Received 8 December 2023, accepted 26 January 2024, date of publication 30 January 2024, date of current version 7 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3360217

UNSOLVED PROBLEM

Can There Be a Two Way Hash Function?

TIMUCIN KOROGLU[1] **AND REFIK SAMET**[2]
[1]Computer Programming Department, Pamukkale University, 20600 Denizli, Turkey
[2]Department of Computer Engineering, Ankara University, 06830 Ankara, Turkey

Corresponding author: Timucin Koroglu (tkoroglu@pau.edu.tr)

ABSTRACT It is computationally impossible for one-way hash functions to obtain the original data again. In this study, Two Way Hash Function (TWHF) is proposed which provides the security properties of one-way hash functions. TWHF aims to hash/encrypt and decrypt input data of arbitrary length. This allows data to be encrypted or decrypted regardless of its size. The proposed method uses iteration-key and secondary-key to ensure confidentiality. In the study, strong hints were obtained that TWHF fulfils the security properties of one-way hash functions. TWHF stores the original data as small bit sets in a generated decimal number pattern. The last decimal number of the generated decimal number pattern is the hash code and the encrypted data. The contribution of the work is to propose an original data change function, data/block classification and workflow, and a functional method that has not yet been studied in the literature. The results show that the hashing/encryption process works consistently and that on average 99.83% of the data is decrypted smoothly in the decryption process.

INDEX TERMS Cryptology, hash functions, one way hash functions, steganography, two way hash functions.

I. INTRODUCTION

The increasing number of internet users shows that internet technology has become one of the most fundamental needs of global society, especially in the areas of communication, data storage, and data access. Today, when the number of individual internet users in the world has reached 66%, the amount of data generated in parallel with this has reached very large dimensions [1]. Research shows that in the digital world, where each person generates 17 megabytes of data daily, the total amount of daily data generated is over 2.5 quintillion bytes [2]. This situation leads to problems in data storage and transmission.

The first Parkinson's law states that an increase in data storage and transmission capacity doubles the need for data storage and transmission. Technology and devices are constantly being developed. Despite this development, they cannot respond to the growth rate of data. Therefore, there are problems with storing and transmitting big data with high accuracy [3]. Numerous studies have been conducted in the areas of data compression, hashing, and cryptography to solve this problem. Partial solutions to existing problems were found in these study areas. However, a reversible cryptographic hash function that combines the functions of the three fields to digest and compress the original data to bit levels has not yet been reported. The fact that a single algorithm can fulfill these functions by combining them is worth investigating. It is thought that a two-way hash function may be the methodology that can solve the problems mentioned here and the article focuses on this issue.

The current state of data compression, hash functions, and cryptographic algorithms in the literature can be summarized as follows:

Data compression algorithms aim to reduce the size of data compared with their original size by exploiting patterns that exist between data. Although data compression algorithms are highly advanced, the relationship between the original data length and the compressed data length is not the same as that in hash functions. Gupta and Nigam [4], in their research on popular lossless data compression algorithms, found that these algorithms provide an average space saving of 65.21%. These statistics indicate that the output lengths of lossless compression algorithms are considerably larger than the length of summary data provided by a hash function. In hash functions, regardless of the size of the input data, the output data are bit level and fixed length. In today's world, where

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

information is transformed into digital data, data security has become an important problem because of the increase in transmitted data and globalization of networks [5]. Unlike encryption algorithms, which are two-way, hash functions are one-way, and even if it is technically possible to hash a hash code in the reverse direction, computational power makes this impossible [6]. Therefore, the encryption and decryption of data are performed using cryptographic algorithms. With respect to the size of the input and output data in symmetric cryptographic algorithms, the size of the encrypted data is equal to that of the original data.

In this paper, it is aimed to convert a data of arbitrary size into a bit-level and fixed-length encrypted data. In this context, Two Way Cryptographic Hash Function, which can have data summarization functions, such as hash functions, and encryption functions, such as symmetric crypto algorithms, is proposed, and the concepts, functions, operations, limitations, and results of TWHF are presented.

TWHF is based on a decimal number model iteratively generated using change and reduction functions. All the input data to be hashed and encrypted are stored as small sets of bits in the appropriate decimal numbers in the number pattern. The length of the binary equivalents of the decimal numbers forming the number pattern is 52 bits. This can be explained as follows:

The decimal part of the decimal numbers can be a maximum of 60 bits in Python language coded to test the workflow of the proposed method and to provide statistical data. The length of the decimal part required for the stable operation of the system was determined to be 28 bits. The relationship between the length of the integer part of the decimal number and that of the decimal part is given by (3). According to (3), when the decimal part is 28 bits, the integer part must be 24 bits. In this case, the entire decimal data is $24+28 = 52$ bits long.

In TWHF, the binary equivalent of the last decimal number of the pattern is both the hash code and the ciphertext. Accordingly, regardless of the input length, the output is always 52 bits. In the decryption process, in which the last decimal number is the initial data, the pattern is regained by taking the inverse of the functions in the encryption/hash code generation process, and a hidden message is extracted from the pattern.

TWHF can be used in many fields such as data compression, data storage, cryptology, data digestion, and steganography. Considering only the function of the method in image steganography, it can provide the function of transmit large amounts of data with minimum distortion that may occur in the cover image, thereby increasing the success of data transmission with steganographic methods [7].

TWHF methodology includes several concepts, functions, and operations. The change function, which is an important element of the TWHF, was originally developed. The change function can be used not only in the proposed method but also in many other areas where pseudo-random data generation is important. In this study, decimal data were divided into four different data types according to their functions and structures, and meanings were attributed to each of them according to their functions. These types of data can be evaluated in future studies in various fields. In addition to the classification of the data, the data blocks were classified according to their functions to ensure the stable operation of the process. The block classification is another contribution of this study to the literature. In addition, it has been demonstrated by various tests that TWHF can have the security properties of one-way hash functions.

The remainder of this paper is organized as follows. Section II is described the background of this study. The methods proposed in the literature on cryptographic hash functions are evaluated in Section III. In Section IV, definitions of the proposed methodology, functions used, and operation of the method are described in detail. In Section V, TWHF and one-way hash functions are compared in terms of their security properties. Section VI describes the areas of possible applications of TWHF. Test results are given in Section VII and discussed in Section VIII. Conclusion is presented in Section IX.

II. BACKGROUNDS

Data security refers to the processes used to protect data throughout its lifecycle from modification and unauthorized access [8].

Several technologies and methods have been developed for ensuring security. Cryptography is one of the most secure technologies for this purpose [9]. Cryptography is the art of hiding text and dates back to Roman times and today. Its aim is to secure information [10]. Cryptography uses computer science and mathematics to fulfill these goals. Fig. 1 shows how secure message transmission over an insecure channel should occur against third-party attacks. Alice and Bob are authorized to see the message content for secure communication. Eve and Mallory are third parties and are not authorized to see the content of the message.

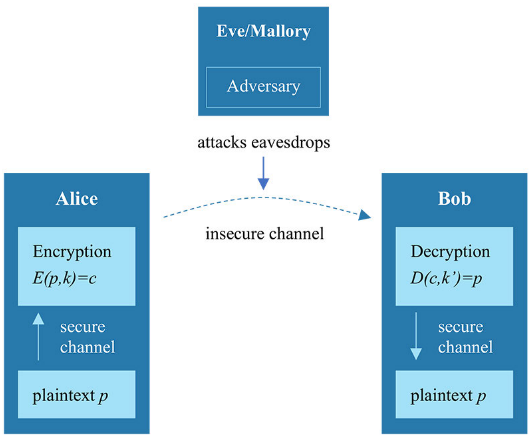

FIGURE 1. Cryptography functioning block diagram.

Currently, two different schemes, symmetrical and asymmetrical, are accepted.

According to the symmetric scheme, Alice and Bob must have the same keys to encrypt the messages. This key must be exchanged over a secure channel before communication. The asymmetric scheme originated in 1976 with Diffie–Hellman's concept of key exchange. According to this scheme, Alice and Bob must have two keys, private and public. A public key can be shared among anyone. Bob encrypts the message sent by Alice by using a public key. However, Alice can decrypt the message from Bob using only her private key [11].

Cryptography has other functions besides encrypting and decrypting data. These are listed below.

Authentication: This is the process of providing an identity for private resources that only authorized persons can access using a key.

Confidentiality: The main goal of cryptography. This guarantees that the message reaches only authorized persons using the key.

Data Integrity: This process of ensuring that data can only be changed by groups or individuals who have access to it. Consistency and accuracy of the data throughout the life cycle were ensured.

Non-repudiation: This is the process by which both the sender and the receiver confirm that the message they are communicating has been transmitted and delivered by them. In this case, the sender is obliged to acknowledge that they have transmitted the message and that the receiver has received it [12].

Fig. 2 shows the cryptography classification [13]. This classification system comprises three main techniques. These are the symmetric, asymmetric, and cryptographic hashing algorithms, respectively.

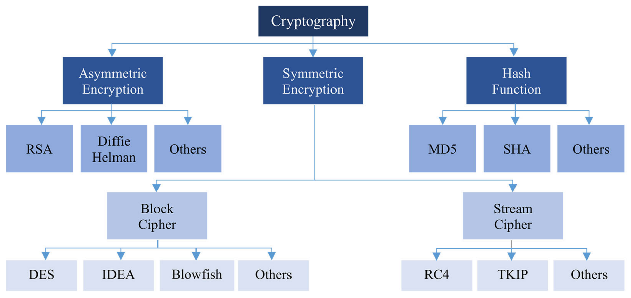

FIGURE 2. Classification of cryptography.

Symmetric encryption algorithms such as the Data Encryption Standard (DES), Rivest Cipher 2 (RC2), RC4, Blowfish, RC5, RC6, and Advanced Encryption Standard (AES) are the oldest and simplest. The main disadvantage of these algorithms is that the sender and receiver use a single secret key for encryption and decryption, respectively. The reason for this disadvantage is that attackers can eavesdrop on the communication channel and it is possible to obtain it when changing the secret key. Symmetric encryption requires a secure communication channel, such that the key can be exchanged. In contrast to symmetric encryption, asymmetric encryption algorithms, such as the Digital Signature Algorithm (DSA), Rivest-Shamir-Adleman (RSA), and Elliptic Curve Cryptography (ECC) use two keys. These are the public and private keys [14].

In encryption methods, the relationship between the key and the encrypted data must be strong. Shannon's theory explains the importance of confusion and diffusion in this relationship. Confusion refers to making it difficult to estimate the statistical relationship between the original data and encrypted data. Diffusion is the homogeneous distribution of the original data to encrypted data [15]. The strength of encryption is directly proportional to the success of the confusion-diffusion complexity and randomness of the keys. Hash functions are commonly used in encryption. It plays an important role in converting keys to fixed lengths and increasing security [16]. Hash algorithms transform messages into irreversible forms by using mathematical operations. The message used as input cannot be obtained again using hash outputs [17].

A. SYMMETRIC ENCRYPTION

Symmetric cryptography covers two broad cryptography families: block ciphers and stream ciphers. Both of these are widely used in encryption schemes. However, their data-encryption methods are quite different.

In stream cipher algorithms, each bit is separately encrypted. There are two types of stream ciphers, synchronous and asynchronous. In a synchronous stream cipher, the output of the key-stream generator and plaintext data is processed to produce ciphertext. If the output, shown by the dotted lines in Fig. 3, is returned to the keystream generator as input and participates in key generation, then the type of encryption is asynchronous.

