Comparative Analysis of Cryptographic Hash Functions: Evaluating SHA-256 and BLAKE-256 within Cryptocurrency Applications

DAGMAWI ZERIHUN, Denison University, USA

This paper explores the crucial role of cryptographic hash functions in cryptocurrencies such as Bitcoin. These functions, crucial for securing transactions without a centralized authority, illustrate the advanced capabilities of digital signatures and hash algorithms. Cryptocurrencies are digital currencies that enable transactions without the need for a central authority through computer network-based transaction verification. Cryptocurrencies leverage cryptographic hashing algorithms to ensure the security and decentralization of their networks. Through a detailed examination and comparison of SHA-256 and BLAKE-256, two notable cryptographic hash functions in the cryptocurrency space, this study highlights their vital features and comparatively analyzes them with regard to speed, security, application space, and complexity within digital currency systems.

Additional Key Words and Phrases: Cryptocurrency, Cryptography, Hashing, Secure Hashing, BLAKE, SHA-256, Cryptographic Hash Functions

ACM Reference Format:

Dagmawi Zerihun. 2024. Comparative Analysis of Cryptographic Hash Functions: Evaluating SHA-256 and BLAKE-256 within Cryptocurrency Applications. 1, 1 (May 2024), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the digital financial landscape, secure transaction processing poses a significant challenge. Cryptographic hash functions play a crucial role in cryptocurrencies by providing a mechanism for transaction security without centralized oversight. The emergence of cryptocurrencies like Bitcoin and Ethereum introduces complexities to digital transactions that operate independently of central authorities.

In this paper, I survey select cryptographic hash functions, their properties, and their use case in the networks of cryptocurrencies. These functions are integral to the security framework of cryptocurrencies. After a brief overview of cryptographic principles, this paper gives a detailed overview of two important cryptographic hash functions used in the security of popular cryptocurrencies: SHA-256 and BLAKE. This will be followed by a comparative analysis of these algorithms, evaluating their critical features and performance regarding security and efficiency.

2 BACKGROUND

Cryptocurrency is a digital asset that utilizes blockchain technology or a distributed ledger to ensure a secure transaction [11]. It utilizes cryptography to secure transactions and effectively "eliminate" the intermediaries in financial transactions such as central banks or other comparable traditional financial institutions while ensuring security and performing the functions of a traditional currency (store of value, medium of exchange)[11]. Unlike traditional financial transactions, which are controlled and verified by a central entity, cryptocurrencies are decentralized, using a network

Author's address: Dagmawi Zerihun, Denison University, Granville, USA.

Manuscript submitted to ACM

of nodes and a distributed ledger to verify transactions, making them secure and reliable [11]. The concept of the first cryptocurrency – Bitcoin – was first introduced in 2008 under the pseudo-name "Satoshi Nakamoto". It was designed based on the principles of cryptography – decentralization, cryptographic hashing, asymmetric key cryptography, and digital signatures. These principles are foundational to Bitcoin's underlying blockchain technology - a decentralized way of recording transactions across a network of computers [23].

Cryptography is the technique used to convert plaintext(readable data) into ciphertext(encoded, decryptable data) [7]. Plaintext, in this context of secure communications, means all the data—text, images, video, and everything else—in its understandable form and accessible without any decryption procedures. On the other hand, the ciphertext results from the encryption process in which the plain text data, using a cipher and an encryption key, is converted into scrambled form to not allow it to be seen by an unauthorized entity. This will assure the confidentiality of the transmitted data so that only an authorized party possessing the correct decryption key can revert the ciphertext back to its original plaintext form. Cryptography can be broadly classified into two categories: symmetric key cryptography and asymmetric key cryptography, and each has very well-defined methodologies and applications [7].

Encryption technologies are critical for transforming readable data into formats that unauthorized users cannot decipher, preserving the confidentiality of information [18]. Despite this, they do not ensure that data remains unaltered over time. Data integrity is the assurance that information is accurate and has not been subject to unauthorized changes, whether accidental or deliberate[18]. It is a crucial component in the broader landscape of data security, ensuring the reliability and trustworthiness of information.

The concept of hashing is at the forefront when considering mechanisms for maintaining data integrity in cryptocurrencies. Central to the functioning of cryptocurrencies is the concept of hashing. A hashing function is a deterministic procedure that takes an arbitrary size input, or 'message,' and returns a fixed-size string of bytes, typically meant to represent the 'message' known as a hash. It serves as a digital signature for the data, with any alteration to the data resulting in a different hash output.
[14]. More formally,

Definition 2.1. A hash function is a function $h : D \rightarrow R$, where the domain $D = \{0,1\}^*$ and $R = \{0,1\}^n$ for some $n \geq 1$. (According to Merkle, as cited in [20])

Hash functions could have several essential properties that make them suitable for cryptographic purposes: they are fast to compute, difficult to reverse (pre-image resistance), infeasible to find two different inputs that produce the same output (collision resistance), and a small change in the input radically changes the output (avalanche effect)[20].

