II. The same procedure is applied to the components of Hortex, namely the f function and the ELM algorithm.

C. Steps of Linear Span Test on Hortex

The probabilities of the expected frequency to assess the randomness, using the linear span test for the test subjects, are illustrated in Table XVI as computed by [26]. The testing method for the linear span test is executed by evaluating the rank of an $m \times m$ matrix formed from the output of a hash function. In a single trial, the linear span test requires t linearly independent messages. The size of the input messages is 32-bit, and $t = 5$ linearly independent messages are employed to generate $m = 2^5 = 32$ linearly independent messages. The significance level employed in the linear span test is $\alpha = 0.01$.

The steps for conducting the linear span test on Hortex, specifically for the first linear span test as shown in Table V, are as follows: 1.

1) Conduct experiments on the set of messages X_a for $1 \leq a \leq 2^{20}$. Let X_1 be chosen from the members of the set shown in Table XVII, second column.
2) Generate $m = 2^t = 32$ messages resulting from linear combinations of five linearly independent messages in

TABLE XVI
PROBABILITIES USED IN LINEAR SPAN TEST ($m > 19$)

Class	Rank	Probability
1	$\leq m-2$	0.133636
2	$m-1$	0.577576
3	m	0.288788

TABLE XVII
THE RESULTS OF X_i, X_1^i, AND H_i^i FROM THE FIRST LINEAR SPAN TEST ON HORTEX.

i	X_1	X_1^i	H_1^i
1	(14cf6e47, 5f3093e9, 6a80ba9b, 32b11dd8, 21c98dfd)	00000000	dafc7531c75c25dd 2cf756025a98fdf6
2		14cf6e47	4393b66e410d2c47 1c70c62675ae1316
⋮		⋮	⋮
32		3207d710	7979043478bad1dd d6627835f82992a9

the set X_1. This yields 32 messages from X_1, denoted as X_1^i with $1 \leq i \leq 32$.

3) Compute the hash value of X_1^i and obtain H_1^i for $1 \leq i \leq 32$.
4) Extract the 32-bit value from H_1^i, then arrange it into a 32×32 matrix. The data for X_1^i, H_1^i, and the extraction of the 32-bit H_1^i (highlighted in red) are displayed in Table XVII.
5) Determine the rank of the matrix obtained and tally the results within the classes according to the range of rank values presented in Table XVI.
6) Repeat steps 3 to 6 for the remaining $2^{20}-1$ samples from the input message set X_a.
7) The observed frequency o_i represents the result of the frequency distribution of the rank values obtained in all 2^{20} samples in the set.
8) Calculate the expected frequency e_i using Equation (7) with the number of experiments conducted, $v = 2^{20}$, and p_i as the probability of expected frequency in class i according to Table XVI. The observed frequency o_i and the expected frequency e_i resulting from the linear span test in the first trial of the Hortex algorithm are presented in Table Tab XVIII.
9) Obtain the χ^2 value using Equation (8) based on the values of o_i and e_i in Table XVIII and the p_{value} using Equation (9) with $df = 2$. The obtained χ^2 value is 6.6865, and the p_{value} is 0.7094, as shown in Table V.
10) Determine whether the null hypothesis $\mathbf{H}_0$ is accepted or rejected based on p_{value} and α. It is known from the previously obtained p_{value} that $p_{value} \geq \alpha$, indicating the acceptance of the null hypothesis $\mathbf{H}_0$. Consequently, due to the acceptance of the null hypothesis, Hortex passes the linear span test (for the first trial).

Perform the aforementioned steps four more times, resulting in a total of five linear span tests. This is carried out to ensure that the results of the linear span tests are unbiased. The results of the five linear span tests conducted on Hortex all passed, as indicated in Table V.

TABLE XVIII
THE RESULTS OF o_i AND e_i FROM THE FIRST LINEAR SPAN TEST ON HORTEX.