A block cipher encrypts data with fixed lengths of n bits. The blocks are typically 64, 128, and 256 bits in length. Block cipher algorithms are iterative algorithms that convert plaintext data blocks into encrypted blocks of fixed length. Block cipher algorithms are divided into two groups according to their internal structure: Substitution Permutation Networks (SPNs) and Feistel networks [18]. Some block cipher algorithms in the literature are as follows.

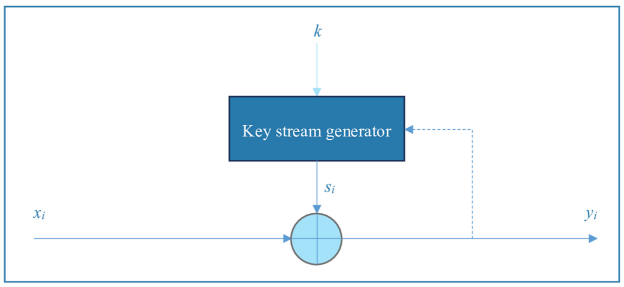

FIGURE 3. Stream cipher working principle block diagram.

DES is a typical example of a symmetric encryption algorithms that use a 56-bit long key. The DES algorithm uses two inputs. These are the plaintext and key data. The plaintext and key lengths are 64 bits. However, 56 bits of keys are used in the encryption functions. The remaining eight bits are parity bits. Fig. 4 shows a block diagram of the DES algorithm.

DES is not considered secure because of its short key length. DES encryption algorithms have been replaced by another symmetric encryption algorithm, AES [19].

The AES algorithm, which is not only secure, but also very fast, can be used in both hardware and software applications. AES completes the 128-bit data-encryption process in 10, 12, or 14 cycles. The number of cycles depends on the length of the key used. AES, which has successfully passed many security tests, is used on different platforms, particularly small devices [20].

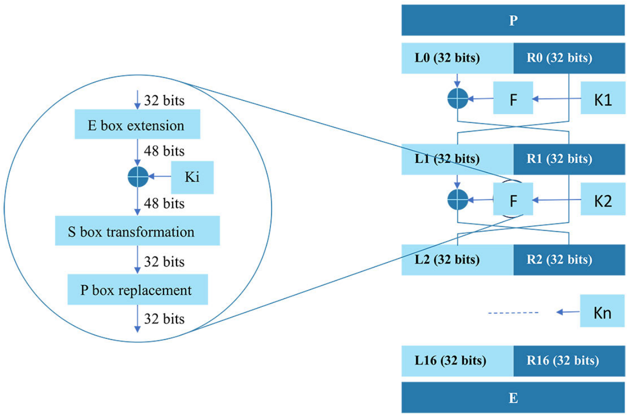

FIGURE 4. Block diagram of DES algorithm.

The Blowfish symmetric encryption algorithm was developed by Schneier in 1993 to replace DES and the International Data Encryption Algorithm (IDEA). The Blowfish algorithm uses keys ranging in length from 32 bits to 448 bits. It encrypts faster than the DES algorithm [21].

B. ASYMMETRIC ENCRYPTION

Asymmetric encryption algorithms use different keys for both encryption and decryption. The public key is used for encryption and the secret or private key is used for decryption. In the encryption process, plaintext and ciphertext must be encoded as integers. Similarly, during decryption, an integer is converted into a message. The mathematical functions shown in (1) and (2) are applied to these integers for encryption and decryption.

$$Ciphertext, C = f(publickey, P). \tag{1}$$

$$Plaintext, P = g(privatekey, C). \tag{2}$$

Function f is used for encryption and function g is used for decryption. Function f is a one-way function that prevents attackers from decrypting and allows the receiver to securely receive the message [22].

Some asymmetric encryption algorithms in the literature are as follows.

In 1978, Rivest, Shamir, and Adleman designed one of the best-known asymmetric cryptosystem algorithms, RSA. The RSA is used for key exchange, digital signing, and encryption of block data [23]. The Diffie-Helman algorithm was the first asymmetric encryption algorithm. This model was designed by Diffie and Hellman (1976). This algorithm is also known as DH algorithm. The DH algorithm is often used in key exchange applications for message encryption and decryption in insecure communication networks [24].

C. HASH FUNCTIONS

A cryptographic hash function is a one-way function that converts an input of arbitrary length into an output of fixed length. The output is often referred to as a "hash value." The general scheme of the cryptographic hash function is shown in Fig. 5 [25].

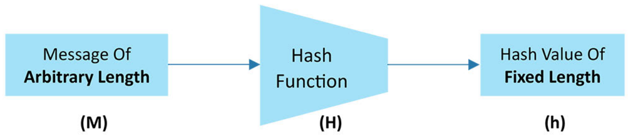

FIGURE 5. Block diagram of hash function.

Cryptographic hash functions are used in many important applications such as digital signatures, data integrity, password protection, random number generation, and authentication protocols. An error in the hash functions used in many applications negatively affects the applications in which they are used [26].

Applications are developed based on different properties of hash functions. The three basic properties of a hash function are as follows.

Pre-image resistance: Given code h as the output of a hash function, it is computationally impossible to find any input x using hash function H(x).

Second pre-image resistance: Given m as the input to a hash function, the equation H(y)=H(m) should not be computable for an input y different from m, which yields the same output.

Collision resistance: When pairs of values (x,y) are accepted as inputs, no input pair should provide equality $H(x) = H(y)$ [27].

A block diagram of the hash function security properties is shown in Fig. 6. The common hash functions in the literature are as follows.

The Merkle Damgard structure, designed by Merkle and Damgard in 1989, is used by widely used hash functions such as Message Digest 5 (MD5), Secure Hash Algorithm-1 (SHA-1), and Secure Hash Algorithm-2 (SHA-2). In the Merkle Damgard structure, which uses an iterative compression function, a message is divided into fixed-length blocks. The blocks are processed sequentially using the compression function h [28]. The Merkle Damgard structure requires an

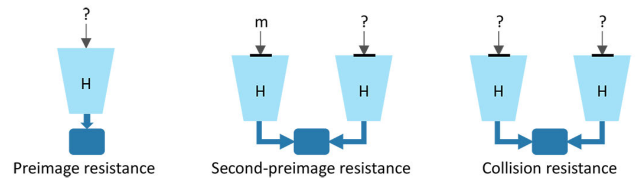

FIGURE 6. Hash function security properties.

initial vector (IV). The initial data are used to process the first block. The output of each block is processed in the next block. After the last block of the message is processed, the final output is obtained [29].

The MD5 algorithm was designed by Rivest et al. It produces a 128-bit long hash output [30].

The SHA-1 hash algorithm was designed by the National Institute of Standards and Technology (NIST) in 1995. The biggest factor in the design of this algorithm is security concerns regarding the SHA-0 algorithm. SHA-1 gives a 160 bit long hash output. This output is the result of 80 rounds of processing functions in the SHA-1 structure [31].

SHA-2 hashing algorithms are classified into SHA-256 and SHA-512 variants, which operate with words of different lengths. The SHA-256 algorithm accepts 32 bit long words as input, whereas the SHA-512 algorithm accepts 64 bit long words as input. Other differences were also observed between the groups. These are the fixed parameters and initialization values. There are four additional versions of the SHA-2 family: SHA-224, SHA-384, SHA 512/224, and SHA 512/256. These hashing algorithms differ in their initialization and output length. However, their basic structures are identical. According to this information, it can be said that the SHA-2 hash algorithm family is based on the SHA-256 and SHA-512 algorithm structures [32].

The SHA-3 algorithm was developed by Bertoni and his team. In 2015, NIST announced this algorithm, which offered a new structure. Unlike other members of the SHA family, this algorithm, called Keccak, uses a sponge structure instead of a Merkle Damgard structure [33].

III. RELATED WORKS

Today, it has become very important for researchers working in the field of cyber security to offer solutions by investigating ways to keep data safe against the increasing number of cyber-attacks owing to the increasing use of communication and data exchange over the Internet. Many effective methods against attacks have been proposed and implemented. Related work can be summarized as follows.

Rajeshwaran and Kumar [34] proposed a cellular-automata based hashing algorithm in their study. They designed a strong cryptographic hash function by providing diffusion and confusion to hash code generated using the proposed algorithm. Two basic properties of the hash function were tested to demonstrate the effectiveness of the algorithm. These are the avalanche effect and possibility of collision. In this study, these two main features were tested using two sentences containing only one different letter and two different keys for the same sentence. According to the obtained results, the avalanche impact characteristics and collision probability were strengthened. However, the evidence that these features were enhanced would be more illustrative with the use of additional datasets.

Kheshaifaty and Gutub [35] presented a multilayered system that combines captcha, cryptography, and hash functions for strong authentication. After confirming the captcha input, the password was encrypted and hashed using AES algorithm. For login confirmation, the hash code of the password must match the code stored in the system. This study is based on the existing cryptographic algorithms and hash functions. The proposed method contributes to the literature by changing only the cryptographic algorithms and hash function types. No original proposal has been presented for intermediate layers.

Li and Ge [36] proposed a cryptographic hash function based on cross-coupled map lattices to enhance the multimedia communication security. In this study, the initial values to be input into the cross-coupled maps were generated using a linear chaotic map and extended using a matrix to enhance the correlation between the original message characteristics. The intermediate hash values of the message blocks were generated using a matrix, which is the input of the cross-linked maps. To generate the final hash code, all message blocks were processed in parallel and intermediate hash values were processed logically. In this study, the generated hash code was tested for collision, confusion, and propagation properties, which a hash code should have. The results obtained were statistically better than those of the studies in this field and only contributed to the improvement of existing studies.

Ali and Farhan [37] presented an algorithm that strengthens the MD5 algorithm against attacks by dynamically extending the output of the MD5 algorithm, which fails against various types of attacks owing to the small output length in the range of 128-2,096 bits. The algorithm performs this function with the help of a key generated using the Linear-Feedback Shift Register (LFSR) for RNA encoding and the initial permutation (IP) table of the DES algorithm. The key is used to generate a random matrix. The results showed that the MD5 algorithm increased its success against attacks by increasing its output length. However, after the MD5 algorithm with 128 bit length was presented by Rivest in 1992, many hash functions were produced more successfully than MD5. The authors did not compare their proposed algorithm with other hash functions in the literature but only with the original MD5 algorithm. This makes it unclear how successful the proposed algorithm is compared with other algorithms.

William et al. [5] aimed to increase the efficiency of encryption and decryption using a hybrid algorithm comprising AES, ECC, and SHA-256 algorithms. In the proposed method, the message was encrypted using an encrypted key with the ECC algorithm in the AES algorithm. The encrypted message was summarized using the SHA-256 algorithm and prepared for the receiver. The proposed algorithm was compared with other algorithms in the literature to demonstrate that its efficiency was improved. Providing details of the algorithms compared in this study would provide a more accurate evaluation of efficiency. In addition, instead of feeding the outputs of the algorithm directly to other algorithms, the contribution level of the study could have been increased by using uniquely developed intermediate layers.

Abouchouar et al. [38] proposed a hash function that uses a noniterative structure, which differs from classical hash functions. The authors argued that classical hash functions in the literature have the same design model and are vulnerable. The main differences in the proposed method are that it does not have an initialization vector and the transformation operations in its internal structure are based on expansion rather than compression. In this method, the data entered the expansion function in parallel blocks. Each function output was XORed with the output of the next function. The XOR result of the last expansion function and previous expansion function yields the final hash code. In this study, theoretical information was provided that the proposed concept is more resilient to attacks than the classical hash functions. The properties of a hash function and its robustness against attacks must be tested using datasets. The results obtained should be compared with the hash functions in the literature based on test data, and the effectiveness of the method should be proven.