3 CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash functions build upon this concept by incorporating security features that make them suitable for protecting sensitive information. They create hash values that are theoretically impossible to reverse-engineer, ensuring that the original data cannot be retrieved from the hash alone [6]. Additionally, these functions are sensitive to input changes. Altering even a single bit of the original data leads to a significantly different hash [6]. This sensitivity is

integral to the security protocols of cryptocurrencies, as it allows for the verification of data integrity and the prevention of tampering, supporting the stability and decentralized nature of blockchain technology.

The National Institute of Standards and Technology (NIST) formally defines cryptographic hash functions as follows:

Definition 3.1. A cryptographic hash function is a function that maps a bit string of arbitrary length to a fixed-length bit string. Approved hash functions satisfy the following properties: 1. One-way – It is computationally infeasible to find any input that maps to any pre-specified output. 2. Collision resistant – It is computationally infeasible to find any two distinct inputs that map to the same output. 3. Second pre-image resistant – given one input value, it is computationally infeasible to find a second (distinct) input value that maps to the same output as the first value. [6].

Depending on whether a secret key is used in the hashing process, cryptographic hash functions can be broadly classified into two. Unkeyed hash functions are hash functions that only use a string as an input in the hashing process, whereas keyed hash functions use a secret key in addition to a string as a separate input. The term "hash functions" generally refers to unkeyed hash functions[21].

3.1 Properties of Cryptographic Hash Functions

Cryptographic hash functions must fulfill certain properties beyond the basic hash function properties for use cases such as ensuring transaction integrity and security in cryptocurrencies.

One-Way Property. A hash function is deemed one-way if it is computationally infeasible to reverse-engineer the original input from its hash value. Given a hash function ($h : D \rightarrow R$), where (D) represents the domain of all possible inputs and (R) denotes the range of fixed-length outputs (hash values), the one-way property asserts: [For any output $y \in R$, finding an input $x \in D$ such that $h(x) = y$ is computationally challenging.] In other words, the inverse operation—finding the pre-image (x) given the hash value (y)—should be extremely difficult. For example, if "XYZ" is hashed and the output is "123456abc", it should be computationally infeasible to deduce that the original input was "XYZ" from this hash output alone. This property ensures that hash functions act as irreversible transformations, crucial for data security [15].

Collision Resistance. Collision resistance in a hash function h is the property that it should be computationally infeasible to find any two distinct inputs $x, x' \in D$, where $x \neq x'$, that hash to the same output, i.e., $h(x) = h(x')$[21]. This is a vital security feature that protects against the possibility of two different messages being indistinguishable based on their hash values alone. Example: It would be a significant security issue if both "XYZ" and another input, say "ABC", resulted in the identical hash value of "7890defghi".

Second Pre-Image Resistance. Second Pre-image Resistance. A hash function h demonstrates second pre-image resistance if, for any input $x \in D$ and its hash output $y = h(x)$, it is computationally infeasible to find a different input $x' \in D$, where $x' \neq x$, that results in the same output y, meaning that $h(x') = y$ [11]. This property ensures the uniqueness of the hash output for every distinct input[15]. Example: If "XYZ" hashes to "abcdef123", finding any other string that also results in the hash abcdef123 should be highly unlikely.

3.2 Notable Cryptographic Hash Algorithms

This section covers some of the notable cryptographic hashing algorithms in the recent past with a vast array of applications in the cryptocurrency space.

SHA(Secure Hashing Algorithm) family(SHA1 & SHA2). The SHA family is a fundamental group of cryptographic hashing algorithms with extensive use in cryptocurrencies. Published by the National Institute of Standards and Technology (NIST), this family of algorithms encompasses seven different hash functions: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 [15]. The first three hash functions process data in 512-bit blocks segmented into 32-bit words, whereas the remaining four handle data in 1024-bit blocks segmented into 64-bit words [15].

Each function differs in the message digest size that it produces and the security strength it provides [15]. A message digest is a cryptographic hash function's unique output, similar to a digital 'fingerprint' for any form of data. For instance, a digest could turn the sentence "Less is more" into a jumble of characters like "5c6ffbdd40d9556b73". The security of these algorithms was called into question when weaknesses in SHA-1 were discovered, making it possible for attackers to find two different inputs that yield the same output, or to deduce the original input from its output more easily than expected [15]. These issues led to SHA-1 being considered compromised in its ability to serve as a reliable security mechanism [15].

Among the SHA family, the SHA-256 algorithm is particularly notable for its use in Bitcoin, the first and largest cryptocurrency in the digital currency market at this time [13]. SHA-256 is key to maintaining the integrity and security of Bitcoin's transaction recording system. It takes transaction data and converts it into a fixed-size string of characters. This serves as a seal that verifies the information's authenticity and connects it in a chronological and unalterable series, much like how entries in a ledger are recorded in order. This method is essential for creating trust in the system, as it makes altering recorded data computationally impractical. This helps prevent fraud and ensures the system's credibility ([13]).