Class	Rank	o_i	e_i
1	$\leq m-2$	139892	140127.5023
2	$m-1$	605574	605632.3318
3	m	303110	302816.1659

TABLE XIX
RANGES AND PROBABILITIES OF COLLISION TESTS FOR A 16 *msb* OUTPUT OBSERVATION AND 12 *msb* INPUT MODIFICATION.

Class	Range	Probability
1	0 - 116	0.206246
2	117 - 122	0.194005
3	123 - 128	0.219834
4	129 - 134	0.183968
5	134 - 4096	0.195947

D. Steps of Collision Test on Hortex

The probabilities of the expected frequency to assess the randomness, using the collision test for the test subjects, are illustrated in Table XIX as computed by [26]. Two parameters are used in the collision test. The parameter n denotes the number of input alterations performed, while the parameter m signifies the number of collision observations performed on the *msb* output. The designated parameter pair is $(n, m) = (2^{12}, 2^{16})$, with a message input length of 32 bits. The significance level used in the collision test is $\alpha = 0.01$.

The steps for conducting the collision test on Hortex, specifically for the first collision test as shown in Table VI, are as follows: 1.

1) Generate input message samples, denoted as X_a for $1 \leq a \leq 2^{20}$. Assume a selected input message sample, X_1, to be 000f1329.
2) Modify 12 *msb* of the input message X_1 to obtain modified messages, X_1^i, where i ranges from 1 to 2^{12}.
3) Calculate the output message H_1^i for each modified message X_1^i, where $H_1^i = H(X_1^i)$. Modifications to 12 *msb* of the input message X_1^i and the resulting hash values H_1^i are shown in Table XX.
4) Extract the 16 *msb* of the output (as indicated in Table XX in red) and determine collisions within the set of 16-bit outputs H_1^i. After obtaining the total number of collisions, count the occurrences within predefined classes based on the collision count ranges, as shown in Table XIX.
5) Repeat steps 2 through 5 for the remaining $2^{20}-1$ input message samples, X_a.
6) The observed frequency o_i represents the result of the frequency distribution of collisions obtained across all 2^{20} samples.
7) Calculate the expected frequency e_i using Equation (7) with the number of experiments conducted, $v = 2^{20}$, and p_i as the probability of the expected frequency in class i according to Table XIX. The observed frequency o_i and the expected frequency e_i resulting from the collision test in the first trial of the Hortex algorithm are presented in Table XXI.

TABLE XX
THE RESULTS OF X_1^i, AND H_i^i FROM THE FIRST COLLISION TEST ON HORTEX.

i	X_1^i	H_1^i
1	000f1329	3d73d1a18a24ab74 e3ddac073ef093af
2	001f1329	327ae9d7cee668fd 53a96c01ec816d0a
⋮	⋮	⋮
2^{12}	ffff1329	0d0e730ac59ea7ec 872a93f12dc8dab8

TABLE XXI
THE RESULTS OF o_i AND e_i FROM THE FIRST COLLISION TEST ON HORTEX.

Class	Rank	o_i	e_i
1	0 - 116	216095	216264.6057
2	117 - 122	203818	203428.9869
3	123 - 128	230944	230512.6564
4	129 - 134	192370	192904.4296
5	134 - 4096	205349	205465.3215

8) Obtain the χ^2 value using Equation (8) based on the values of o_i and e_i in Table XXI and the p_{value} using Equation (9) with $df = 4$. The obtained χ^2 value is 3.2305, and the p_{value} is 0.52, as shown in Table VI.
9) Determine whether the null hypothesis $\mathbf{H}_0$ is accepted or rejected based on p_{value} and α. It is known from the previously obtained p_{value} that $p_{value} \geq \alpha$, indicating the acceptance of the null hypothesis $\mathbf{H}_0$. Consequently, due to the acceptance of the null hypothesis, Hortex passes the collision test (for the first trial).

Perform the aforementioned steps four more times, resulting in a total of five collision tests. This is done to ensure that the results of the collision tests are unbiased and not merely coincidental. The results of the five collision tests conducted on Hortex all passed, as indicated in Table VI.