Abroshan [39] proposed a hybrid structure that combines the Blowfish, EC, and MD5 algorithms with fast and low memory consumption for cloud computing. Abroshan encrypted the data with Blowfish and the key with the EC while ensuring data integrity with MD5. Thus, they have increased safety and performance in their study. The results show that the proposed hybrid structure works slower than the AES algorithm when the data size increases. However, the resources shared in cloud computing are often extensive large. In this context, the hybrid structure proposed by the author needs to be developed.

Wang et al. [40] presented a chaotic image-encryption algorithm using SHA-256 and an iterative shift. In this study, a flat grayscale image of size $W \times H$ was used as the input, and an encrypted color image was obtained as the output. In the SHA-256 algorithm, where the original image is used as the input, the output is used as the key. Using this key, six initial values are generated as inputs to the linear chaotic map. The image is encrypted at the pixel level using a random sequence obtained from the chaotic map output and other process operations. The authors developed an algorithm that is robust to attacks by conducting experiments and security analysis. However, the analyses showed that the study produced statistical values close to those of other algorithms in the literature.

Alotaibi et al. [41] proposed a hybrid structure that combines a hash function, AES algorithm, and image steganography techniques for authentication systems on mobile devices. According to this structure, the hash code of the password was encrypted using the AES algorithm in which the username is used as the key. The encrypted password hash code was embedded in a cover image using the LSB stego technique. Tests of the proposed structure were based on the embedding time of the password hash code generated using different hash functions in the cover image. However, a comparison of the proposed structure with other studies in this field would provide better information regarding the safety of the structure.

Koptyra and Ogiela [42] presented an imagechain structure as a different type of blockchain. In the imagechain structure, the block data were embedded in the images. The blocks consists of JSON-formatted field and value pairs. The most important element of a block is the hash code of images in the chain. Each image stores the hash code of its previous image. Forgery is detected if any of the images in the image chain structure were removed or changed from the chain.

When the studies were analyzed, it was observed that the proposed methods were aimed at improving the properties of the existing cryptographic hash functions. The proposed studies were aimed at improving collision, confusion and propagation properties for hash functions, while aiming at time efficiency, low memory consumption and security enhancement for data security algorithms. To achieve these goals, researchers have proposed hybrid systems that combine algorithms from literature. They also made parametric changes in the intermediate layers of existing primitives. Most of the work done by researchers on cryptographic hash functions is aimed at improving existing methods rather than proposing new ones.

With the TWHF, an original method that has not yet been studied in the literature is proposed. The proposed method contains an original function, data classification concepts and principles, and a workflow that combines them. The main idea of this method is to generate the hash code of the data and obtain it again from this hash code. With this feature, the TWHF is suitable for use in many areas, ranging from cybersecurity to data storage. The proposed method has the potential to have a special place among other studies in the literature with a different perspective on cryptographic hash functions. The TWHF contributes as an original work, in contrast to studies in the literature that aim to improve the statistical values of existing algorithms.

IV. METHOD

In this section, the basic concepts of the TWHF, its functions, and the operation of the method are explained using block diagrams and algorithms. Fig. 7 shows the description sequence diagram of the TWHF method. The block consisted of two parts. The first is to hash and encrypt the original data, and the second is to obtain the original data from the encrypted hash code.

An overview of the TWHF method and its functions is given in Sections IV-A and IV-B respectively. The details of the functions are explained in Sections IV-C and IV-D. The

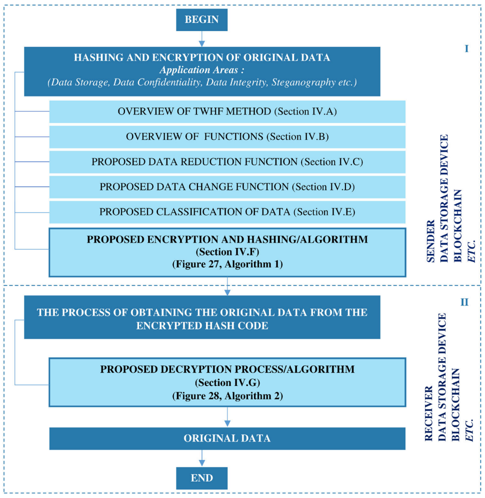
BEGIN
HASHING AND ENCRYPTION OF ORIGINAL DATA
Application Areas :
(Data Storage, Data Confidentiality, Data Integrity, Steganography etc.)
I
OVERVIEW OF TWHF METHOD (Section IV.A)
OVERVIEW OF FUNCTIONS (Section IV.B)
PROPOSED DATA REDUCTION FUNCTION (Section IV.C)
PROPOSED DATA CHANGE FUNCTION (Section IV.D)
PROPOSED CLASSIFICATION OF DATA (Section IV.E)
PROPOSED ENCRYPTION AND HASHING/ALGORITHM (Section IV.F) (Figure 27, Algorithm 1)
SENDER DATA STORAGE DEVICE BLOCKCHAIN ETC.
THE PROCESS OF OBTAINING THE ORIGINAL DATA FROM THE ENCRYPTED HASH CODE
II
PROPOSED DECRYPTION PROCESS/ALGORITHM (Section IV.G) (Figure 28, Algorithm 2)
RECEIVER DATA STORAGE DEVICE BLOCKCHAIN ETC.
ORIGINAL DATA
END

FIGURE 7. TWHF method description sequence diagram.

four data classes that play an important role in the workflow of the TWHF and their details are explained in Section IV-E. After providing the basic information, the hashing/encryption workflow and TWHF algorithm are detailed in Section IV-F. The workflow and algorithm for obtaining original data from the encrypted hash code are presented in Section IV-G.

A. OVERVIEW OF TWHF METHOD

In Fig. 8, data addition (DA), change function (CF), inverse data change function (ICF), reduction function (RF) and inverse data reduction function (IRF) are shown in the general working diagram of the TWHF method. The TWHF method generates a decimal number pattern that moves in two directions and performs operations on this number pattern. In this section, the binary equivalent of a decimal number is defined as the decimal data.

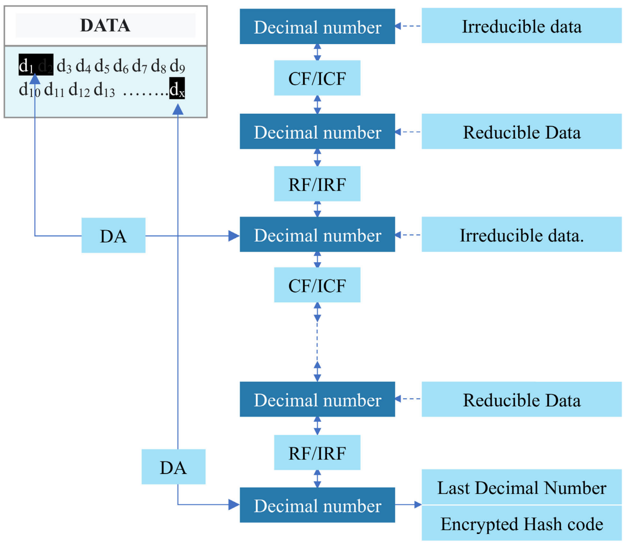

FIGURE 8. TWHF general operating principle.

The TWHF encryption and hash code generation process begins with a random decimal data. At this stage of the process, the change function is applied until it finds a decimal data that can be reduced according to Huffman encoding. This part of the TWHF functions like a pseudorandom number generator. If a appropriate decimal data is found, it is reduced according to the Hufmann coding, and a portion of the original data equal to the reduction amount is added to this decimal data in the form of concatenation. Thus, the decimal data reaches the original bit length. This process is repeated until all the bits of the original data are hidden in the number pattern. The last decimal of the pattern is the encrypted hash code.

The decryption process of TWHF is implemented by taking the inverse of the functions applied in the encrypted hash code generation process. The first decimal data of this process is the hash code, which is the last data of the encryption and hash code generation processes. Thus, the process moves in the opposite direction to that of the encrypted hash code generation process. The aim is to retrieve the number pattern and reveal all the bits of the original data hidden in the pattern.

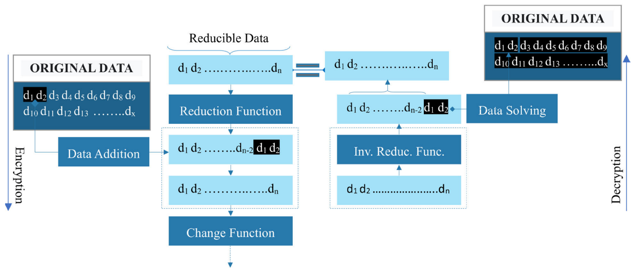

FIGURE 9. Overview of functions.

B. OVERVIEW OF FUNCTIONS

Fig. 9 details the data reduction, inverse data reduction functions and data adding-solving operations. The change function is included in the figure to find other reducible data and to show that the pattern continues.

The figure assumes that the appropriate decimal data entering the reduction function is reduced by two bits. The resulting 2-bit gap is filled with bits $d_1 d_2$ of the original data. Thus, the data collection process is completed.

The data decoding process is also included in the figure. To get the $d_1 d_2$ data embedded in the decimal data, the inverse exchange function is applied. At the output of the function, the added data and the reducible data are obtained again.

C. DATA REDUCTION FUNCTION

Huffman coding is a lossless data compression algorithm with better results and lower algorithm complexity than other data compression algorithms [43]. In terms of efficiency, coding-decoding time, compression ratio, etc., tests conducted between RLE, Shannon-Fano, Huffman, LZW and Arithmetic Coding lossless compression algorithms determined that the best performance was achieved by the Huffman coding algorithm [44]. Therefore, Huffman coding was used in the TWHF methodology.

The Huffman coding is based on a frequency table created according to the Huffman coding tree. In the proposed method, the bit sequences to be reduced are treated as groups of two. In this case, the groups can take $2^2 = 4$ values. These are $(00)_2$, $(01)_2$, $(10)_2$ and $(11)_2$. Assigning symbols to these values facilitates the use of a Huffman coding tree. Symbols form letters. The associated codes and symbols are presented in Table 1.

TABLE 1. **Codes and letter symbols.**

Code	Symbol
00	a
01	b
10	c
11	d

The symbolic representation of the data $(00\ 00\ 00\ 00\ 00\ 10\ 10\ 11\ 01)_2$ divided into groups of two is "aaaaaccdb." Huffman coding is effective for high-frequency data [45]. Success in reducing data length depends on the fact that long data blocks within the data are described with few symbols, and short data blocks with many symbols.

TABLE 2. **Frequence table.**

		a	b	c	d
		00	01	10	11
Frequency table	=	["1",	"00",	"010",	"011"]

The frequency table in which the values obtained before and after coding with the Huffman coding tree are matched is shown in Table 2. According to this frequency table, the higher the frequency of the two-bit long $(00)_2$ bits in the data, the greater is the amount of reduction.

1) DATA ADDITION

This process involves the addition of data with a length of y bits to the data reduced from x bits to x-y bits using Huffman coding. In this case, the data of length x-y reaches again a length of x bits. A block diagram of the data-addition process is shown in Fig. 10.

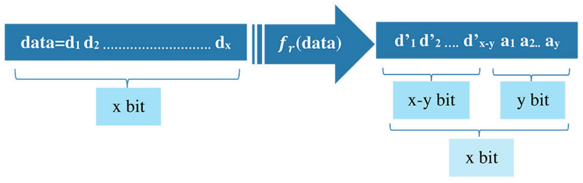
data=d₁ d₂ .... dₓ
f_r(data)
d'₁ d'₂ .... d'ₓ₋ᵧ a₁ a₂.. aᵧ
x bit
x-y bit
y bit
x bit

FIGURE 10. **Block diagram of the data addition process.**

An example of the implementation of the reduced and added data functions is as follows.

