Mathematics as a basis for the generation of cryptographic tools

Las matemáticas como base para la generación de herramientas criptográficas

A matemática como base para a geração de ferramentas criptográficas

Guillermo León Murcia[1]
Yeny Liliana Casas Méndez[2]
Segundo Leonardo Cortés López[3]
Cristian Eduardo Cano López[4]

Received: May 15th, 2023
Accepted: August 20th, 2023
Available: January 20th, 2024

How to cite this article:
G. León Murcia, Y.L. Casas Méndez, S. L. Cortés López, C.E. Cano López, "Mathematics as a basis for the generation of cryptographic tools" *Revista Ingeniería Solidaria,* vol. 20, no. 1, 2024.
doi: https://doi.org/10.16925/2357-6014.2024.01.03

Research article. https://doi.org/10.16925/2357-6014.2024.01.03

[1] Faculty of Engineering University of Cundinamarca Ubaté Campus
Email: gleonm@ucundinamarca.edu.co
ORCID: https://orcid.org/0000-0003-3756-1330
Cvlac: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001689491

[2] Faculty of Engineering University of Cundinamarca Ubaté Campus
Email: ylcasas@ucundinamarca.edu.co
ORCID: https://scienti.minciencias.gov.co/
Cvlac: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001048619

[3] Faculty of Engineering University of Cundinamarca Ubaté Campus
Email: sleonardocortes@ucundinamarca.edu.co
ORCID: https://orcid.org/0000-0002-1317-9623
Cvlac: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001841364

[4] Systems Engineering - University of Cundinamarca Ubaté Campus
Email: cecano@ucundinamarca.edu.co
ORCID: https://orcid.org/0000-0003-3112-691X
cvlac: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001401533

CC BY

Abstract

This article is the product of the research "Mathematics as a basis for the generation of cryptographic tools", developed at the Universidad de Cundinamarca, sectional Ubaté in the year 2021.

Problem: Improve the security of code with differential calculus, plane and spatial geometry, polar coordinates, and functions; the resulting encryption should improve on the standards of current encryption systems.

Objective: Explain the algorithm's operation with fundamental issues that use and generate a modification proposal to make the algorithm more robust when subjected to an attack by unethical hackers.

Methodology: A bibliographic review, encryption codes, history of algorithms, and operation of SHA-256 in the Ethereum platform and Bitcoin explains how it works and gives an idea of how to improve security.

Result: The algorithm's operation is explained step by step using subject topics such as precalculus and calculus. Also, topics such as functions, truth tables, logical operators, programming functions, and a way to complicate the algorithm were proposed by applying issues of plane geometry, space, polar coordinates, ceiling function, and absolute values.

Conclusion: It is possible to assimilate the operation of SHA-256 (256-bit secure hash algorithm) and the mathematical basis for generating codes in hexadecimal, depending on the message to be encrypted. Algebraic models allow taking different points of the figures generated in the Cartesian and polar planes to use them as a means of encryption.

Originality: Explaining an algorithm widely used in encryption issues; however, according to a survey conducted, only a few understood the code that explains the mathematics, all the aspects behind its operation, and why it is so secure. In addition, the project proposes a way to improve its security.

Limitations: The algorithm used was a hyperbolic and linear function combined. In this sense, common random points between the two functions are selected for encryption, and it is not aligned with the encryption systems used in blockchains; it is a restriction for the implementation. Currently, several updates are aligned with the SHA-256 system, involving limiting the inclusion of updated models.

Keywords: SHA-256, Blockchain, Ethereum, Cryptography, Security

Resumen

El artículo es producto de la investigación "Las matemáticas como base para la generación de herramientas criptográficas" desarrollada en la Universidad de Cundinamarca, seccional Ubaté en el año 20021.

Problema: se quiere mejorar la seguridad del código con cálculo diferencial, geometría plana y espacial, coordenadas polares y funciones en las reales.

Objetivo: explicar el funcionamiento del algoritmo a partir de cuestiones fundamentales que utiliza y generar una propuesta de modificación para hacer más robusto el algoritmo ante un ataque de hackers no éticos.

Metodología: realizar una revisión bibliográfica referente a códigos de encriptación, historial de algoritmos y funcionamiento del SHA-256 (algoritmo de hash seguro de 256 bits) en la plataforma Ethereum y Bitcoin que explica su funcionamiento y da una idea de cómo mejorar su seguridad.

Resultado: el funcionamiento del algoritmo explica paso a paso utilizando temas como precálculo, cálculo. Además, se propusieron temas como funciones, tablas de verdad, operadores lógicos, funciones de programación y una forma de complicar el algoritmo aplicando temas de geometría plana, espacio, coordenadas polares, función techo y valor absoluto.

Conclusión: es posible asimilar el funcionamiento del algoritmo SHA-256 y la base matemática para generar códigos en hexadecimal, dependiendo del mensaje a encriptar. Los modelos algebraicos permiten tomar diferentes puntos de las figuras generadas en los planos cartesiano y polar para utilizarlos como medio de encriptación.

Originalidad: parte de la idea de explicar un algoritmo muy utilizado en temas de encriptación; sin embargo, en una encuesta, solo algunos entendieron el código que explica las matemáticas, todas las cuestiones detrás de su funcionamiento y por qué es tan seguro. Además, una idea propone formas de optimizarla.