BLAKE. BLAKE is a cryptographic hash function that stands out for its speed and security, developed as a candidate for the NIST SHA-3 competition [4]. Although it was not selected as the final standard, BLAKE's design was highly appreciated for its robustness and efficiency. The algorithm is known for its high security and speed in software implementations. BLAKE operates on 32-bit words for its BLAKE-256 variant and 64-bit words for BLAKE-512, producing hash outputs of corresponding sizes, which make it adaptable for diverse cryptographic purposes [4]. Despite not being as pervasive as SHA-256 in cryptocurrencies, BLAKE's influence is notable, especially in its optimized form, BLAKE2, which has been adopted by several cryptocurrencies due to its great performance and security features.

4 A SURVEY OF SELECT CRYPTOGRAPHIC HASH FUNCTIONS USED IN CRYPTOCURRENCIES

This section discusses the practical application of cryptographic algorithms in cryptocurrency, including securing transactions, creating wallets, and maintaining blockchain integrity. Different algorithms that are foundational to the security of cryptocurrencies are surveyed.

4.1 SHA-256

The SHA-256 algorithm is an iterative hash function designed to condense an input message into a compact summary known as a message digest [15]. It is best known for its use in the first cryptocurrency Bitcoin among others. The

algorithm can be described in two stages: preprocessing and hash computation. During processing, the input message is padded and parsed into m-bit blocks of 512 bits. The parsed message is represented as sixteen 32-bit words [15]. Initialization values are initialized for use in the hash computation. A padded message is taken, and a message schedule is generated using the padded message [15]. This computation iterates, using the message schedule with functions, constants, and word operations to form a series of hash values [15]. The final hash value generated by the hash computation is used to determine the message digest [15].

Before delving deeper into each component, the following are important operations, symbols, definitions, functions, and parameters:

Symbol	Description
$H^{(i)}$	The ith hash value, from initial $H^{(0)}$ to final $H^{(N)}$, used to determine the message digest.
K_t	Constant value used for iteration t of the hash computation.
M	Message to be hashed.
$M^{(i)}$	Message block i, size m bits, processed during hashing.
l	Length of the message M, in bits.
N	Number of blocks in the padded message, important for process structuring.

Table 1. Secure Hash Algorithm Parameters (adapted from [15])

Symbol	Operation Description
$\wedge$	Bitwise AND operation. Used in various logical functions within SHA-256 to combine bits from multiple words.
$\vee$	Bitwise OR operation. Utilized in SHA-256 for combining multiple hash computation results.
$\oplus$	Bitwise XOR operation. Fundamental in creating unique results in each round of SHA-256 by combining bits from the words and constants.
$+$	Addition modulo 2^{32} or 2^{64} (depending on the word size). Performs arithmetic addition within constraints of word size to prevent overflow.
$\mathbf{ROTR}^n(x)$	The rotate right (circular right shift) operation, where x is a w-bit word and n is an integer with $0 \leq n < w$. Essential for mixing bits across the word to ensure uniform diffusion in SHA-256.
$\mathbf{SHR}^n(x)$	The right shift operation, where x is a w-bit word and n is an integer with $0 \leq n \leq w$. Used in the message schedule of SHA-256 to generate new words from the padded message blocks.

Table 2. Operations in SHA-256 Hash Algorithm Specifications (Adapted from [15])

SHA-256 Functions

SHA-256 uses six logical functions each operating 32-bit words, denoted by x, y, and z. The result of each function is a new 32-bit word [15].

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{1}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{2}$$

$$\sum_{256}^{0}(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{3}$$

$$\sum_{256}^{1}(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{4}$$

$$\sigma_{256}^{0}(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x) \tag{5}$$

$$\sigma_{256}^{1}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \tag{6}$$

[15]

SHA-256 Constants

SHA-256 uses a sequence of sixty-four constant 32-bit words, $K_0^{256}, K_1^{256}, \ldots, K_{63}^{256}$. These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers [15].

Preprocessing

Preprocessing is designed to prepare the message for a secure and effective hashing process[15]. By standardizing the message length and structuring it into uniformly sized blocks, preprocessing ensures that messages of varying lengths are accommodated within the algorithm's fixed block size. This uniformity is crucial for the integrity and security of the hashing process. It allows the hashing algorithm to apply the same processing steps repeatedly on every block, ensuring that every bit of the input data influences the final hash consistently [15]. Uniform processing also ensures that the algorithm functions predictably and efficiently, safeguarding against potential vulnerabilities associated with handling data of varying sizes which could compromise the hashing process. Additionally, this standardized approach facilitates the subsequent phases of the SHA-256 algorithm, where complex mathematical operations are performed on these blocks to generate a unique hash value. In the SHA-256 algorithm, the pre-processing phase consists of three steps [15].

1.Padding the message. The padding of a message aligns the message's length to 512 bits. This padding process can be applied at the beginning of hash computation, during the message setup, or it can be implemented at any point before the hash computation enters the stage where it processes the message blocks that will include the padding. Consider the length of the message, M, to be l bits. To pad the message for SHA-256, one appends the bit "1" to the end of the message. Following this, k zero bits are added, where k is the smallest non-negative solution to the equation $l + 1 + k \equiv 448 \pmod{512}$ [15]. The padding concludes by appending a 64-bit representation of the length l of the original message in binary.