Sample data: 24 bits $(110001010010000111010000)_2 = (12\,919\,248)_{10}$.

The sample data were divided into 12 groups, each two bits long, according to the frequency table. The two-bit data in each group were matched with the code values in the frequency table. The bit length of the new group of 12 obtained as a result of the matching must be less than 24 bits. The 22 bit long reduced data obtained after matching is $(0111000010101000110011)_2$ as shown in Fig. 11. In this case, two bits of data must be added. In this example, the data to be added were assumed to be $(00)_2$. These data have been added to the LSB side. Thus, the new 24 bit long data is $(011100001010100011001100)_2 = (7\,383\,244)_{10}$.

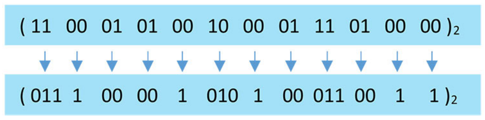

FIGURE 11. **Reduction function application example.**

2) DATA SOLVING

It is the process of separating the data as reduced data + added data. This is performed using the inverse Huffman reduction function $f_r^{-1}(x)$. A block diagram of the data solving process is shown in Fig. 12.

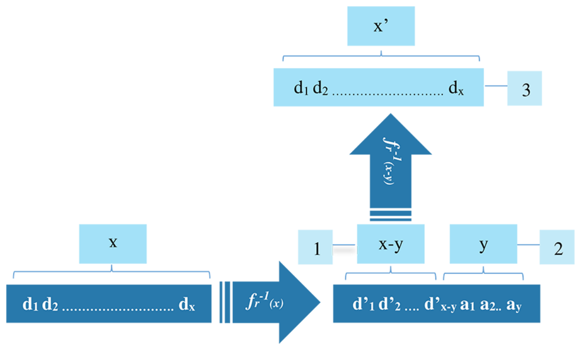
x'
d₁ d₂ .... dₓ
3
$f_r^{-1}(x\text{-}y)$
x
1
x-y
y
2
d₁ d₂ .... dₓ
$f_r^{-1}(x)$
d'₁ d'₂ .... d'ₓ₋ᵧ a₁ a₂.. aᵧ

FIGURE 12. **Data solving process block diagram.**

Data to be extracted:

1) x-y bits length reduced data,
2) y bits of added data,
3) Original data x', from which the data of length x-y are reduced.

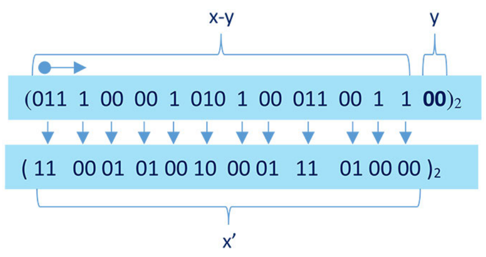

FIGURE 13. **Data solving example.**

An example of the inverse reduction function and data solving application is as follows. An example of data solving is shown in Fig. 13 for data $(011100001010100011001100)_2 = (7\,383\,244)_{10}$ in the reduced data + added data format. When implementing the inverse function, the data were grouped based on bits $(1)_2$, $(00)_2$, $(010)_2$, and $(011)_2$ in the frequency table. For the data to be considered "solvable," the total bit length in the 12 groups must be less than 24 bits. Accordingly, if $f_r^{-1}(x)$ is applied, then:

1) x-y = $(0111000010101000110011)_2$,
2) y = $(00)_2$,
3) x' = $(110001010010000111010000)_2$ = $(12\ 919\ 248)_{10}$ is found as.

D. DATA CHANGE FUNCTION

Another function used in this method is the data change function. This function was originally developed in this study. Its purpose is to change the value of the data. Without additional precautions, when the inverse of the output of the exchange function based on fractional operations is taken, the input data of the function cannot be obtained correctly. This is because of the rounding. Rounding errors are a troublesome problem in which every computer programmer must pay close attention [46]. However, when the output of the change function is the input of the inverse function, the same data must be obtained. Otherwise, the number pattern generated during the encryption process cannot be obtained during the decryption process. This renders encrypted data indecipherable.

1) MATHEMATICAL EXPRESSION OF THE DATA CHANGE FUNCTION

The function that finds the number of digits of the integer part of decimal data is shown in (3). The decimal data to be used as input for the change function were normalised between 0-1 with the normalisation function given in (4). The error amount function is given by (5) and is used to convert the error amount into an integer. The data obtained at the output of the inverse change function are not equal to the decimal data, which is the input of the change function owing to rounding error. This difference indicates the amount of the error.

The change function is the sum of the error and normalization functions. This is shown in (6).

The data change function equation is as follows.

d data and $d \in \mathbb{R}$.

d_x: x. data in a set of n elements and $n \in \mathbb{N}$.

d_{int}: The bit length of the integer part of the decimal data; $\{d_{\text{int}} \mid 24 \leq x \leq 52, x \in \mathbb{Z}\}$.

d_{pre}: Precision of decimal parts of decimal data.

$f_{norm}(d_x)$: Function for normalising the data to the range 0-1.

$f_{err}(d_x)$: Error amount function.

$f_{change}(x)$: Data change function.

$$d_{pre} = \lfloor \log_{10}(2^{d_{int}-1}) \rfloor + 2 \tag{3}$$

$$f_{norm}(d_x) = round(\frac{x}{(2^{d_{int}} - 1)}, d_{pre}) \tag{4}$$

$$f_{err}(d_x) = [(f_{norm}(x)(2^{d_{int}} - 1) - x)(10^{d_{pre}})] + 2^{d_{int}-1} \tag{5}$$

$$f_{change}(d_x) = f_{err}(x) + f_{norm}(x) \tag{6}$$

2) MATHEMATICAL EXPRESSION OF THE INVERSE DATA CHANGE FUNCTION

The inverse change function is given by (7). The integer part of the decimal data, which is the input of this function, provides the error amount, and the decimal part provides the normalized number. First, inverse normalization was applied to the decimal parts of the data. However, the output from this operation is inaccurate owing to rounding. Using the integer part of the decimal data, where the error is stored, the original data is obtained again without error.

$$f_{change}^{-1}(d_{x'}) = [(x - \lfloor x \rfloor)(2^{d_{int}} - 1)] - [\frac{\lfloor x \rfloor - 2^{d_{int}-1}}{10^{d_{pre}}}] \tag{7}$$

An example was made for the inverse change function. In this example, the length of the integer part of the decimal data d = 195.00000000 was taken as 24 bits and the decimal part was taken as 28 bits. The parameters and outputs of these functions are listed in Tables 3 and 4, respectively.

E. CLASSIFICATION OF DATA

In the encryption and decryption processes in TWHF, pattern generation is performed according to certain rules. These rules are applied according to the type of decimal data used. The data types are determined separately for integer and decimal parts of the decimal data. The data types identified in this study are divided into four classes. These are:

1) Reducible and Solvable,
2) Reducible and Unsolvable,
3) Irreducible and Solvable,
4) Irreducible and Unsolvable.

Two functions were used to identify data type. These are the reduction and inverse-reduction functions. The reduction function is first applied to determine the data type. For the data to be considered reducible, it must be reduced by at least one bit.

After this condition is satisfied, an inverse reduction function is applied to the data. Two conditions must be satisfied for the data to be considered solvable. In the first condition, when the inverse reduction function is applied to the data, the output of the function should be data bit length/2 groups according to the Huffman frequency table. The second condition is that the total bit length of the group should be less than that of the data.

Details of the data types defined within the scope of this study are as follows.

RD = Reduced Data, AD = Added Data.

1) TYPE NO 1

This data can be reduced and solved. The reduction and inverse reduction functions are applied separately to the data. If the data can be reduced and solved, they are classified as Type 1. The block diagram is shown in Fig. 14.

The bits of the data to be encrypted and hashed are added to type-1 data. This is because their size can be reduced. The data are reduced by the reduction process to bit lengths

TABLE 3. Data change function example and results.

Data change process	Description	Result
d_{int}	Bit length of the integer part of the decimal data	24
d_{pre}	Precision of the decimal part of decimal data	$\lfloor \log_{10}(2^{24-1}) \rfloor + 2 = 8$
d_x	Decimal data to change	195.00000000
$f_{norm}(195.00000000)$	$[195.00000000/(2^{24}$ -1)]	0.00001162
$f_{err}(195.00000000)$	$[(0.00001162)(2^{24}$ -1)-195.00000000) $(10^8)]+(2^{23})$	3 512 438
$f_{change}(195.00000000)$	3 512 438+0.00001162	3 512 438.00001162

TABLE 4. Inverse data change function example and results.

Data solving process	Description	Result
$f^{-1}_{change}(3512438.00001162)$	$[(0.00001162)(2^{24}$ -1)]-[(3 512 438-$(2^{24}$ /2)]/(10^8))	195.00000000

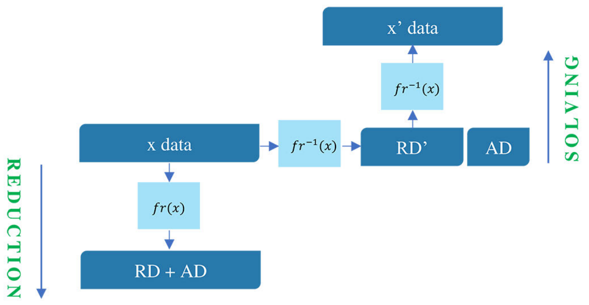

FIGURE 14. Block diagram of data type 1.

that are smaller than the original bit length. Thus, plain-text bits can be added to the reduced data based on the reduction amount.

2) TYPE NO 2

These data exhibited reducible and unsolvable properties. The block diagram is shown in Fig. 15.

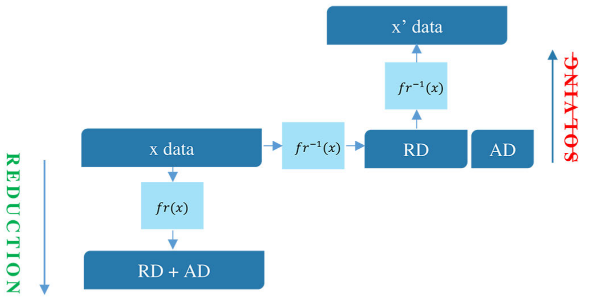

FIGURE 15. Block diagram of data type 2.

Their numbers are very small compared to those of the other types. Because this type of data can be reduced, plain text data bits can be added to the type 2 data. However, because their number is very small, they are categorized as data types for which no data can be added. In the cryptographic process, when such data are encountered, a changing function is applied instead of a reducing function.

In the decryption process, the inverse change function is applied.

3) TYPE NO 3

It is data that cannot be reduced and can be solved. The block diagram is shown in Fig. 16.

This is the most numerous data type in the dataset used in this study. Data bits cannot be added to this type of data. This is because they cannot be reduced in size. Therefore, the change function is applied. Type 3 data can be divided into reduced and added data parts, because they are solvable. These characteristics lead to uncertainty in the decryption process. This uncertainty can be explained by the scenario in which the block diagram is shown in Fig. 17. Different functions can be applied to different data types. However, data at the output of the function may be of the same type.

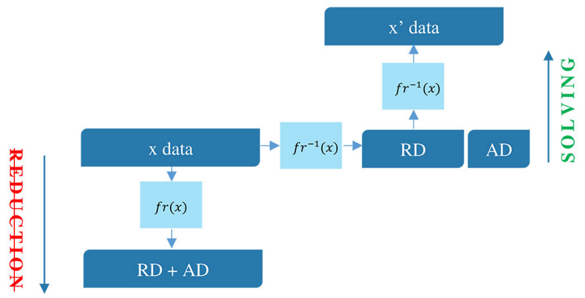

FIGURE 16. Block diagram of data type 3.