Limitaciones: dado que el algoritmo ya está funcionando y aún no se han reportado roturas en su funcionamiento. Una limitación puede ser que no se considere la sugerencia encontrada para modificar el código ya que funciona bien, entonces esta sugerencia puede ser rechazada.

Palabras clave: Sha-256, Blockchain, Ethereum, Criptografía, Seguridad

Resumo

O artigo é produto da pesquisa "A matemática como base para a geração de ferramentas criptográficas" desenvolvida na Universidade de Cundinamarca, seção Ubaté em 20021.

Problema: queremos melhorar a segurança do código com cálculo diferencial, geometria plana e espacial, coordenadas polares e funções reais.

Objetivo: explicar o funcionamento do algoritmo com base nas questões fundamentais que ele utiliza e gerar uma proposta de modificação para tornar o algoritmo mais robusto contra ataques de hackers antiéticos.

Metodologia: realizar uma revisão bibliográfica a respeito de códigos de criptografia, histórico de algoritmos e funcionamento do SHA-256 (algoritmo hash seguro de 256 bits) na plataforma Ethereum e Bitcoin que explique seu funcionamento e dê uma ideia de como melhorar seu segurança.

Resultado: O funcionamento do algoritmo é explicado passo a passo usando tópicos como pré-cálculo, cálculo. Além disso, foram propostos tópicos como funções, tabelas verdade, operadores lógicos, funções de programação e uma forma de complicar o algoritmo aplicando tópicos de geometria plana, espaço, coordenadas polares, função teto e valor absoluto.

Conclusão: é possível assimilar o funcionamento do algoritmo SHA-256 e a base matemática para geração de códigos hexadecimais, dependendo da mensagem a ser criptografada. Os modelos algébricos permitem-nos pegar diferentes pontos das figuras geradas nos planos cartesiano e polar para utilizá-los como meio de criptografia.

Originalidade: parte da ideia de explicar um algoritmo muito utilizado em questões de criptografia; No entanto, num inquérito, apenas alguns compreenderam o código que explica a matemática, todas as questões por trás do seu funcionamento e por que é tão seguro. Além disso, uma ideia propõe formas de otimizá-la.

Limitações: pois o algoritmo já está funcionando e ainda não foram relatadas interrupções em seu funcionamento. Uma limitação pode ser que a sugestão encontrada para modificar o código não seja considerada por funcionar bem, então esta sugestão pode ser rejeitada.

Palavras-chave: Sha-256, Blockchain, Ethereum, Criptografia, Segurança

1. INTRODUCTION

The work presented below is framed in the line of research of learning, knowledge, technologies, communication, and digitalization, founded by the University Research Directorate. The seedbeds of the Ebaté group, founded in 2009, prioritized each of the respective faculties of these academic units. The lines related to the field of knowledge and accumulated production are generated specifically from the research groups. The topic worked on in this article is an update of the SHA-1 algorithm, which was rendered insecure by 2016, leading to the need for the improved security provided by SHA-256. The aim of this research was to investigate the quantity and quality of mathematical structures that compose it and give a point of view on how it improves its level of security using more mathematical tools, then send an email to the NSA for study and possible approval [1].

Pseudocode-based encryption has made it possible to encrypt messages at a low computational cost to exchange an enormous amount of data to offer a new generation of services. However, we believe that so far, the potential of the applications used for this purpose has not been significantly exploited, due to security issues arising from the sensitive nature of the data handled by such programming languages.

Blockchain technology can contribute substantially to solving this problem by providing integrity and privacy when using the hashes generated by SHA-256. However, blockchains entail a significant demand for computational capabilities; fortunately, today's processors are able to meet this demand relatively easily [2].

The desire is to find a way to improve security while complicating code collision by sacrificing hardware acceleration of the most computationally intensive part of the code development that generates the hash corresponding to the encryption of a message, be it image text or number text and/or combination thereof, typically consisting of the massive computation of the hash function with the SHA-256 algorithm.

In addition to its essential role in blockchain deployment, the modification of the SHA-256 code can benefit the encryption capabilities as it is desired to make it more robust by strengthening the security schemes in the Digital Signature Algorithm, SHA-256 being one of the most adopted cryptographic hash functions nowadays, due to its security level, will generate more confidence if some fixed variables are changed by random ones with specific conditions based on the actual variable calculation [3].

Although ideally a balance should be reached between the desired performance and the available resources, especially the energy consumed and the computational capacity, what is desired here is to sacrifice just that item of computational capacity and machine resource to improve the integrity of the code, a sacrifice that is strengthened by the security issue and possible collision attack.

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

As a contribution to the code and to improve the choice of SHA-256 algorithm architecture for blockchain applications that best suit all requirements in each context, this research proposes an enhancement that is configurable concerning various encryption parameters. What is proposed can be adapted to achieve different levels of machine resources to satisfy the security complexity of the algorithm in question[4].

1.1 REVIEW OF LITERATURE OR RESEARCH BACKGROUND

To explain the encryption code's operation, we established precalculus, calculus and geometry analytics as bibliographic references [5], [6], [7], [8], [9], [10], [11], [12], [13],[14],[15],[16]. With this compendium, it was possible to identify the topics of binary numbers, functions, truth tables, logical connectors, permutations, combinations, rotations, and changes in variables. Additionally, various articles and documents were used and listed as bibliographic references.