E. Steps of Coverage Test on Hortex

The probabilities of the expected frequency to assess randomness, using the coverage test for test subjects, are illustrated in Table XXII calculated by [26]. In the coverage test, modifications will be applied to 12 *msb* of the message, and the corresponding variations in coverage will be observed in the 12 *msb* of the output hash message. The length of the message to be used is 32 bits. The significance level used in the coverage test is $\alpha = 0.01$.

The steps for conducting the coverage testing on Hortex, specifically for the first coverage test as shown in Table VII, are as follows: 1.

1) Generate 2^{20} samples of input messages, denoted as X_a, where $1 \leq a \leq 2^{20}$. Let the selected input message sample be $X_1 = \texttt{000f1329}$.
2) Modify 12 *msb* of the input message X_1 to obtain the modified message X_1^i, where $1 \leq i \leq 2^{12}$.
3) Compute the output message H_1^i for each modified message X_1^i, or $H_1^i = H(X_1^i)$, where $H(\cdot)$ represents the hash function. Modifications to the 12 *msb* of the message X_1^i and the resulting hash values H_1^i are presented in Table XX.

TABLE XXII
RANGES AND PROBABILITIES OF COVERAGE TESTS FOR A 12 *msb* OUTPUT OBSERVATION AND 12 *msb* INPUT MODIFICATION.

Class	Range	Probability
1	1 - 2572	0.199176
2	2573 - 2584	0.204681
3	2585 - 2594	0.197862
4	2595 - 2606	0.203232
5	2607 - 4096	0.195049

TABLE XXIII
THE RESULTS OF o_i AND e_i FROM THE FIRST COVERAGE TEST ON HORTEX.

Class	Rank	o_i	e_i
1	1 - 2572	209310	208851.1734
2	2573 - 2584	214289	214623.5843
3	2585 - 2594	207717	207473.3445
4	2595 - 2606	212579	213104.1976
5	2607 - 4096	204681	204523.7002

4) Extract the 12 *msb* from each generated value of H_1^i (the first 3 hex digits highlighted in red in the third column of Table XX) and calculate the total number of coverage variations observed.
5) Record the tally in classes based on the range of coverage variations, utilizing the classes and ranges specified in Table XXII.
6) Repeat steps 2 through 5 for the remaining $2^{20} - 1$ input message samples, X_a.
7) The observed frequency o_i represents the result of the frequency distribution of the coverage variations obtained in all 2^{20} samples.
8) Calculate the expected frequency e_i using Equation (7) with the number of experiments conducted, $v = 2^{20}$, and p_i as the probability of the expected frequency in class i according to Table XXII. The observed frequency o_i and the expected frequency e_i resulting from the coverage test in the first trial of the Hortex algorithm are presented in Table XXIII.
9) Obtain the χ^2 value using Equation (8) based on the values of o_i and e_i in Table XXIII and the p_{value} using Equation (9) with df=4. The obtained χ^2 value is 3.231, and the p_{value} is 0.519, as shown in Table VII.
10) Determine whether the null hypothesis $\mathbf{H}_0$ is accepted or rejected based on p_{value} and α. It is known from the previously obtained p_{value} that $p_{value} \geq \alpha$, indicating acceptance of the null hypothesis $\mathbf{H}_0$. Consequently, due to the acceptance of the null hypothesis, Hortex passes the coverage test (for the first trial).

Perform the aforementioned steps four more times, resulting in a total of five coverage tests. This is carried out to ensure that the results of the coverage tests are unbiased and not merely coincidental. The results of the five coverage tests conducted on Hortex all pass, as indicated in Table VII.

REFERENCES

[1] M. Kiss, G. Breda, and L. Muha, "Information security aspects of industry 4.0," *Procedia Manufacturing*, vol. 32, pp. 848–855, 2019.
[2] M. E. Whitman and H. J. Mattord, *Principles of Information Security*. Cengage learning, 2021.