Let us consider the scenarios shown in Fig. 17. In the first case, the data for Type 1 were reduced using the function $f_r(x)$. Bit or bits equal to the reduction amount were added to the reduced data at the function output. In the second case of the same scenario, the change function was applied to the Type 3 data. In both cases, the data type derived from the function outputs was assumed to be Type 3. Because the decryption process proceeds in the opposite direction of the crypto process, in both cases of the scenario for the decryption process, type 3 data at the output of the functions $f_r(x)$ and $f_c(x)$ were encountered. Owing to the solvable nature of

Type 3 data, in both scenarios, the perception of added data within Type 3 data occurs.

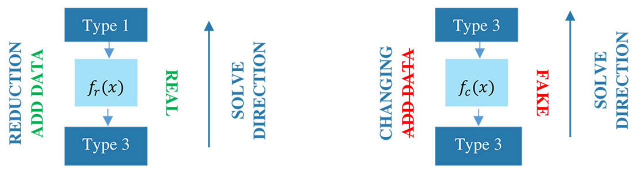

FIGURE 17. Possibilities that may be encountered in the data solving process.

In the first scenario, the LSB side of the Type 3 data bits contained plain-text bits. Such added data are characterized as "real." In the second case, no data were actually added. This is because type 3 data were obtained using the change function. However, it behaves as if it were obtained using a reduction function. In this case, it may contain data characterized as "fake." During decryption, "real" and "fake" data cannot be distinguished from each other unless additional rules are established. In this study, the amount of "fake" data was significantly reduced by adding control data to the decimal part of the decimal data. However, this situation has not yet been completely eliminated. Therefore, an extra data block was created where "real" data is mapped to $(1)_2$ and "fake" data to $(0)_2$. This data block is referred to as "crypto raw data."

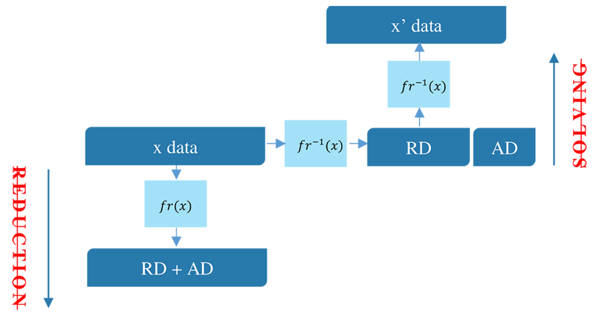

FIGURE 18. Block diagram of data type 4.

4) TYPE NO 4

These data, which are irreducible and irresolvable, are characterized as Type 4. The block diagram is shown in Fig. 18. No output is produced when the reduction function and data-solving operations are applied to this type of data. Therefore, only the change function is applied in the crypto process. If such data are revealed during decryption, there is no possibility of uncertainty.

5) STATISTICS OF DATA TYPES

Statistical data were obtained using a 24-bit dataset. In this case, the dataset covers integers in the range 0-16 777 215. The application was developed using Python. Integers in the range [0 - (2^{24}-1)] were sent parametrically to the data-type determination function, and the data type was obtained as the return value. The number of data types varies depending on the frequency table used. The frequency table used and number of data types obtained are listed in Table 5.

TABLE 5. Classified counts of data types obtained based on 24-bit long data.

Frequency Table = [“0” “11” “101” “100”] (00 01 10 11)		
Type no	Description	Number of data types
1	Reducible and Solvable	1 907 370
2	Reducible and Unsolvable	12 773
3	Irreducible and Solvable	12 949 703
4	Irreducible and Unsolvable	1 907 370
Ratio of type 3 data to type 1 data.		12 949 703/ 1 907 370=**6.8**
Total		2^{24}=16 777 216

F. HASHING/ENCRYPTION PROCESS

1) DATA LOADING FUNCTIONS AND CONDITIONS

CI: Integer part of changed decimal data, CD: Decimal part of changed decimal data.

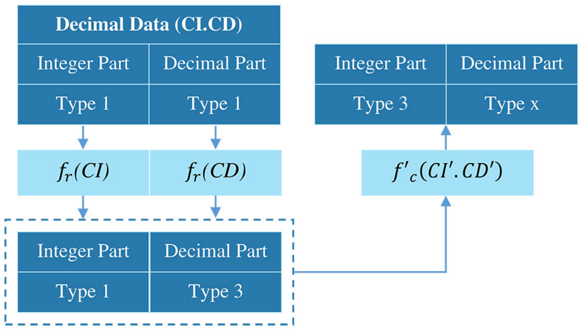

FIGURE 19. Functions and conditions used in the data loading block.

When encrypting data and generating hash codes, finding the appropriate decimal numbers to which data can be added is the most important stage of the process. The functions and conditions required for this process are shown in Fig. 19.

Plain text bits are added to the integer part of the decimal data, whereas the control data are added to the decimal part.

Condition 1 is as follows.

The first condition for adding data is that both the integer and decimal parts of the decimal data must be Type 1. This is because type 1 data are reducible.

Condition 2 is as follows.

The reduction function is applied to both the integer and decimal parts of the decimal data. Data of length equal to the length of the reduction are added to both sides of the decimal data. The added data are at bit level and on the LSB side. Consequently, both the integer and decimal parts of the decimal data reached the original bit length. These operations change the numerical values of the decimal data. The second condition requires that the integer part of the changing decimal data be of type 1 and the decimal part be of type 3. The data for types 2 and 4 cannot be included in the second condition, because they are unsolvable. The data for type 3 are approximately 6.8 times more than those for Type 1. The ratio is shown in Table 5. According to this ratio, conditioning the integer part of the data with type 3 instead of type 1 leads to a considerable increase in the number of "fake" data blocks. For this reason, it was preferred that the integer part be of Type 1. Type 3 was preferred for conditioning the decimal parts. This preference facilitates fulfilment of the second condition. Thus, the balance of conditions for both parts of the decimal data is preserved.

Condition 3 is as follows.

The third condition is applied to the decimal data that satisfies the second condition. For this purpose, an inverse reduction function is applied. The output of the function is decimal data. The integer part of these data must be type 3. The decimal part can be of any type. The last condition is intended to contribute to the minimization of "fake" data.

If one of the conditions is not satisfied, a change function is applied to the decimal data, which are the initial data of the block, instead of a reduction function. The number pattern continued for the next block.

An example of data loading conditions:

The decimal data $(8\ 479\ 184.15860065)_{10}$ which fulfills all of the data loading conditions, and its values at the function outputs are given in the block diagram in Fig. 20.

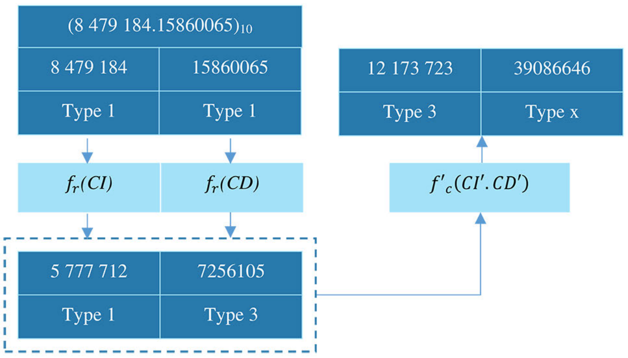

FIGURE 20. Data loading block for $(8\ 479\ 184.15860065)_{10}$ data.

In the first step of the process, type determination was performed for integer $(8\ 479\ 184)_{10}$ and decimal $(15860065)_{10}$ parts of the decimal data. Because the type of data is 1.1, the first condition was fulfilled. In the next step, the reduction function was applied to both the integer and decimal parts of the decimal data. Because the integer part of the decimal data is reduced by four bits with the reduction function, four bits from the plain text are added to the integer part. The example assumed that the data to be added was $(0000)_2$. After adding the data, the bit length of the integer part was 24 bits and the data were $(5\ 777\ 712)_{10}$. By adding the control data to the data at the output of the reduction function, value $(15860065)_{10}$ was converted to $(7256105)_{10}$. The new decimal data calculated using the reduction function and data addition were $(5\ 777\ 712.7256105)_{10}$. According to the second condition, the integer part of the decimal data generated must be of type 1 and the decimal part must be of type 3. $(5\ 777\ 712.7256105)_{10}$ fulfilled these conditions. Finally, an inverse change function was applied to the data. When the function was applied, decimal data $(12\ 173\ 723.39086646)_{10}$ were obtained. The integer part of $(12\ 173\ 723)_{10}$ must be of type 3. This condition was satisfied. The decimal part of $(3908664646)_{10}$ can be any type. Thus, all the conditions were fulfilled. The data addition process ended with the addition of four bit long $(0000)_2$ data to the integer part of the data and one bit long $(1)_2$ control data to the decimal part of the data.

2) BLOCK TYPES

All the possible block types in the proposed structure are presented in this section. Block types are determined by the types of the integers and decimal parts of the starting and ending decimal data of the block. The representation of the data types is in the format "integer data type.decimal data type." Example formats are 1.1, 1.3, 3.1, 3.3, 3.4 etc.

Block types are classified as follows.

Block start type 1.1 - Block end type 1.3 properties are as follows:

These blocks are data loading blocks. The block starts with 1.1 and ends with 1.3. There is no different data type between these two blocks. The functions applied to the block initial data (1.1) are the reduction function and data addition. The block end data must be of type 1.3 as a result of functions and operations. An inverse change function is applied to the end block data (1.3). The type of result must be in the 3.x format. The block diagram in Fig. 19, which shows the data loading functions and conditions, belongs to this block structure.

The situations that can occur in blocks of type 1.1-1.3 are listed below.

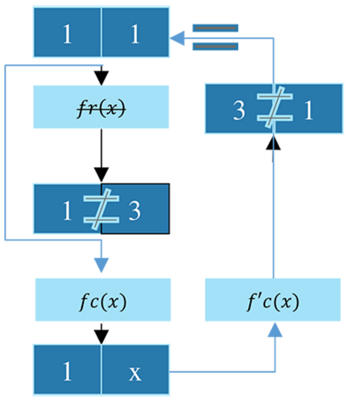

FIGURE 21. Block diagram for the case where the data of block type 1.1-1.x does not fulfil the data loading rule - I.

Failure to fulfill condition 2: If one of the conditions is not fulfilled, no data can be added to the type 1.1 block initial data. One of them (condition 2) is that the data obtained after adding data by applying the reduction function does not satisfy the required type condition (1.3). In this case, the function to be applied is a change function. The decimal data type to be obtained at the output of the change function can be 1.x. Type 1.x data do not cause the problem of "fake" data. Of these data types, 1.1/1.2/1.4 do not satisfy the second condition 1.3, so there is no possibility for "fake" data. The data of type 1.3 satisfies the second condition and provide the message that it contains added data. However, this is not the case in reality. Because decimal data is the output of the change function. In this case, the third condition prevents the emergence of "fake" data. All these cases are shown in Fig. 21.

Fulfilment of condition 2, non-fulfilment of condition 3:

When a reduction function and a data addition operation are applied to the decimal data of type 1.1, the resulting data may be of type 1.3. In this case, the second condition is satisfied. However, if the third condition is not fulfilled, the reduction function applied to type 1.1 decimal data is cancelled and the change function is applied. The decimal data at the output of the change function can be type 1.x. In such a case, there is no "fake" data. This is because the output of the function is of type 1.1 when the inverse change function is applied. This case is illustrated in Fig. 22.