In the previous example, the ASCII-encoded message "XYZ", has a length of $3 \times 8 = 24$ bits. The padding begins with the addition of a "1" bit, followed by k zero bits. Here, k is chosen such that the total length — including the original 24 bits, the appended "1" bit, k zero bits, and the 64-bit length field — is a multiple of 512. The length field will encode the decimal number 24, or 00011000 in binary, which occupies the final 64 bits of the padded message.

$$\underbrace{01011000}_{\text{“}X\text{”}}\ \underbrace{01011001}_{\text{“}Y\text{”}}\ \underbrace{01011010}_{\text{“}Z\text{”}}\ \underbrace{1}_{1}\ \underbrace{0\ldots0}_{423\ \text{‘0’s}}\ \underbrace{0\ldots011000}_{\ell=24}$$

This process ensures that the length of the padded message is a multiple of 512 bits, making it suitable for processing by the SHA-256 hashing algorithm.

2.Parsing the message. Upon completing the padding, we proceed to parse the message for SHA-256 processing. This crucial step involves dividing the padded message into N blocks, each consisting of 512 bits [15]. For our example using the message "XYZ", the parsing phase would result in a single 512-bit block due to the message's original length plus padding. This block is then expressed as sixteen 32-bit words per the SHA-256 algorithm specifications.

Let's consider how the "XYZ" padded message is dissected into its constituent words:
The first 32 bits contain the ASCII representations for "X," "Y," and "Z," along with the beginning of the padding. In our case, since "XYZ" occupies only 24 bits, the subsequent 8 bits would incorporate the '1' bit of padding followed by the first '7' '0' bits of padding.

The 512-bit message block is then denoted by $M_0^{(1)}, M_1^{(1)}, \ldots, M_{15}^{(1)}$, where $M_0^{(1)}$ encapsulates the "XYZ" plus padding, and each subsequent $M_j^{(1)}$ (for $j = 1$ to 15) comprises a 32-bit word formed by contiguous bits of the padded message [15]. The final word, $M_{15}^{(1)}$, includes the binary representation of the length of the original message, which is 24 bits. Parsing transforms the initial message into a format compatible with the SHA-256 hashing process, setting the stage for the next phase, where these blocks undergo an intricate series of operations to produce the final hash value.

3.Setting the initial hash value. In the SHA-256 hashing algorithm, the computation begins with a set of predefined initial hash values, $H^{(0)}$, which consist of eight 32-bit words[15]. These initial values are derived from the square roots of the first eight prime numbers, ensuring a deterministic yet non-arbitrary start to the hashing process. For example, the initial hash value $H_0^{(0)}$ is calculated from $\sqrt{2}$, resulting in '6a09e667' in hexadecimal.

Index	Initial Hash Value	Index	Initial Hash Value
$H_0^{(0)}$	$6a09e667$	$H_4^{(0)}$	$510e527f$
$H_1^{(0)}$	$bb67ae85$	$H_5^{(0)}$	$9b05688c$
$H_2^{(0)}$	$3c6ef372$	$H_6^{(0)}$	$1f83d9ab$
$H_3^{(0)}$	$a54ff53a$	$H_7^{(0)}$	$5be0cd19$

Hash Computation

Once the input string undergoes the three parts of the preprocessing phase, the hash computation phase combines mathematical functions and logical operations to transform the input string into a unique and irreversible hash output.

Preparing the message schedule. In the SHA-256 algorithm, the computation of each message block begins by creating a message schedule—an essential component for the subsequent operations. The message schedule, denoted as W_t, consists of 64 words, each 32 bits in size [15]. The first 16 of these words are populated directly from the processed message blocks, ensuring that every part of the original message impacts the hash computation. The construction of the remaining 48 words employs a formula that significantly increases complexity and unpredictability through bitwise

operations and functions. Specifically, each word from W_{16} to W_{63} is generated using the following equation:

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$$

[15] where σ_0 and σ_1 are functions designed to rotate and shift the bits of their inputs, thereby mixing the data thoroughly. This process not only disperses the influence of the initial bits across the entire block but also intertwines the dependencies among the words, which is critical for the security and integrity of the resultant hash. By the end of this step, the message schedule is fully prepared, laying the groundwork for the complex transformations in the next phase of the algorithm.

Initializing the eight working variables. The second step in the SHA-256 hash computation is initializing the eight working variables: a, b, c, d, e, f, g, h. These variables serve as the backbone for the transformation functions that follow and are initialized with the hash values derived from the previous block's results. Specifically, each variable a through h is set to H_0 through H_7, respectively, which are the current hash values computed in the preceding block, or the initial hash values for the first message block [15]. This initialization links the hash computation of sequential blocks, ensuring that the hash output represents the cumulative effect of all previous message blocks. By carrying forward the results of one block to the next, SHA-256 ensures that each bit of the entire message influences the final hash outcome, making the algorithm resistant to attack and reliable for cryptographic purposes.