Vector calculus texts were used, such as Vector Calculus for Engineers by Jeffrey R. Chasnov Source: The Hong Kong University of Science and Technology, and articles such as vector spaces by M.A. García Sánchez and T., Ramírez Alzola, the Vector Calculus by Juan Guillermo Rivera Berrío, and Elena Esperanza Álvarez Sáiz for improvements in encryption. Vector calculus is based on developing and applying the four operations, including gradient, rotor, divergence, and laplacian. In Cartesian coordinates, hyperbola, the line was transformed to polar coordinates with the help of the texts mentioned above.

Among the journals used for this article is the Journal of Solidarity Engineering, volumes 5 and 17 [17],[18]. The information works as a reference for the conditions needed by the functions, which will be encrypted and explained so that students from the first to the seventh semester can understand how the code of SHA-256 works. It was also possible to establish that it was necessary to contribute to the research, which would consolidate all this information and serve as a basis to make the code and have a more significant degree of security; similarly, reducing the risks of any attack that could occur in terms of vulnerability by third parties overcoming the exploitation of these.

2. MATERIALS AND METHODS

Performing a comparison exercise, if one asks, "What is the interpretation of what is written in the text of Table I?", they will probably perceive meaningless or incoherent

data. However, this is part of the result of encryption through this algorithm, suggesting relevant information stored in these characters.

This research focuses on the mathematical explanation of the behavior of this encryption. Thus, describing the process through algebra and the theoretical basis this process entails. In addition, a contribution from the number theory, graph theory, and algebraic geometry to improve the result of the encryption process.

TABLE I. Encryption example

Hvjmmfsnp Mf´pm

Source: Own work

If we carefully analyze, we can observe that the letters of the two hidden words are encrypted simply under the following rule:

Either f(a,b,c,...,y,z) = a, b, c, d, e, f, g, h, i, j, k, l, m, n, ñ, o, p, q, r, s, t, u, w, x, y, z; A function with which it can generate words of any kind such as names, things, places, among others.

In this case, for the letters of Guillermo León's words, we assign a change in their writing with a condition. For each encrypted letter, it places the letter in the order of the alphabet, as shown in Table II.

TABLE II. Original vs. encrypted name

Original	G	u	i	l	l	e	r	m	o		L	e	ó	n
Encrypted	H	v	j	m	m	f	s	n	p		M	f	p´	o

Source: Own work

The previous encryption was creatively written with a low level of encryption. It seeks to explain through an example of more advanced encryption how Ethash works. It is the latest version of the Dagger-Hashimoto algorithm, which is proof-of-work (PoW) and is used to encrypt Ether, which is the Ethereum cryptocurrency. Based on this, we will talk about SHA-256 and its operation.

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

To explain the SHA-256 algorithm's operation, it is necessary to convert the message to be encrypted to the binary system, considering that this can be a numeric, alphanumeric, string, document, or image value.

Subsequently, the procedure explains the sum in modules from two to thirty-two and the conditions of how it should apply to encryption. Then, the conditions of operator XOR are mentioned, as well as the functions that are used and programmed, together with the use of words and constants already declared from the hexadecimal code. Additionally, the importance of the application in blockchain, bitcoin, and Ethereum technologies is indicated.

Within the functions used by the algorithm for encryption, there is one called Right Rotate, called RotR (x, n), being x a binary array of 32 bits; and ¨n¨ , a displacement number that will make the variety. This arrangement consists of taking the last binary digit and placing it at the beginning of the agreement, as its name says. It rotates as often as desired in the initial arrangement to modify its structure and thus generate a new array of 32 bits but with a different order [19].

Before continuing, we will revisit the topics we will use in mathematics and computer science, such as ASCII code, binary system, hexadecimal system, module function, and XOR operator.

The code ASCII (American Standard for Information Interchange) is the American standard code for the exchange of communication. It is a numerical code whose purpose is to represent the characters. Each character has a code associated that identifies it as unique from the other characters, which, when standardized, allows for a dynamic communication between our native language and the language of the computer. In 1963, this code was born. In 1967, this code grew to include lowercase and numbers, and in 1981 the code was updated and generated the 8 bits today known with 256 characters divided into three groups. First, the control characters used to control devices range from 00 to 31. Second, the printable characters range from 32 to 127, and third, extended characters that are graphic and unique characters from 128 to 255, as depicted in the following table.

TABLE III. The ASCII Code

Caracteres ASCII de control			Caracteres ASCII imprimibles						ASCII extendido (Página de código 437)							
00	NULL	(carácter nulo)	32	espacio	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(inicio encabezado)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(inicio texto)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(fin de texto)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(fin transmisión)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(consulta)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(reconocimiento)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(timbre)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(retroceso)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(tab horizontal)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(nueva línea)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(tab vertical)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(nueva página)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(retorno de carro)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(desplaza afuera)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(desplaza adentro)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(control disp. 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(control disp. 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(control disp. 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(control disp. 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(conf. negativa)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(inactividad sínc)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(fin bloque trans)	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(cancelar)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(fin del medio)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(sustitución)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(sep. archivos)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(sep. grupos)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(sep. registros)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(sep. unidades)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(suprimir)														

Reference: [20]

As an example of how it works, using the word in our native language to greet "HELLO" (HOLA in Spanish). As seen in the table, the capital letter H = 72, the letter O = 79, the letter L = 76, and A = 65, so now our HOLA became 72 79 76 65, which is passed to binary code. Transforming a number to binary is done as follows.

72/2 = 36 with residue 0
36/2 = 18 with residue 0
18/2 = 9 with residue 0
9/2 = 4 with residue 1
4/2 = 2 with residue 0
2/2 = 1 with residue 0

So, 72 written in binary code is 1001000. The same is done with the numbers 79, 76, and 65.