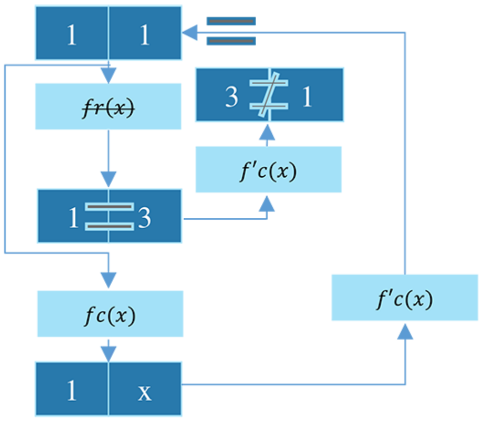

FIGURE 22. Block diagram for the case where the data of block type 1.1-1.x does not fulfill the data loading rule - II.

If the output of the first change function applied to the data with block start type 1.1 is one of the types 2.x/3.x/4.x; the block with block start type 1.1, intermediate types 2.x/3.x/4.x and end type 1.x should be defined. The properties of this block are as follows:

Such blocks are those where "fake" data may occur. The first condition for such a block to occur is that the initial block type is 1.1, and one of the data-load conditions is not fulfilled. Because data loading is not possible, a change function is applied to the initial block data. The other condition for the block to be formed is that the output of the first change function applied must be of a data type other than 1.x. These are 2.x/3.x/4.x types. This type of data does not terminate the block. The block terminates when data of type 1.x are obtained.

If the block terminates with one of the types 1.1, 1.2, or 1.4, no "fake" data are output. However, terminating the block in type 1.3 may cause "fake" data. When the inverse change function is applied to these data, type 3.x data is obtained, and if the control data condition is met, the block is considered as a data loading block. This consideration leads to the application of an inverse reduction function instead of an inverse change function during the decryption process. Applying the wrong function causes the pattern to be generated incorrectly upwards from this block, and the decryption process fails. The temporary solution to this problem until the TWHF reaches its ideal operating state is provided by "crypto raw data" as follows.

During encryption, blocks with "fake" data are mapped to $(0)_2$ and blocks with "real" data are mapped to $(1)_2$. This map is used in the decryption process. Thus, no errors were made in the inverse functions, and cryptographic data are decoded.

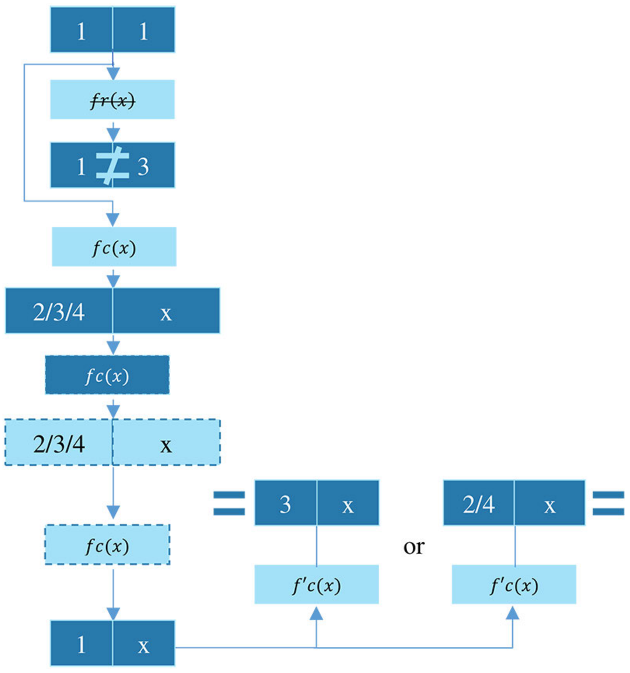

FIGURE 23. General representation of rules and behaviour for blocks of type 1.1-.....2.x/3.x/4.x.....-1.x.

Fig. 23 shows the behavior of the blocks with start type 1.1, intermediate types 2.x/3.x/4.x, and end type 1.3. The data loading conditions were not fulfilled in these blocks. Therefore, in the process of encryption and hash code generation, the change function was used in the blocks. The block ends when the first data of type 1.x at the output of the iteratively applied change function is found.

Fig. 24 shows that the end data of both blocks are of type 1.3. Fig. 24 shows that one of the two blocks cannot contain "fake" data, while the other block can contain "fake" data. It can be seen that the block without "fake" data has one of the 2.x/4.x data types before its last data. When the inverse change function is applied to the last data, the type 2.x/4.x cannot be equal to the type 3.x, so there is no "fake" data in this block.

It is possible that the block on the left in Fig. 24 is "fake" data. This block fulfills the first two of the three conditions. The first is that the end data of the block are of type 1.3 and the second is that the data at the output of the function are of type 3.x when the inverse change function is applied to the end data of the block. If the control data loaded into the decimal part of the type 1.3 data also satisfies the condition, the type 1.3 data is "fake." The control data is added to the decimal part of the decimal data in the form of bit sequences of $(1)_2$ according to the amount of reduction. If all the control bits obtained in the decryption process consist of $(1)_2$ bits, the condition related to the control bits is fulfilled. In this case, the data is treated as "real" even though it is not. For example, if the amount of reduction of the decimal part of the data of type 1.3 is 3 bits, the control data must be $(111)_2$. The 3-bit control data that does not provide this can be $(000)_2$, $(001)_2$, $(010)_2$, $(011)_2$, $(100)_2$, $(101)_2$, and $(011)_2$. If any of this control data is accessed during the decryption process, it is revealed that the data is not "real." However, if the value of $(111)_2$ is obtained as the control data, type 1.3 is considered "fake."

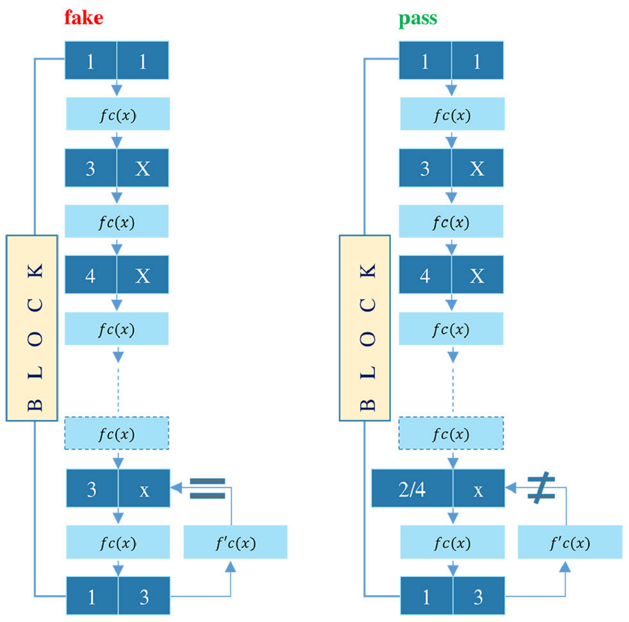

FIGURE 24. Examples of the behaviour of blocks of type 1.1-...2.x/3.x/4.x...-1.3.

The last data types of the blocks shown in Fig. 25 are 1.1, 1.2, and 1.4. As the end types of these blocks are not type 1.3, the second condition is not satisfied. Therefore, "fake" data does not occur in this block type. Because the second condition is not fully satisfied, the third condition does not need to be verified.

Block initial type 2.x/3.x/4.x and end type 2.x/3.x/4.x specifications are as follows.

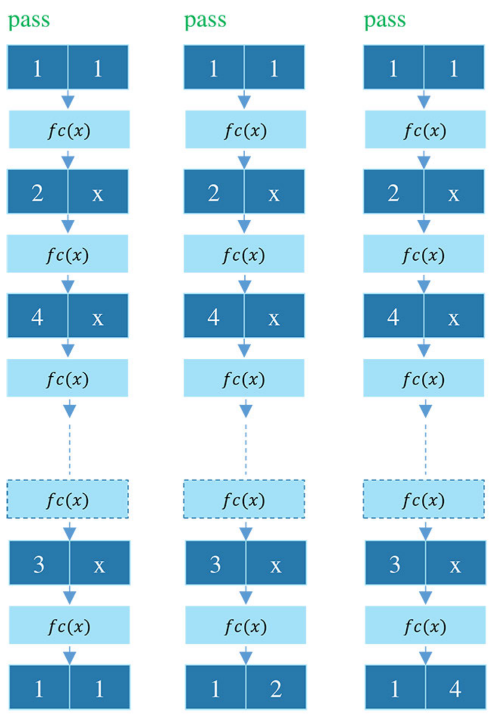

FIGURE 25. Example demonstrations of all behaviours for blocks with 1.1-..2.x/3.x/4.x..-1.1/2/4.

Blocks with block initial type 2.x/3.x/4.x are the most produced blocks. They do not contain "fake" or "real" data because they do not meet the conditions. All the functions applied in such blocks are change functions. Similarly, all the functions in the decryption process are inverse change functions. Because there is no uncertainty in the functions, the entire block is solved without error. A block whose initial type is type 2.x/3.x/4.x must end with data of type 2.x/3.x/4.x. An example of this block type is shown in Fig. 26.

3) BLOCK DIAGRAM DESCRIPTION OF THE ENCRYPTION AND HASH CODE GENERATION PROCESS

A block diagram of the encryption and hash code generation process of TWHF is shown in Fig. 27, and the algorithm used is given in Algorithm 1.

The process begins by applying a change function to the randomly selected decimal data. The data at the output of the function are the initial data of the first block to be created. The type of initial data determines whether the data can be added to the block, the function of which is to be applied, the block length, and the data type with which the block ends. The initial data can have three different forms: These are data types 1.1, 1.2/1.3/1.4, and 2.x/3.x/4.x. From these types, data can only be added to blocks with initial data type 1.1.

All situations that may occur in the encryption and hash code generation process of the model were explained according to the block diagram in Fig. 27.

Flow-1 - Block initial data type 1.1 and providing all conditions: The first type of data to be checked is 1.1. If the

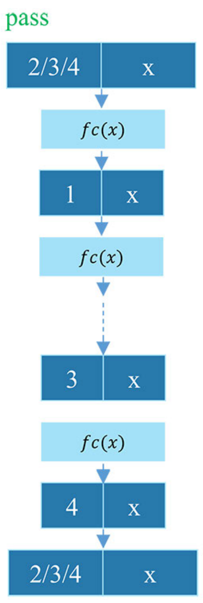

FIGURE 26. Example demonstration of rules and behaviours for 2.x/3.x/4.x-2.x/3.x/4.x block types.