[3] R. Sobti and G. Geetha, "Cryptographic hash functions: A review," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 2, p. 461, 2012.

[4] Y. Yang, F. Chen, X. Zhang, J. Yu, and P. Zhang, "Research on the hash function structures and its application," *Wireless Personal Communications*, vol. 94, pp. 2969–2985, 2017.

[5] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD," *Cryptology EPrint Archive*, 2004.

[6] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*. Springer, 2005, pp. 17–36.

[7] S. K. Sanadhya and P. Sarkar, "New collision attacks against up to 24-step SHA-2," in *Progress in Cryptology-INDOCRYPT 2008: 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings 9*. Springer, 2008, pp. 91–103.

[8] A. Zellagui, N. Hadj-Said, and A. Ali-Pacha, "Comparative study between Merkle-Damgård and other alternative hashes construction," in *Proceedings of the Second Conference on Informatics and Applied Mathematics IAM, Guelma, Algeria*, 2019, pp. 24–25.

[9] M. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," 2015-08-04 2015.

[10] I. Dinur, O. Dunkelman, and A. Shamir, "Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials," in *International Workshop on Fast Software Encryption*. Springer, 2013, pp. 219–240.

[11] J. Guo, G. Liao, G. Liu, M. Liu, K. Qiao, and L. Song, "Practical collision attacks against round-reduced SHA-3," *Journal of Cryptology*, vol. 33, pp. 228–270, 2020.

[12] S. Huang, X. Wang, G. Xu, M. Wang, and J. Zhao, "New distinguisher on reduced-round Keccak sponge function," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 102, no. 1, pp. 242–250, 2019.

[13] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 317–337.

[14] M. A. AlAhmad, "Design of a new cryptographic hash function–Titanium," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 10, no. 2, pp. 827–832, 2018.

[15] B. Widhiara, Y. Kurniawan, and B. H. Susanti, "RM70: A lightweight hash function." *IAENG International Journal of Applied Mathematics*, vol. 53, no. 1, pp. 94–102, 2023.

[16] M. A. Alia, "Cryptosystems based on chaos theory," *Chaos, Complexity and Leadership 2013*, pp. 129–145, 2015.

[17] M. Alawida, A. Samsudin, N. Alajarmeh, J. S. Teh, M. Ahmad *et al.*, "A novel hash function based on a chaotic sponge and DNA sequence," *IEEE Access*, vol. 9, pp. 17 882–17 897, 2021.

[18] N. Abdoun, S. El Assad, K. Hammoud, R. Assaf, M. Khalil, and O. Deforges, "New keyed chaotic neural network hash function based on sponge construction," in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2017, pp. 35–38.

[19] W. Huang and L. Wang, "A hash function based on sponge structure with chaotic map for spinal codes," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2019, pp. 1–5.

[20] H. Zhu, W. Qi, J. Ge, and Y. Liu, "Analyzing Devaney chaos of a sine–cosine compound function system," *International Journal of Bifurcation and Chaos*, vol. 28, no. 14, p. 1850176, 2018.

[21] N. K. Pareek, V. Patidar, and K. K. Sud, "Image encryption using chaotic logistic map," *Image and vision computing*, vol. 24, no. 9, pp. 926–934, 2006.

[22] M. Ahmad, S. Khurana, S. Singh, and H. D. AlSharari, "A simple secure hash function scheme using multiple chaotic maps," *3D Research*, vol. 8, pp. 1–15, 2017.

[23] B. Bouteghrine, C. Tanougast, and S. Sadoudi, "Novel image encryption algorithm based on new 3-d chaos map," *Multimedia Tools and Applications*, vol. 80, pp. 25 583–25 605, 2021.

[24] Z. Hua, Y. Zhang, H. Bao, H. Huang, and Y. Zhou, "N-dimensional polynomial chaotic system with applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 2, pp. 784–797, 2021.