Finally, zeros are added to complete this string of 8-bit ones and zeros.

H = 72 = 01001000 O = 72 = 01001111
L = 76 = 01001100 A = 65 = 01000001

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

So, our HOLA for the computer is 01001000 01001111 01001100 01000001

A binary system or base two systems whose original characters are only 0 and 1 is because, in the computer language, there are only two possibilities either: the voltage step is needed, or it is not needed. If there is or one is required, one is assigned. If there is not or it is not needed, then it is given a 0, and the name of the bit is set. Thus, the computer receives a sequence of bits, given from our native language, homologated to the programming language.

The binary system uses a power rule, as shown in the following table.

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	256	128	64	32	16	8	4	2	1

With this assignment, we can discover what number written in binary represents in our native language.

Código binario	1	0	0	1
Asignación	2^3	2^2	2^1	2^0
Resultado	8	4	2	1
Multiplicación	1 x 8	0 x 4	0 x 2	1 x 1
Valor total	8	0	0	1

Figure 1. Relationship of binary numbers
Source: Own work

Then we just add 8 + 0 + 0 + 1 = 9 so that 1001 = 9

If you want to convert text to binary, you need to follow two procedures, go from ASCII code to binary.

Module function MOD( ) is a function that demonstrates if two numbers can be divided precisely or otherwise isolate the residue or remainder of a calculation in a division. Let's see why 18 MOD(7) = 4

$$\begin{array}{r|l} 18 & 7 \\ \hline -14 & 2 \\ \hline 4 & \end{array}$$

As can be seen, when dividing 18 by seven, the remainder is 4, that is to say, MOD() divides one number by another and delivers; as a result, its residue, in computing, works like this, MOD (number1, number2), the conditions are:

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

Number1

A first input parameter is a number with a value of type INTEGER, SMALLINT, BIGINT, or DECIMAL. If each parameter is of type DECIMAL, the other parameter must also be a DECIMAL type. If each parameter is an INTEGER value, the other parameter can be INTEGER, SMALLINT, or BIGINT. Both parameters can be SMALLINT or BIGINT, but one parameter cannot be SMALLINT if the other is BIGINT.

Number2

A second input parameter is a number with a value of type INTEGER, SMALLINT, BIGINT, or DECIMAL. The same data type rules apply to numbers two and number one [5].

The XOR operator is a binary operator that has one output whenever the inputs are not equal, which occurs when one of the two inputs is exclusively true. This is the definition that coincides with the mod sum (2). We see its purpose:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Hex notation is a numbering system based on sixteen, which uses the digits 0123456789ABCDEF. It has 16 symbols that can represent a single digit—considering the powers of 16, which are 161 = 16, 162 = 256, 163 = 4096, 164 = 65 536, 165 = 1 048 576, and so on. Passing the number 1 779 033 703 to hexadecimal requires dividing it into 16 as many times as necessary until the residue is less than 16. In this case, the residues are 7, 6, 6, 14, 9, 0, 10, 6, which in hexadecimal is 6A09E667.

The hexadecimal system facilitates reading large numbers or very long sequences of bits since they group them into four bits. In the former ASCII codes, the Hexa numerical system is used in the source and destination addresses of Internet protocols in IP addresses.

Functions used in the encryption process.

Below are listed some of the functions that serve as the basis for the encryption process and are part of the background process consulted for elaborating the project.

The Right Shift function is called ShR(x,n), where x is a 32-bit array and "n" is the number of rotations that are required so that the last digit of that array does not rotate but becomes zero.

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

The Sigma function named δ0(x) takes the above two parts and becomes δ0(x) = (RotR(x,n) ⊕ RotR(x,n)) ⊕(ShR(x,n)), where n has the same conditions as the previous functions and ⊕ is the operator "XOR" which works in binary code leaving one if you have 0 and 1 or 1 and 0 and leaves 0 if you have 1 and 1 or 0 and 0.

The Choose function Ch(x,y,z) accepts three arrays, each of 32 bits, named Ch(x,y,z) = (xΛy) ⊕ (¬xΛz). It uses the other two functions and decides between two of the three functions with the operator XOR, considering the negation of the first array to combine it with the operator y (Λ).

The Majority Maj(x,y,z) function, named Maj(x,y,z) = ((xΛy) ⊕ (xΛz)) ⊕ ((yΛz)), combines the three arrays 1 to 1 with the operator Λ and then the distributive property for the operator ⊕ [21].

These functions operate depending on the message the user assigns to the variables X, Y, or Z. In the end, they become a string of bits. When this arrangement is obtained, one is added and converted to 32 bits adding how many zeros are necessary depending on the conditions of the message you want to encrypt. It becomes binary at 64 bits, and this message extends to N pieces of 512 bits converted to sexagesimal [22].

You then have a message that must pass to binary code; the characters are counted and multiplied by 8 to obtain the number of bits; followed by this, a one is added to the binary message. To complete the array, you must place x number of zeros to the prior number of bits that were obtained to pass the message to binary using the following formula x = (448 – L –1) mod 512, where "L" is the length of the original message, [8],[9]. As a result, a note should show how module 512 is applied. After this, the message is completed by passing to binary and ending with zeros on the left of the array until obtaining a 64-bit display. Therefore, this should result in a 512-bit array or multiple of 512.