Algorithm 1 TWHF Encryption and Hash Code Generation Process

```
Input: Idd: Initial decimal data
pt: Plain text
Output: ch: Chiphertext and hashcode
Definitions:
$f_r$ : Reduction function
$f_c$: Change function
$f_t$: Type determination function
rd: Reduced data, cd: Changed data, cdt: Control data
dlc: data loading conditions
fds:fake data status
Set Initial Parameters: idd, pt while(length(pt)length(added data))
do
    cd ← $f_c(cd)$          Flow1 operations
    if $f_t(cd) == 1.1$ then
        rd ← $f_r(cd)$
        Add pt and ctd bits in rd
        if (dlc == True) then
            Mapping process "real"
        Else
            Flow2 and Flow 2.1 operations
            cd ← $f_c(cd)$
            if $f_t(cd)! = 1.x$)  then
                while($f_t(cd)! = 1.x$) do
                    cd ← $f_c(cd)$
                End
                if (fds==True) then
                    Mapping process "fake"
                End
            End
        End
    Flow3 operations
    else if ($f_t(cd) == 1.2$ or $f_t(cd) == 1.3$ or $f_t(cd) == 1.4$) then
        cd ← $f_c(cd)$
        if ($f_t(cd) = 1.x$) then
            Flow3.1 operations
            while(cd! = 1.x) do
                cd ← $f_c(cd)$
            End
            if (fds == True) then
                Mapping process "fake"
            End
        End
    Flow4 operations
    else if ($f_t(cd) == 2.x$ or $f_t(cd) == 3.x$ or $f_t(cd) == 4.x$) then
        cd ← $f_c(cd)$
        while($f_t(cd)! = 2.x$ or $f_t(cd)! = 3.x$ or $f_t(cd)! = 4.x$ ) do
            cd ← $f_c(cd)$
        End
    End
End
```

data are of type 1.1, the reduction function is applied to both the integer and decimal parts of the data. Plain text bits are added to the integer part of the decimal data, whereas the control data are added to the decimal part. The process continues with the verification of the second and third data-loading conditions. If these conditions are fulfilled, block mapping is performed and the block is terminated. Since the block ends with the addition of data, it is considered "real" and the data $(1)_2$ is added to the map. Subsequently, the workflow is redirected to Connection 1. A change function is applied to the end data of the block to obtain the initial data of the next block.

Flow 2 and Flow 2.1 - Block initial data type 1.1 and non-compliance with the conditions:

First, a reduction function is applied. The data loading conditions are then checked. If one of the conditions is not fulfill, the reduction function applied to the initial block data is cancelled and the change function is applied (Flow-2). If the output of the change function is type 1.x, the block is terminated. The flow continued from Connection Number 1.

If the output of the change function is not of type 1.x, the block is not terminated and the flow is redirected to Connection 2 (Flow-2.1). The application of the change function continues until data of type 1.x are obtained. If data of type 1.x are obtained, the block is terminated. If a block is terminated in this manner, then two different situations arise. If the block ends with one of the types 1.1, 1.2, or 1.4, the workflow is redirected to Connection 1 to receive the next block's initial data and continues from this point.

If the block ends with type 1.3, it is checked for "fake." If all conditions are fulfill, the block is characterized as "fake" and bit $(0)_2$ is added to the map. If the conditions for "fake" data are not fulfill, the workflow continues from Connection 1.

Flow-3 and Flow 3.1-Block initial data type 1.2/1.3/1.4:

If the initial data of the block are of type 1.2/1.3/1.4, the first condition for loading the data is not fulfill. The flow continues with Connection number 2. In this case, a change function is applied. If the output of the first applied change function is of type 1.x, then the block is terminated (Flow-3). If the block ends this way, there is no "fake" data. This is

Algorithm 2 TWHF Decryption Process

Input: *edd: End decimal data*
Idd: Initial decimal data
Output: *pt: Plain Text*

Definitions:
f_r^{-1}: *Inverse reduction function*
f_c^{-1}: *Inverse change function*
f_t: *Type determination function*
cd: Changed data
dlc: data loading conditions
rs: Real Status

```
Set Initial Parameters: edd
Idd←edd
cd ←$f_c^{-1}(edd)$
while(cd! = Idd)) do
        cd ←$f_c^{-1}(edd)$
        Flow1 and Flow 1.1 operations
        if ($f_t(cd)$ == 1.3) then
                 if (dlc == True) then
                         if (rs == True) then
                                 Add crypto data bit to pt
                         Else
                                 cd ←$f_c^{-1}(cd)$
                                            while ($f_t(cd)$! = 1.x)
                                      cd ←$f_c^{-1}(cd)$
                                     End
                         End
                 Else
                         cd ←$f_c^{-1}(cd)$
                               while($f_t^{-1}(cd)$! = 1.x)
                                 cd ←$f_c^{-1}(cd)$
                                     End
                 end                  Flow2 operations
        else if ($f_t(cd)$ == 1.1 or $f_t(cd)$ == 1.2 or $f_t(cd)$ == 1.4) then
                         cd ←$f_c^{-1}(cd)$
                               while($f_t(cd)$! = 1.x)
                                 cd ←$f_c^{-1}(cd)$
                                     End
        Flow3 operations
        else if ($f_t(cd)$ == 2.x or $f_t(cd)$ == 3.x or $f_t(cd)$ == 4.x) then
                           cd ←$f_c^{-1}(cd)$
                                 while($f_t(cd)$! = 2.x or $f_t(cd)$ = 3.x or $f_t(cd)$
= 4.x)
                                 cd ←$f_c^{-1}(cd)$
                                        End
        End
End
```

because the third condition is not fulfilled. After the block is terminated, the flow is redirected to Connection 1 and continues from this point. The block does not terminate if the output of the first change function applied to it is of type 2.x/3.x or 4.x. To terminate the block, the change function is iteratively applied until data of type 1.x are obtained. If one of the data types 1.1/1.2/1.4 is obtained at the function output, the block terminates. "Fake" data does not occur in this case. However, if type 1.3 are obtained at the end of the iteration, the status "fake" data are checked. If the conditions for "fake" data are satisfied, the process continues with mapping. At this stage, bit $(0)_2$ is added to the map, indicating a "fake" state. If the condition "fake" data is not fulfill, the flow continues after Connection 1.

Flow 4 - Block initial data type 2.x/3.x/4.x:

If the block start type is not 1.x, the process is redirected to flow-4. In this case, the initial data type of the block is one of the data types 2.x/3.x or 4.x. The process continues with iterative application of the change function. The condition for terminating the block is that one of the 2.x/3.x or 4.x data types be encountered for the first time. There is no possibility of "fake" data in these blocks. When the block ends, the flow continues from Connection 1.

The TWHF process is completed by loading the last bit of plain text into the generated number pattern. The final decimal data of the generated number pattern are the encrypted data and hash codes.

G. DECRYPTION PROCESS

The flowchart of the decryption process was shown in Fig. 28 and its algorithm in Algorithm 2.

The flow direction is upward. The first decimal data processed during decryption are the last decimal data of the number pattern created during the encryption. In the decryption process, the inverse of the change and reduction functions is used. The goal is to generate the same numeric pattern created during the encryption process.

All the situations that may occur in the decryption process block diagram of the model and their descriptions are listed below.

Flow-1 (Block 1.x-1.3 data decryption block) or Flow-1.1 (Block 1.x-1.3 or Block 1.x - 2.x/3.x/4.x - 1.3):

The decryption process uses the end data of the encryption and hash code generation processes as initial data. The process begins with the application of the first inverse change function to this data. It is checked whether the type of the changing data is of type 1.3. If this condition is fulfilled, it is checked whether the other conditions are also fulfilled. The first condition is that when the inverse reduction function is applied to data of type 1.3, the resulting initial block data are of type 1.1. Another condition is that, if the inverse change function is applied to type 1.3, the output must be of type 3.x. In addition, the control data in the decimal part of the type 1.3 data must consist entirely of bit sequence $(1)_2$. If all the conditions are met, the "real/fake" status of the data is determined by looking at the map. In the map, if the next data are $(1)_2$, it is treated as "real" data, and the plain text bits loaded into the integer part of the type 1.3 data are parsed. Thus, during the encryption process, a few bits of data from plain text loaded into the block are obtained.

However, if the next data in the map are $(0)_2$, the data are considered "fake." The inverse reduction function applied to type 1.3 data is cancelled, and the process continues from Connection 2 (Flow-1.1). From this point on, the inverse change function is iteratively applied until it reaches data of type 1.x. If data of type 1.x are obtained, the block is terminated. For the next block of operations, the process continues from Connection 1.

According to the decryption direction, if the last data of block (1.3) do not fulfill one of the data loading conditions, the process continues from Connection 2 (Flow-1.1). For type 1.3 data, the inverse change function is applied iteratively instead of the inverse reduction function. The iteration ends when data of type 1.x are reached. For the next block of operations, the process continues from Connection 1.

Flow-2 (Block 1.x – 1.1/1.2/1.4) or (Block 1.x – 2.x/3.x/4.x – 1.1/1.2/1.4):

According to the decryption direction, if the last data of the block is not of type 1.3, it is checked whether the data are of types 1.1, 1.2, or 1.4. If the condition is fulfilled for one of these types, then the iteration in which the inverse change function is applied begins. The iteration ends when data of type 1.x are found, and the block is terminated. The process continues from Connection 1.

Flow-3 (Block 2.x/3.x/4.x - 2.x/3.x/4.x):

According to the decryption direction, if the last data of the block are of type 2.x/3.x/4.x, the process continues with the iteration in which the inverse change function is applied. The first data of type 2.x/3.x/4.x to be obtained at the output of the inverse change functions terminates the block. After the end of the block, the flow continued from Connection No. 1.

In Table 6, an example for the generation and decoding of the encrypted hash code of $(0011)_2$ data with TWHF is shown by block types. Some abbreviations are used in this table. These abbreviations are Fake/Real/Pass (F/R/P), Embedded Data (ED), Crypto Raw Data (CRD), Data Embedding (DE) and Situation (St). The RF/CF used in the table is used in the encrypted hash code generation process and IRF/ICF in the decryption process. Since the proposed method is two-way, these functions were given together. When analyzing the example, RF/CF is considered for encrypted hash code generation and IRF/CF for decryption.

The decimal numbers forming the pattern are given in the format "integer part type.decimal part type." Thus, it was easier to follow the pattern and flow. Each flow in the table also represents a block type. The types of decimal data between the beginning and end of the blocks were randomly selected in accordance with the block types classified within the scope of the study. For example, iterations 5-7 are given by the "type sequence" 1.1-3.1-1.1. However, this "type sequence" could have been 1.1-4.2-1.1 or 1.1-2.3-1.1 etc. The flows in the table were chosen in an order that facilitates the understanding of the method. Only the important iteration intervals are described below.

In iterations 1-2, the data $(0)_2$ on the MSB side of the original data $(0011)_2$ was embedded in the data of type 1.3. In this block, it is understood that all conditions are satisfied, and the reduction amount is one bit. In addition, mapping was performed with a value of "1" for "crypto raw data." In the 3rd and 4th iterations, because the conditions were not met in the block starting with Type 1.1, the change function was applied instead of the reduction function. In the range of iterations 8-12, there is a possibility that the decimal data in the 12th iteration is "fake" data because it has type 1.3. In this example, it was accepted that the conditions for "fake" data were not met. Therefore, the block was characterized as "pass" and no "0" was added to the "crypto raw data" bit sequence. In the range of the 21-27th iterations, the 27th iteration is of type 1.3. There is a possibility of "fake" data in this block. Here, it was accepted that the conditions for being "fake" data were met and mapping was done with a value of "0" for "crypto raw data." Similarly, in the 33-36th iteration interval, the data of type 1.3 in the 36th iteration was considered "fake" and mapped with "0." The data at the 45th and 46th iterations were assumed to meet the conditions and the reduction amount was assumed to be two bits. Therefore, the next data $(01)_2$ of the original data was embedded in this block and mapped with "1." The last data embedding block in the example is at iterations 53 and 54. The reduction amount for this block was accepted as one bit. Therefore, the last bit of the original data $(1)_2$ was embedded here and mapped to "1." With the last iteration, the "crypto raw data" become $(11001)_2$. There is no possibility of data embedding and "fake" data in flows other than the described iterations.

The flow from iteration 1 to 54 of the table covers the embedding of the original $(0011)_2$ data into the generated number pattern. The decimal data at the 54th iteration is considered to be the encrypted hash code (root). In this process, the undesired "crypto raw data" bit sequence has also emerged.