[25] M. Alawida, J. S. Teh, A. Mehmood, A. Shoufan *et al.*, "A chaos-based block cipher based on an enhanced logistic map and simultaneous confusion-diffusion operations," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 10, pp. 8136–8151, 2022.

[26] A. Doğanaksoy, B. Ege, O. Koçak, and F. Sulak, "Cryptographic randomness testing of block ciphers and hash functions," *Cryptology ePrint Archive*, 2010.

[27] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC press, 2018.

[28] Z. Lin, C. Guyeux, S. Yu, Q. Wang, and S. Cai, "On the use of chaotic iterations to design keyed hash function," *Cluster Computing*, vol. 22, pp. 905–919, 2019.

[29] J. S. Teh, K. Tan, and M. Alawida, "A chaos-based keyed hash function based on fixed point representation," *Cluster Computing*, vol. 22, pp. 649–660, 2019.

[30] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in *ECRYPT Hash Workshop*, vol. 2007, no. 9, 2007.

[31] D. R. Stinson and M. Paterson, *Cryptography: Theory and Practice, Fourth Edition*. CRC Press, 2019.

[32] J. Amigo, L. Kocarev, and J. Szczepanski, "Theory and practice of chaotic cryptography," *Physics Letters A*, vol. 366, no. 3, pp. 211–216, 2007.

[33] H. E. Nusse, J. A. Yorke, E. J. Kostelich, H. E. Nusse, J. A. Yorke, and E. J. Kostelich, "Bifurcation diagrams," *Dynamics: Numerical Explorations: Accompanying Computer Program Dynamics*, pp. 229–268, 1994.

[34] R. M. May, "Simple mathematical models with very complicated dynamics," *Nature*, vol. 261, no. 5560, pp. 459–467, 1976.

[35] S. Weisberg, *Applied Linear Regression*. John Wiley & Sons, 2005.

[36] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.

[37] I. H. Masri and B. H. Susanti, "General chaos implementation as a construction element of primitive cryptography," in *2023 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 2023, pp. 168–174.

[38] V. A. Thakor, M. A. Razzaque, and M. R. Khandaker, "Lightweight cryptography for Iot: A state-of-the-art," *arXiv preprint arXiv:2006.13813*, 2020.

[39] 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.

[40] C. E. Shannon, "A mathematical theory of cryptography," *Mathematical Theory of Cryptography*, 1945.

[41] A. Maetouq and S. M. Daud, "Hmnt: Hash function based on new mersenne number transform," *IEEE Access*, vol. 8, pp. 80 395–80 407, 2020.

[42] B. H. Susanti, M. H. N. Ilahi, A. Amiruddin, and S. S. Carita, "Finding collisions in block cipher-based iterative hash function schemes using iterative differential," *IAENG International Journal of Computer Science*, vol. 48, no. 33, pp. 634–645, 2021.

[43] B. A. Forouzan, *Introduction to Cryptography and Network Security*. McGraw-Hill, 2008.

[44] G. Yuval, "How to swindle Rabin," *Cryptologia*, vol. 3, no. 3, pp. 187–191, 1979.

[45] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability And Statistics for Engineers and Scientists*. Macmillan New York, 1993, vol. 5.

Ikhwanul Hakim Masri is a Research Assistant at Badan Siber dan Sandi Negara. He earned a Bachelor's Degree in Cryptographic Engineering from Politeknik Siber dan Sandi Negara in 2023. His research focuses on cryptography and cryptanalysis.

Bety Hayat Susanti (M'2020) is an Associate Professor at Politeknik Siber dan Sandi Negara. She earned her Ph.D. in Mathematics from Institut Teknologi Bandung in 2019, following a Bachelor's degree in Mathematics (2000) and a Master's degree in Economics (2005), both from Universitas Indonesia. Her research interests include cryptography, cryptanalysis, discrete mathematics, and graph theory. She is an active member of the Indonesian Combinatorial Society (InaCombS), the Indonesian Mathematical Society (IndoMS), and the International Association of Engineers (IAENG).