Encryption continues to divide the message into N parts; this is done by dividing the extended message into 512. As 1024 is a multiple of 512, then N will be an integer that establishes the parts into which the message should be divided for later encryption.

At this moment, there are 8 HASH; these HASH are tables that store records and objects and then perform the search in base 0 and 1 [10], which must be used as follows: hash number 1 called H0 = 1779033703 which is written in decimal and must be passed to hexadecimal, so it happens with eight codes called hash, which is based on the behavior of the first square roots of prime numbers.

How do these HASHes work? Well, the square roots of 2,3,5,7,11,13,17,19 numbers are expressed in decimal form, so √2=1.414213562373095048801688724209 69

8078 56967 which to pass to binary has 1.01101010000010011110011001100110011 11111100111 10011100110111. Now we take the first 32 bits after the point, that is the fractional part of the square root of two, 1.01101010100001111001110011111001111 0011110111011101110 which is 1779033703. This number is written in a hexadecimal form called Ho, so Ho = 6a09e667. Similarly, it is done with h1, h2, h3, h4, h5, h6, h7, and h8 [23].

We generate the N pieces of our code with the message in binary code. We will generate eight new registers from Ho to H7 [24], different from the hashes already known previously. These assignments assign letters from A to H, which initiates in 32-bit zeros for every 1 64 constants called w1 to w64. As they are 32-bit arrays, a division of 512 digits of the original message makes it into 32 parts; in this way, 16 displays of 64 are obtained that must be finished. To complete the arrays from W17 to W64, we will use the following formula and give an example of how it works in Word 17.

$$Wi = \delta(w(i-2)\,) + w(i-7) + \delta_0(w(i-15)\,) + w(i-16) \qquad (1)$$

This sum is made in modules 2 to 32, as shown below.

$$wi = \delta(w\_(i-2)\,) + w\_(i-7) + \delta 0(w\_(i-15)\,) + w\_(i-16)$$

$$w\_17 = \delta 1(w\_15\,) + w\_10 + \delta 0(w\_2\,) + w\_1$$

At this moment, a question arises: What happens to parts other than 1 with w1? Two temporary variables, T1 and T2, are created [25].

$$T_1 = \sum(1(e) + ch(e,f,g) + k_1 + w_1) \qquad (2)$$

$$T_2 = \sum(0(a) + Maj(a,b,c)) \qquad (3)$$

After performing the previous procedure, the following process continues:

An “h” is assigned “g”,
A “g” is assigned “f”,
An “f” is assigned “e”,

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

An “e” is assigned “d” + T1,
A “d” is assigned “c”,
A “c” is assigned “b”,
A “b” is assigned to “a”,
and an “a” is assigned T1+T2

The last piece is completed, as shown below.

Ho = a + Ho
H1 = b + H1
H2 = c + H2
H3 = d + H3
H4 = e + H4
H5 = f + H5
H6 = g + H6
H7 = h + H7

All the hashes are joined, a process with which a binary code is subsequently generated. When passed to hexadecimal, it corresponds to the encrypted message as desired from the beginning [26].

APPLICATION OF HYPOTHESES

To contribute to the research that was done, we want to complicate the SHA-256 algorithm. For this, we came up with an idea to apply basic concepts of differential calculus, plane geometry, and polar coordinates.

Now, new modeling is applied to these functions to identify their behavior and possible solutions within the encryption process. For example, we take:

$$\text{Right Rotate} = \text{RotR}(x,n)$$
$$\text{Shift Rotate} = \text{ShR}(x,n)$$
$$\delta 0(x) = \text{RotR}(x,n) \oplus \text{RotR}(x,n) \oplus \text{ShR}(x,n).$$

The n is what we want to complicate in the Code since they are the number of rotations of the 32-bit array. Starting with a hyperbola and a line under normal conditions, whose equations are $x^2/a^2 - y^2/b^2 = 1$ ¨y¨ y = x, respectively.

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

In this way, the generalized form of the hyperbola $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$ and the line

$y = mx + b$ is obtained.

Given these conditions, a specific method is used to generate a number "n" for the encryption of the program [27].

Starting with a circumference in the xy plane of the form $x^2 + y^2 = r^2$

a line is located on the circumference tangent to the rim but on the XYZ plane, as shown below.

Vector tangent to the circumference

Autonomously, a point was created that moves positively and negatively on that line. The circumference is rotated, which may or may not vary in radius as desired, so the following figure is obtained.