Core Transformation Process. The third step is the core of the SHA-256 hashing mechanism where the actual transformation of input data occurs. Over 64 rounds, the eight working variables (a, b, c, d, e, f, g, h) are updated using a mix of logical functions, bitwise operations, and constants derived from cube roots of the first 64 prime numbers [15]. This step uses two main temporary values, T_1 and T_2, which are calculated in each round and used to update the working variables. Here is how it works:

- T_1 **Calculation:** This is computed as a sum of the current value of h, a function $\Sigma_1(e)$ that rotates and shifts the bits of e, a choice function $Ch(e, f, g)$ that outputs one of its inputs based on the condition provided by another input, a constant K_t, and the word of the message schedule W_t. This value is pivotal as it introduces complexity and randomness in the mixing of bits, crucial for cryptographic security [15].
- T_2 **Calculation:** This involves the sum of two rotated and shifted versions of a and a majority function $Maj(a, b, c)$ that picks the majority bit from three inputs, ensuring that no single input can dominate the outcome [15].

Subsequently, the eight working variables are updated in the following manner:

- $h = g$
- $g = f$
- $f = e$
- $e = d + T_1$
- $d = c$
- $c = b$
- $b = a$
- $a = T_1 + T_2$

[15]

This series of updates enhances the diffusion of input characteristics across the hash computation, ensuring that each bit of the input data significantly influences the output.

Compute the Intermediate Hash Values. In the final step of each round in the SHA-256 hash computation, the intermediate hash values ($H_0, H_1, \ldots, H_7$) are updated by incorporating the transformed working variables (a, b, c, d, e, f, g, h): [15]

$$\begin{aligned} H_0' &= H_0 + a, \quad & H_4' &= H_4 + e, \\ H_1' &= H_1 + b, \quad & H_5' &= H_5 + f, \\ H_2' &= H_2 + c, \quad & H_6' &= H_6 + g, \\ H_3' &= H_3 + d, \quad & H_7' &= H_7 + h. \end{aligned}$$

These updates ensure that the results of the current block processing are effectively carried over to the next steps or contribute to the final hash output when all blocks have been processed.

Final Step. After repeating steps one through four for a total of N times, i.e., after processing all N message blocks, the final hash value of the SHA-256 algorithm is obtained by concatenating the last updated intermediate hash values: [15]

$$H^{(final)} = H_0' \| H_1' \| H_2' \| H_3' \| H_4' \| H_5' \| H_6' \| H_7'$$

This concatenated sequence forms the 256-bit message digest, representing a unique fingerprint of the entire input data, ensuring its integrity and authenticity.

Application of SHA-256 in Cryptocurrencies

Bitcoin and Mining. The most notable use of SHA-256 is in Bitcoin's mining process and the maintenance of its blockchain ledger. Mining involves validating new transactions and recording them on the global ledger, a task performed by solving a computationally intensive problem that requires repeatedly hashing data until a solution within specific criteria is found. The SHA-256 algorithm is integral to this process; miners use it to create a hash of the block header, which must meet a difficulty target set by the network [13]. This use of SHA-256 ensures that changing even a single bit of a transaction requires re-mining all subsequent blocks, thereby securing the blockchain against tampering [10].

Proof of Work. SHA-256 is also fundamental to Bitcoin's proof of work (PoW) system. PoW is a mechanism that deters certain types of cyber-attacks such as denial of service attacks—where a network is intentionally overloaded with superfluous requests to disrupt its operations. In this context, a "request" refers to any attempt by a user or a system to access network resources. The PoW requires network participants to do work—solving a cryptographic problem—which consumes time and computational resources, thus making it expensive and time-consuming for attackers to disrupt the network. In Bitcoin, PoW involves the computation of SHA-256 twice—double hashing—to increase the computational cost and secure the network effectively [17].

Address Generation. Besides mining and network security, SHA-256 is employed in Bitcoin for generating wallet addresses through a process called hashing public keys. In this process, SHA-256 is used in conjunction with RIPEMD-160, another hash function, to create what is known as the wallet address. This method not only helps in shortening the public key but also adds an additional layer of security by concealing the original public key, protecting user identities [2].

4.2 BLAKE

BLAKE is a cryptographic hash algorithm notable for its efficiency and security. Designed for the SHA-3 competition but not ultimately selected, BLAKE has since been refined into the BLAKE2 version, which includes BLAKE2b and BLAKE2s[3]. The BLAKE-256 variant processes 512-bit blocks into a 256-bit hash output, similar to SHA-256 in terms of output size. This section will outline the basic workings of BLAKE-256 and examine its application in digital security, highlighting how it compares to other hashing algorithms in terms of speed, security, and practicality in the cryptocurrency space.

BLAKE-256 Initial Values

Similar to SHA-256, BLAKE-256 uses the first 32 bits of the fractional parts of the square roots of the first eight prime numbers as initial values[3].

BLAKE-256 Permutations

BLAKE-256 uses ten permutations of the set $\{0, \ldots, 10\}$ to rearrange the order of processing message words [3]. This significantly contributes to the algorithm's resistance to cryptographic attacks. Each permutation uniquely alters the sequence of message blocks before they are fed into the mixing function, ensuring that the impact of any single input bit is spread across the entire hash output through complex, non-linear transformations.