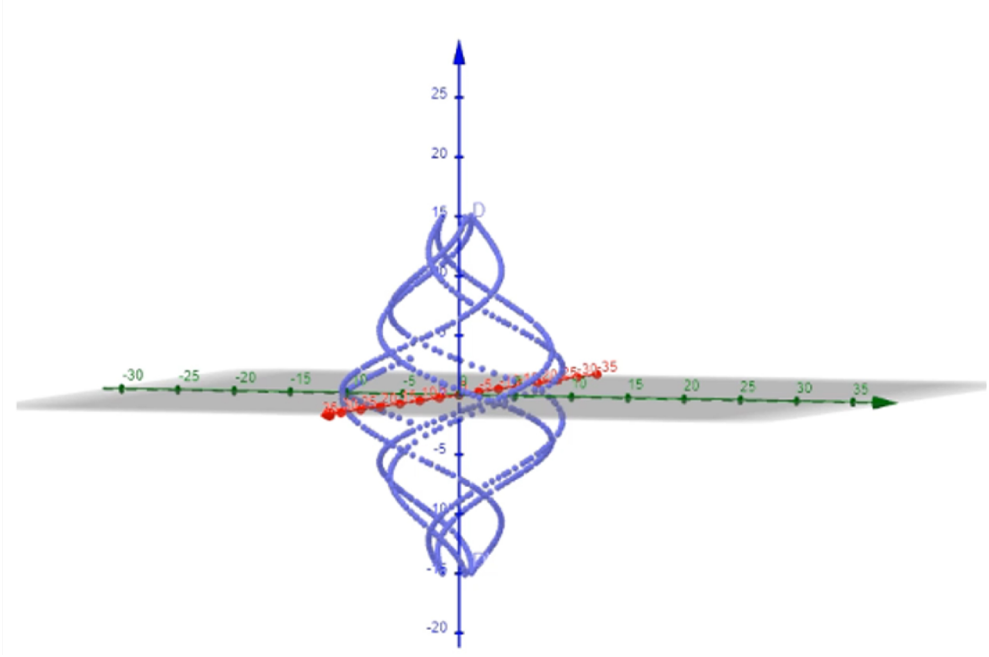

Figure 2. Random points between a hyperbole and a line.
Source: Own work

This way, we can use the polar coordinates to establish that n = r cos (θ). Since this generates a decimal and that, in the rotations of the function, does not make sense, we make "n" an integer by placing the roof function and using its absolute value

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

$$\boldsymbol{n = |[r(cos(\theta))]|.}$$

In this way, "n" is a positive integer as needed by the roles to which you intend to change those rotations [28],[29].

3. RESULTS

The algorithm uses several functions in its encryption process, but we will rely on 5 specifically, RotR(x,n), Right Shift ShR(x,n), Majority Maj(x,y,z), and Sigma δ(x), and Sigma subzero δ0(x). What do these functions have in common? They use a number "n" to generate different rotations in arrays of 32 bits. These rotations are specific to that value of "n", and rely on the properties of randomness, where randomness is intrinsic as a phenomenon of repetition in other spaces. It is here where I want to emphasize that the programming performed in the algorithm used these 5 functions that I mentioned above to which we want to remove the specific numbers and include the random factor and then verify that these rotations can generate a better encryption process in terms of security level, which increases security and the difficulty of access in case of attack [30].

It is important to consider that even after being validated, the algorithm is susceptible to changes that will surely improve its performance; whilst we must also consider the surprises in store with the advent of quantum computing and what it will bring for this type of encryption.

To improve its security the step to follow was to program the algorithm in Python where it is evident that the number "n" does not present a random behavior, since in doing so it generates a hash different from the original. These represent the number of rotations of each array uniquely, as evidenced in the following lines of programming.

```
...for """ in range (64):
    #Choice = (e and f) xor ((not e) and g).
    Choice = Ch(e, f, g).
    #Σ1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25).
    Σ1 = Xor(Xor(Xor(RotR(e, 6), RotR(e, 11)), RotR(e, 25)) # Original....
```

that is, with n=6, 11, and 25 the code works perfectly. Now, if we make that n=|⌈rcosθ⌉| under the conditions expressed in this article, then whoever wants to make the attack must know the way the algorithm selects the point that meets the necessary and sufficient conditions to guarantee the existence of that "n". (The complete

explanation regarding the choice of "n" can be requested by emailing gleonm@ucundinamarca.edu.co)

The evidence is constant and must remain unique to generate the hexadecimal code, but this cannot be done from here, it must be done in the main server and rename the algorithm; that is why a notification will be sent to the National Security Agency (NSA) and the National Institute of Standards and Technology (NIST), which were the creators of the code, to decide whether the suggestion applies or not. [31].

4. DISCUSSION AND CONCLUSIONS

When a message is encrypted with the conditions in which the algorithm was validated, a hash code is generated in approximately "Elapsed time: 0.0149998665 seconds." with a function called "time it". After we implemented the value of n=|⌈rcosθ⌋|, the Elapsed time rose to 0. 0327768666 seconds, which tells us that it requires more machine processing and must do more randomness calculations to find the same hash, this machine time is too much when someone wants to breach a transaction from a malicious aspect.

The characteristics of the mathematical components of SHA-256 seem to provide a better level of security than the hash functions that precede it such as SHA-1. Although the relative number of iterations is somewhat higher than SHA-1, it was possible to improve it significantly after implementing randomization in the 5 functions by improving the selection criteria and security arguments, creating a difficult path to reconstruct the encrypted message from the specification given to the constant "n".

We showed that it is still possible to improve the level of encryption security that can lead to a collision, although the response time is longer, and as expected also increases weight and the presence of both Cartesian and polar functions so that each step makes it very difficult to find the numbers needed to generate the hash corresponding to the encrypted message.

We have presented some ideas on how to modify and improve the encryption code and we have increased the probability of a local collision in SHA-256 which can lead to a false sense of security when compiling the code because it is a power of increasing exponent.

Thanks to the research carried out on the operation of the algorithm, it was evident that when programming it in Python and following each of the instructions described there, a space uses the values defined as constants in functions. For example, RotR(x,n), Right Shift ShR(x,n), Majority Maj(x,y, z) and Sigma δ(x), and Sigma subzero δ0(x) show that the form of encryption becomes predictable. A way to optimize the

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

algorithm is by placing specific numbers that only the algorithm knows since it must satisfy a hyperbolic function in the octal plane with real numbers.

It is possible to correctly assimilate the operation of the SHA-256 algorithm and the mathematical basis to generate the different codes in hexadecimal, according to the message required to encrypt.

Mathematically, algebraic models generate different points that allow taking in the figures generated in the Cartesian and polar planes to use them as a means for encryption. Still, these models must work on the initial conditions and the corresponding constants. Here, we have found a way to perform encryption with random numbers by increasing the difficulty of breaking code, using commonalities between the hyperbolic and linear functions, contributing to possible uses in blockchain security.

The article presents a precise cryptographic analysis of each of the parts. A message is composed of text-type, numeric, alphanumeric, or image data located in the cryptosystem defined by a mathematics protocol, including algebra, geometry, differential calculus, and precalculus. Other crucial aspects will be discussed, such as the different data, types, formats, and their distribution in the structure of the block. As well as the logical-mathematical operations of the cryptographic function SHA-256, which is susceptible to modification to improve its level of security that is used to generate a component resulting hash, that is, a result that meets the characteristics defined in the encryption protocol.

Finally, this hash will not change because its uniqueness does not allow it. However, it still represents the encryption from the original message, where its level for decryption will be enhanced. Therefore, it will be able to feed the blocks in what is known as the chain of blocks, or as the creator, Satoshi Nakamoto himself originally called it, the timing chain today blockchain.

The SHA-256 (256-bit Secure Hash Algorithm) is one of the most prominent cryptographic hash functions for safeguarding data integrity in computer security applications such as digital signature and password authentication.

However, despite SHA-256's acknowledged robustness in resisting attacks, it cannot be considered invulnerable. Among the potential attacks that can undermine its integrity, the so-called "brute force attack" stands out. This type of attack involves systematically testing all possible combinations of inputs until the one that corresponds to the desired hash is found.

Although SHA-256 is admittedly highly resilient to attacks, it is worth noting that, in theory, a cyber attacker could resort to brute-force methods to breach the security of the hash. However, the execution of such an attack requires a significant deployment of computational resources and considerable time to achieve success.

In conjunction, it is evident that, despite the theoretical vulnerability of SHA-256 to brute force attacks its security remains robust, due to the inordinate number of combinations that an attacker must test, resulting in an extremely low probability of success.

There is another attack called the "dictionary attack", characterized by the testing of common words or character strings, instead of the totality of possible combinations. Also included is the "collision attack", in which two different inputs generate the same hash value.

To date, SHA-256 has exhibited resistance to these types of attacks. However, on a theoretical level, the eventuality of collisions cannot be ruled out, given the restriction on hash length. In parallel, the "pre-image attack" is contemplated, which involves finding an input that produces a specific hash. SHA-256 is characterized by its resistance to this type of attack, attributable to the intricate complexity of the pre-image computation.

Additionally, "extended length attacks" reveal a type of threat that exploits the lack of resistance of the original hash function to malicious data extensions. To mitigate this vulnerability, techniques such as HMAC (Hash-based Message Authentication Code) and analogous strategies are commonly implemented.

Broadly speaking, SHA-256 remains a secure and widely employed choice in most security applications. In even more challenging contexts, quantum computing could limit the options, while ensuring the optimization of computational resources and time.

Therefore, the need to review and modify the "n" rotations in 3 specific functions RotR(x,n), δ0(x), and ShR(x,n) is raised. This paper goes into the exploration of a strategy that sacrifices computational resources to increase the probabilities of discovering potential vulnerabilities in the code and making possible attacks with a high degree of computational difficulty. In this context, the creation of an intersection between a hyperbolic function and a non-canonical linear function is proposed, generating points in common.

These points are then converted into polar coordinates using the formula of the cosine of the angle of inclination multiplied by the normal distance of the point. As a result, a decimal number is obtained, to which the roof function is applied. Since the options are translated into negative and positive values, the absolute value function is run as an integral part of the analytical process. In this way, the number of rotations assigned to the new rotation "n" is significantly increased, even at the cost of computational capacity.

e-ISSN 2357-6014 / Vol. 20, no. 1 / january-april 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

5. REFERENCES

[1] N-ABLE, SHA-256, *Algorithm Overview: ¿What is SHA-256?* 2019, p.1. [Online]. Available: https://www.n-able.com/blog/sha-256-encryption

[2] S. Sánchez. P. Domínguez y. L. Velázquez, *Hashing: Técnicas y Hash para la Protección de Datos*. 2011, pp. 4-5. [Online]. Available: https://www.laccei.org/LACCEI2018-Lima/student_Papers/SP96.pdf

[3] AWS, *Funciones matemáticas. Funcion MOD.* 2022, p.1. [Online]. Available: https://docs.aws.amazon.com/es_es/redshift/latest/dg/r_MOD.html.

[4] S.L. Bitcoinforme, *¿Qué es el algoritmo de minería Ethash?* 2015, Agosto 4, p.1. [Online]. Available: https://academy.bit2me.com/que-es-algoritmo-de-mineria-ethash/#0e04f783a4a3503a8

[5] S. James, *Cálculo. Transcendentes tempranas*. 7ª edición. Grupo Editorial Thomson Learning., 2013, pp. 10-58

[6] L. Ron, B. Edwards, *Cálculo*, tomo I. décima edición, Grupo Editorial Cengage Learning., 2014, pp. 383-3891.