BLAKE-256 Compression Functions

The compression function is central to the BLAKE-256 hashing algorithm. It takes the following four inputs[3]:

(1) A 256-bit chaining value derived from the previous hash computations or the initial values if it's the first block and denoted as $h = h_0 \parallel h_1 \parallel \ldots \parallel h_7$.
(2) A 512-bit message block - a segment of the input data processed during one iteration - represented as $m = m_0 \parallel m_1 \parallel \ldots \parallel m_{15}$.
(3) A 128-bit salt – a value for additional randomness to the input given as $s = s_0 \parallel s_1 \parallel s_2 \parallel s_3$.
(4) A 64-bit counter value, representing the length of the message, symbolized by $t = t_0 \parallel t_1$.

These inputs are combined to create a new 256-bit chaining value. The function comprises three main steps: initializing the internal state, transforming the state through a series of rounds, and finalizing to produce the output hash[3].

State Initialization. The internal state of the compression function, v, is initialized with the chaining value, salt, and counter. It maps the above inputs into the state array using a combination of direct assignments and XOR operations with constants, ensuring that every piece of input has a preliminary mixing, It is represented as a 4x4 array[3]:

$$\begin{bmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{bmatrix} \leftarrow \begin{bmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus u_0 & s_1 \oplus u_1 & s_2 \oplus u_2 & s_3 \oplus u_3 \\ t_0 \oplus u_4 & t_0 \oplus u_5 & t_1 \oplus u_6 & t_1 \oplus u_7 \end{bmatrix}$$

Round Function Iteration. In BLAKE-256, the transformation of the internal state v is performed through a series of $N = 14$ rounds to ensure comprehensive mixing and security of the input data. Each round consists of a set of operations grouped into two sequences per round to maximize diffusion and input dependency: [3]

$$G_0(v_0, v_4, v_8, v_{12}), \quad G_1(v_1, v_5, v_9, v_{13}), \quad G_2(v_2, v_6, v_{10}, v_{14}), \quad G_3(v_3, v_7, v_{11}, v_{15}),$$
$$G_4(v_0, v_5, v_{10}, v_{15}), \quad G_5(v_1, v_6, v_{11}, v_{12}), \quad G_6(v_2, v_7, v_8, v_{13}), \quad G_7(v_3, v_4, v_9, v_{14}).$$

The transform $G_i(a, b, c, d)$ is defined as follows for each set of state words during the rounds, where m_x and m_y are indices referring to the message block's words used in the current operation, selected based on round-dependent permutations: [3]

$$\begin{aligned}
a &\leftarrow a + b + (m_x \oplus c_{(r+1)\%8}),\\
d &\leftarrow (d \oplus a) \ggg 16,\\
c &\leftarrow c + d,\\
b &\leftarrow (b \oplus c) \ggg 12,\\
a &\leftarrow a + b + (m_y \oplus c_r),\\
d &\leftarrow (d \oplus a) \ggg 8,\\
c &\leftarrow c + d,\\
b &\leftarrow (b \oplus c) \ggg 7.
\end{aligned}$$

The constants c_r are integral to the non-linearity of the hash function, derived from the cube roots of prime numbers [3]. They vary with each round to ensure that the output does not exhibit symmetrical properties. Each G function call integrates a pair of message words m_x and m_y with these constants, employing bitwise rotations (denoted by $\ggg$) to cyclically shift bits, enhancing the interdependence of the hash output on the input data [3].

Finalization. In the finalization phase of the BLAKE-256 algorithm, the output hashing value $h' = h_0||\ldots||h_{n-1}$ is derived by performing an $\oplus$ operation on the final state values, the initial state h and the salt s, as shown in the following equations[3]:

$$h'_i = h_i \oplus s_i \oplus v_{i+8} \quad \text{for } i = 0, 1, \ldots, 7.$$

This step is important to ensure the non-reversibility of the hash function, making it infeasible to derive the input from the output.

After performing the compression functions, the algorithm goes into the iteration mode, where the state v undergoes a series of transformations across multiple rounds [3]. This iterative process is crucial for diffusing the properties of the input message thoroughly within the hash state, enhancing the security of the final hash output.

Data Padding. The padding process in BLAKE-256 ensures that the message length is a multiple of the block size, which is 512 bits. This is achieved by:

(1) Appending a single '1' bit immediately after the end of the message.
(2) Adding '0' bits until the length of the padded message is 64 bits less than a multiple of 512.
(3) Finally, appending the length of the original message, formatted as a 64-bit little-endian integer, to the end of the padded message. [3]

For example, consider padding an 8-bit message represented as '11010110'. The padded message would look like this:

$$11010110\ 1\ 000\ldots000\ 00000000000000000000000000001000$$

where the final 64 bits represent the original message length in bits, padded to align with 512 bits.

Iterated Hash Function in BLAKE-256. BLAKE-256 processes the padded message as a sequence of 512-bit blocks, employing an iterated hashing approach. Each block undergoes a compression function that iteratively updates the hash state. This method ensures that the hash outcome comprehensively represents the input data [3].