[7] D. Zill, J. Dewar, *Ecuaciones diferenciales con problemas con valores en la frontera*, Octava Edición, España, (2013). pp. 182-189.

[8] J. Stewart, *Cálculo de una variable. Transcendentes tempranas.* 7ª edición. Thomson. 2012, pp. 10-71

[9] L. Leithold, *Cálculo con Geometría Analítica*. 7 edición Editorial Harla. México. 1978 pp. 2-10, 68-75.

[10] G. Thomas, *Calculus Thomas.* Special Edition. México, 2018, pp. 535-545.

[11] R. Larson, R. Hostetler, B. Edwards, *Cálculo de una variable,* Volumen 1. novena Edición. Mc. Graw Hill. Méjico. 2018, pp. 390-399.

[12] C. Edwards, H. Jr.; Penney, E. David, *Cálculo con Geometría Analítica*. Novena edición. Prentice Hall. 2018, pp. 59-69.

[13] T. Apóstol, *Análisis matemático*. Editorial Reverte: Barcelona, 1991. pp. 581-589.

[14] P. Ruiz. *Cálculo vectorial*. Prentice-Hall Hispanoamericana. 2018, Pp. 44-50.

[15] R. Jiménez, *Matemáticas VI. Cálculo Integral*. México: Pearson Educación, 2013, Pp. 129-133

[16] W. Granville, *Cálculo Diferencial e Integral.* México: Editorial Limusa. 2018. Pp. 179-189.

[17] J. A. Ortega, "Performance of routing and spectrum allocation (RSA) algorithms for a last generation centralized optical network (SDON)", *Revista Ingeniería Solidaria*, vol. 17, no. 2, pp. 1–30, May 2021. https://doi.org/10.16925/2357-6014.2021.02.08

[18] J. G. Ferrer Rodríguez, "System of Inductive Reasoning Based on Genetic Algorithms For The Solution Of Problems In Conditions Of Imprecise Information Part I", *Revista Ingeniería Solidaria*, vol. 5, no. 9, pp. 21–26, Jan. 2010. https://doi.org/10.16925/issn.1900-3102

[19] G. Diáz, *Ethereum: historia de la plataforma de contratos inteligentes más usada*. 2018, Julio 30, p.1. [Online]. Available:https://www.criptonoticias.com/tecnologia/ethereum-historia-plataforma-contratos-inteligentes-usada/

[20] Comité Estadounidense de Estándares. *El codigo ASCII*. [Online]. Available: https://elcodigoascii.com.ar/

[21] J.Minimalsm, *Máquina Virtual De Ethereum (Evm).* [Online]. Available: https://ethereum.org/es/developers/docs/evm/, 2022, 12 agosto, p.1

[22] S.L. *Bitcoinforme ¿Qué es un DAG?* 2015, agosto 4, p.1 [Online]. Available:https://academy.bit2me.com/que-es-un-dag/

[23] D. Rachmawati, J. Tarigan, y M. Ginting, ABC, "Un estudio comparativo de Message Digest 5 (MD5) y el algoritmo SHA256," *Journal of Physics: Serie de conferencias*, vol. 978, no. 1, pp. 012116. DOI:10.1088/1742-6596/978/1/012116

[24] X. Fan, B. Niu, Implementación basada en arquitectura multinúcleo y SIMD en SHA-256 de Blockchain. In: Xu, K., Zhu, J., Song, X., Lu, Z. (eds) Blockchain Technology and Application. CBCC 2020. *Comunicaciones en Informática y Ciencias de la Información*, vol 1305. Springer, Singapur. https://doi.org/10.1007/978-981-33-6478-3_4

[25] H. Yoshida, A. Biryukov, "Análisis de una variante SHA-256. En: Preneel, B., Tavares, S. (eds) Áreas seleccionadas en criptografía," *Lecture Notes in Computer Science*, vol 3897. Springer, Berlín, Heidelberg, 2016. https://doi.org/10.1007/11693383_17

[26] W. Andrew, N. Appel, "Verification of a cryptographic primitive: SHA-256," *ACM Trans. Program. Lang. Syst.* vol. 37, no. 2, pp. 31. doi: https://dx.doi.org/10.1145/2701415

[27] R. Martino, A. Cilardo, "Designing a SHA-256 processor for blockchain-based IoT applications, Internet of Things", vol. 11, 2020, 100254. https://doi.org/10.1016/j.iot.2020.100254. P.1

[28] M. Padhi, R. Chaudhari, An optimized pipelined architecture of SHA-256 hash function, *2017 7th International Symposium on Embedded Computing and System Design (ISED),* Durgapur, India, 2017, pp. 1-4. doi: 10.1109/ISED.2017.8303943.

[29] I. Ahmad, A. S. Das, "Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs," *Computers & Electrical Engineering*, vol. 31, no. 6, pp. 345-360. doi https://doi.org/10.1016/j.commpeleceng.2005.07.001.

[30] G. Bertoni, J. Daemen, M. Peeters, V. Aasche, *The Keccak SHA-3*. submission – version 3. 2017, p.1. [Online]. Available: https://keccak.team/files/Keccak-submmission-3.pdf

[31] M. Kammoun, M. Elleuchi, M. Abid y M.S. BenSaleh, "Implementación basada en FPGA del algoritmo hash SHA-256", *Conferencia internacional IEEE 2020 sobre diseño y prueba de micro y nanosistemas integrados (DTS),* 2020, pp. 1-6. doi: https://doi.org/10.1109/DTS48731.2020.9196134