Compression Function Overview. The compression function updates the hash state by combining the current state H, the message block M_i, the salt S, and the counter t. The mathematical representation of the compression function is as follows:

$$H_{i+1} = \text{compress}(H_i, M_i, S, t)$$

where H_i is the state before processing the block M_i, and H_{i+1} is the state after processing M_i [3].

Process Details. Each 512-bit block M_i is processed in sequence:

$$H_{i+1} = \text{compress}(H_i, M_i, S, t), \quad \text{for } i = 0 \text{ to } N-1$$

where N is the total number of blocks. The initial state H_0 is set to the initialization vector. After all blocks are processed, the final hash value is derived from the last state:

$$H_{\text{final}} = H_N$$

[3]

Application of BLAKE in Cryptocurrencies

BLAKE-256 is utilized in various applications within the cryptocurrency space, leveraging its high-speed processing and robust security features. Despite not being as widely adopted as SHA-256, BLAKE-256 offers significant advantages that have been recognized by several emerging cryptocurrency platforms.

Mining. BLAKE-256 is prominently used in the mining processes of cryptocurrencies that prioritize fast block generation, which is essential for maintaining high network throughput and quick transaction confirmations. For instance, Decred incorporates BLAKE-256 to enhance the efficiency and responsiveness of its network operations, thereby supporting its hybrid proof-of-work/proof-of-stake consensus mechanism [1]. Siacoin also employs BLAKE-256 to optimize its decentralized storage solutions, ensuring fast and reliable access to data across the network [22].

Transaction Security. In the realm of transaction security, BLAKE-256 functions similarly to SHA-256 by ensuring the integrity and authenticity of transaction data. It does so by generating unique, irreversible hash values for each transaction, which are then immutably recorded on the blockchain. This application not only prevents tampering but also enhances the overall trustworthiness of the blockchain system [5].

Wallet Address Creation. BLAKE-256 is crucial in generating secure cryptographic wallet addresses by hashing public keys. This process results in wallet addresses that are not only compact but also robust against security threats, facilitating safer and more private transactions within the cryptocurrency ecosystem [5].

Due to its attributes and the ongoing evolution in digital currency technologies, BLAKE continues to be a viable candidate for broader adoption in new cryptocurrency projects, particularly those seeking an optimal balance between speed and cryptographic security [5].

4.3 Comparative analysis of SHA-256 and BLAKE-256

While SHA-256 has become synonymous with Bitcoin and established its credibility through rigorous real-world applications, BLAKE offers a refreshing perspective with design choices optimized for speed and simplicity. This comparative analysis delves deeper into the nuances of both hash functions, emphasizing their implications in the cryptocurrency space.

Complexity. The design complexity of a cryptographic hash function influences its computational efficiency and its vulnerability to attacks. SHA-256, part of the SHA-2 family, is characterized by a fixed block size and a relatively complex series of bitwise operations and permutations [15]. While contributing to its robustness, it also leads to higher computational overhead, especially in environments where rapid processing is crucial. In contrast, BLAKE's design is notably simpler and more streamlined. It employs fewer constants in its compression function and simplifies the number of unique operations involved in each round, reducing the computational burden. This simplicity does not compromise its security but makes it less prone to implementation errors, a significant advantage in cryptographic applications [5].

Speed. Speed is a critical factor in for hash functions in cryptocurrency systems, impacting block generation and transaction verification times. Empirical analyses show that BLAKE outperforms SHA-256 in terms of speed on similar hardware configurations, thanks to its simpler and more direct computation model [5]. The speed advantage of BLAKE could potentially reduce the time it takes to verify transactions and mine new blocks, offering an edge in scalability compared to SHA-256 based systems, which often face bottlenecks due to their intensive computational requirements [1].

Application Space. SHA-256 dominates the application space in established cryptocurrencies like Bitcoin, where it secures everything from transaction encoding to the proof-of-work mechanism integral to network security [13]. Its widespread adoption and testing over the years have made it a standard against which other hash functions are measured.BLAKE's adoption, although more limited, has been driven by its performance advantages. Cryptocurrencies such as Decred and Siacoin have integrated BLAKE to leverage its efficiency for faster block processing and enhanced security, demonstrating its potential in a competitive landscape that increasingly values both security and performance [1, 22].

Security. The security of a hash function is the ultimate test of its efficacy. SHA-256's security is well-tested, having withstood significant scrutiny over years of deployment in Bitcoin. Its resistance to collision and pre-image attacks makes it a benchmark for cryptographic security [15].BLAKE's security features, including built-in defenses against side-channel attacks, make it equally robust. It was designed to meet the high security standards of the SHA-3 competition, which aimed to identify hash functions that could offer security at par with or better than SHA-2 under evolving cryptographic threats [16][5].

5 CONCLUSION

This paper has presented a comprehensive comparative analysis of SHA-256 and BLAKE-256 cryptographic hash functions within the cryptocurrency space, emphasizing their roles in securing digital transactions without centralization. These hash functions enhance the security and integrity of blockchain technologies, each bringing unique strengths to the table. SHA-256, deeply embedded in the architecture of Bitcoin, offers unmatched security and reliability, tested by rigorous real-world applications. On the other hand, BLAKE-256 distinguishes itself with its design efficiency, providing superior performance, particularly in terms of speed making it advantageous for systems with high demand.

While SHA-256 continues to be the cornerstone of major cryptocurrencies, BLAKE-256 presents a viable alternative for newer digital currencies seeking a balance between speed and security. The inherent simplicity and efficiency of BLAKE-256 potentially reduce the computational overhead, offering an edge in environments where transaction speed and system scalability are critical.

REFERENCES

[1] 2016. Decred: A Hybrid Proof-of-Work, Proof-of-Stake System. https://docs.decred.org/research/hybrid-design/. Accessed: 2023-04-30.

[2] Andreas M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies.* O'Reilly Media, Inc.

[3] Jean-Philippe Aumasson, Willi Meier, Raphael Phan, and Luca Henzen. 2014. *The Hash Function BLAKE.* https://doi.org/10.1007/978-3-662-44757-4

[4] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. 2014. *The Hash Function BLAKE.* Springer. https://doi.org/10.1007/978-3-662-44757-4

[5] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. *BLAKE2: Simpler, Smaller, Fast as MD5.* Springer, Berlin, Heidelberg.

[6] Elaine Barker. 2024. *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms.* NIST Special Publication 800-175B Revision 1. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-175Br1 Available free of charge.

[7] Sourabh Chandra, Smita Paira, Sk Alam, and Siddhartha Bhattacharyya. 2014. A comparative survey of Symmetric and Asymmetric Key Cryptography. *2014 International Conference on Electronics, Communication and Computational Engineering, ICECCE 2014.* https://doi.org/10.1109/ICECCE.2014.7086640

[8] Natalia Chaudhry and Muhammad Yousaf. 2018. Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities. 54–63. https://doi.org/10.1109/ICOSST.2018.8632190

[9] Committee on National Security Systems. 2015. *National Information Assurance (IA) Glossary (CNSSI 4009-2015).* Standard CNSSI 4009-2015. Committee on National Security Systems. https://rmf.org/wp-content/uploads/2017/10/CNSSI-4009.pdf Accessed: 2024-04-02.

[10] Nicolas T Courtois and Lear Bahack. 2014. On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency. *arXiv preprint arXiv:1402.1718* (2014).

[11] Wolfgang Karl Härdle, Campbell R. Harvey, and Raphael C.G. Reule. 2020. Understanding Cryptocurrencies. *Journal of Financial Econometrics* 18, 2 (2020), 181–208. https://doi.org/10.1093/jjfinec/nbz033

[12] Fatma Mallouli, Aya Hellal, Fatimah Abdulraheem Alzahrani, A Ali Almadani, and Nahla Sharief Saeed. 2020. Comparative Study of Cryptocurrency Algorithms: Coronavirus Towards Bitcoin's Expansion. *Advances in Science, Technology and Engineering Systems Journal* 5, 5 (2020), 452–459.

[13] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf

[14] A. Natarajan, M. Motani, B. de Silva, K. Yap, and K. C. Chua. 2007. Investigating Network Architectures for Body Sensor Networks. In *Network Architectures*, G. Whitcomb and P. Neece (Eds.). Keleuven Press, Dayton, OH, 322–328. arXiv:960935712 [cs]

[15] National Institute of Standards and Technology. 2015. *Secure Hash Standard (SHS).* Technical Report. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf FIPS PUB 180-4.

[16] National Institute of Standards and Technology. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[17] Meni Rosenfeld. 2011. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009* (2011).

[18] S.L. Salas and Einar Hille. 1978. *Calculus: One and Several Variable.* John Wiley and Sons, New York.

[19] Matthew Scholl. 2014. HIPAA 2014: Safeguarding Data Using Encryption. https://csrc.nist.gov/CSRC/media/Presentations/HIPAA-2014-Safeguarding-Data-Using-Encryption/images-media/scholl_hipaa_2014_day1.pdf Accessed: 2023-04-30.

[20] Rajeev Sobti and Geetha Ganesan. 2012. Cryptographic Hash Functions: A Review. *International Journal of Computer Science Issues, ISSN (Online): 1694-0814* Vol 9 (03 2012), 461 – 479.

[21] Søren Steffen Thomsen. 2009. *Cryptographic Hash Functions.* Ph. D. Dissertation.

[22] David Vorick and Luke Champine. 2014. Sia: Simple Decentralized Storage. https://sia.tech/whitepaper.pdf

[23] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. 2016. Where Is Current Research on Blockchain Technology?—A Systematic Review. *PLoS ONE* 11, 10 (2016), e0163477. https://doi.org/10.1371/journal.pone.0163477

[24] Erkan Ünsal, Humar Kahramanli Örnek, and Şakir Taşdemir. 2023. A Review of Hashing Algorithms in Cryptocurrency. *International Conference on Frontiers in Academic Research* 1 (Feb 2023), 544–550. https://as-proceeding.com/index.php/icfar/article/view/161

[24] [12] [11] [23] [8] [9] [20] [15] [7] [3] [6] [21] [13] [10] [17] [2] [1] [16] [5] [22] [19]