The decryption process can also be explained using this table. The decryption process starts from the last iteration (iteration 54). The iteration direction is opposite to the encryption/hash code generation process. Throughout the process, the inverse of the change and reduction functions is taken and the decimal number from the previous iteration is always obtained. In this process, all blocks are correctly solved by following the mapped "crypto raw data." When the 1st iteration is reached, the original data is obtained.

According to the data in Table 6, several iterations of an example decryption process can be explained as follows. The decryption process starts with data of type 1.3 at iteration 54. Since this data is of type 1.3, there is a possibility that it contains data. To determine this, the CRD value is checked. A CRD value of $(1)_2$ indicates that this data of type 1.3 contains a few bits of the original data. The original data is parsed. In the pattern, IRF is applied to find the data that precedes the data of type 1.3. The output of the function is the data of type 1.1 at iteration 53. If the data accessed is of type 1.1, it indicates the start of a block. The process continues by applying ICF to data of type 1.1. The output of the function is data of type 3.1 at iteration 52. The process continues until we reach the data in iteration 1.

V. TWHF AND ONE-WAY HASH FUNCTIONS

The data solving phase of the TWHF methodology resulted in an average uncertainty of 0.17%. Uncertainty is the situation of not being able to decide which function to apply

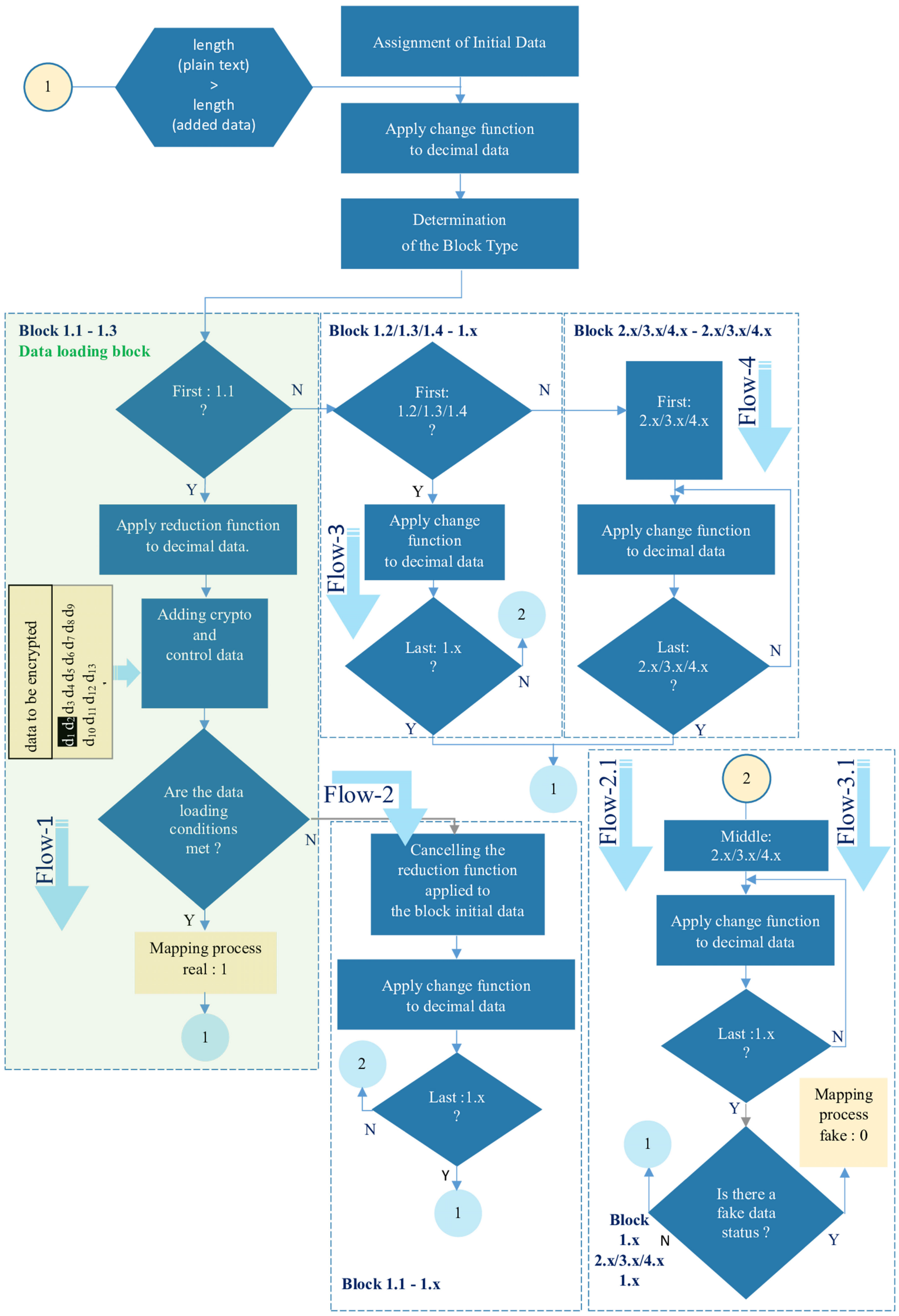
1
length (plain text) > length (added data)
Assignment of Initial Data
Apply change function to decimal data
Determination of the Block Type
Block 1.1 - 1.3
Data loading block
First : 1.1 ?
N
Y
Apply reduction function to decimal data.
data to be encrypted
d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13
Adding crypto and control data
Are the data loading conditions met ?
N
Y
Mapping process real : 1
1
Flow-1
Block 1.2/1.3/1.4 - 1.x
First: 1.2/1.3/1.4 ?
N
Y
Apply change function to decimal data
Flow-3
Last: 1.x ?
N
Y
2
Block 2.x/3.x/4.x - 2.x/3.x/4.x
First: 2.x/3.x/4.x
Flow-4
Apply change function to decimal data
Last: 2.x/3.x/4.x ?
N
Y
1
Flow-2
Cancelling the reduction function applied to the block initial data
Apply change function to decimal data
2
Last :1.x ?
N
Y
1
Block 1.1 - 1.x
Flow-2.1
Flow-3.1
2
Middle: 2.x/3.x/4.x
Apply change function to decimal data
Last :1.x ?
N
Y
Mapping process fake : 0
1
Is there a fake data status ?
N
Y
Block 1.x 2.x/3.x/4.x 1.x

FIGURE 27. Block flow diagram of TWHF encryption and hash code generation process.

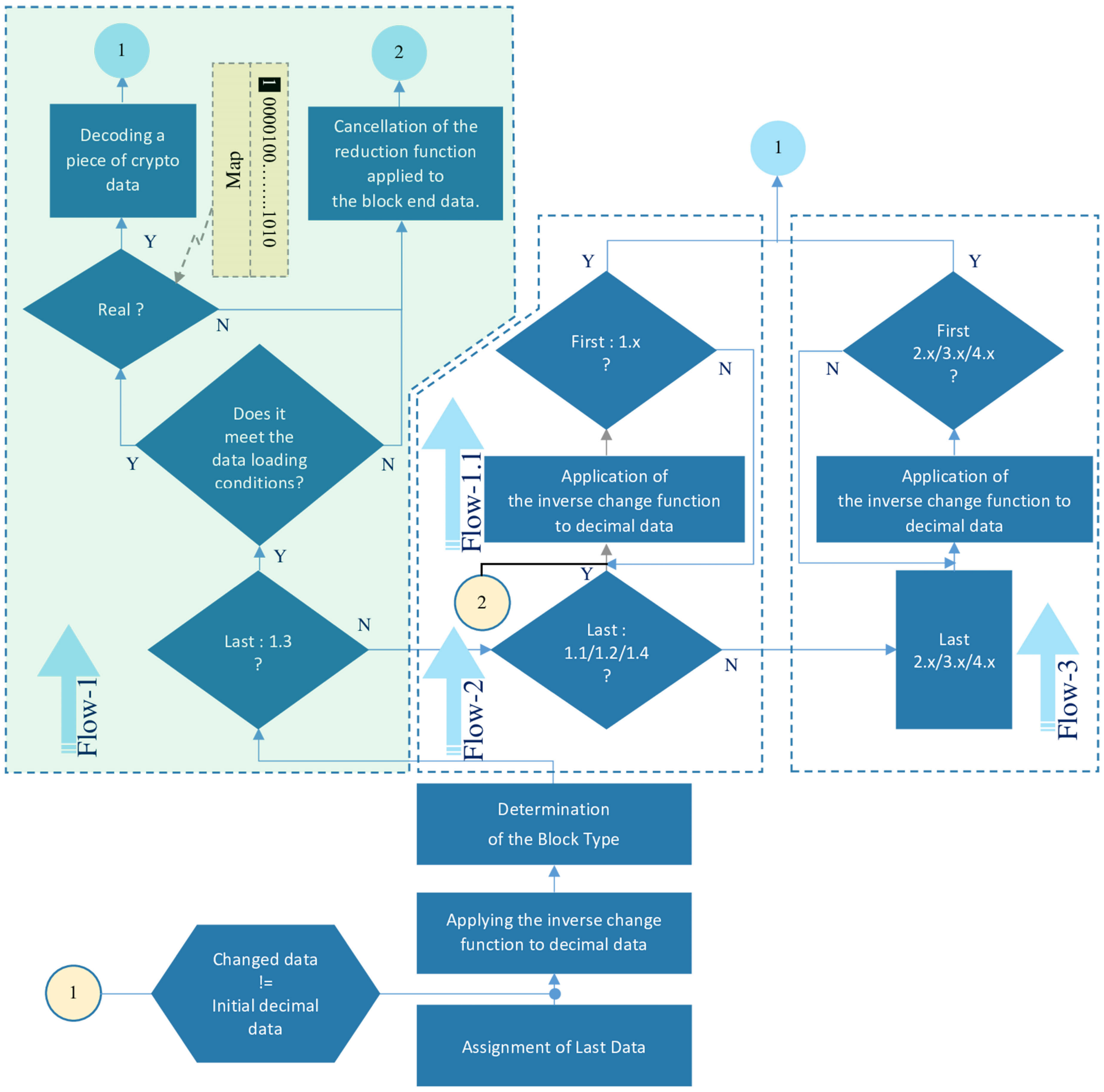
1
2
Decoding a piece of crypto data
Map
0000100......1010
Cancellation of the reduction function applied to the block end data.
Real ?
Y
N
Does it meet the data loading conditions?
Y
N
Last : 1.3 ?
Flow-1
1
First : 1.x ?
Application of the inverse change function to decimal data
Flow-1.1
2
Last : 1.1/1.2/1.4 ?
Flow-2
First 2.x/3.x/4.x ?
Application of the inverse change function to decimal data
Last 2.x/3.x/4.x
Flow-3
Determination of the Block Type
Applying the inverse change function to decimal data
Changed data != Initial decimal data
Assignment of Last Data

FIGURE 28. Block flow diagram of the TWHF decryption process.

to the active decimal data to reach the pattern's previous decimal data. These are inverse reduction and inverse change functions. As a temporary solution for this situation, a so-called "crypto raw data" was proposed. With this data as a guide, the data decryption is successfully completed. In a sense, "crypto raw data" expressing uncertainty is undesirable for part of the study. For the future work of the TWHF, the focus should be on removing the uncertainty and thus the need for this data. In the block diagram in Fig. 29, which shows the general operation of TWHF, this situation is given as "Section I."

TWHF must have the security features of one-way functions. For this purpose, the "iteration-key" and the secondary key specified in Fig. 29 as Section II can be used. "Iteration-key" structure is shown in Fig 30. An example and explanations about this structure are as follows.

In this study, the key is 256 bits long and divided into 32 groups. The key length and the number of groups can be changed according to the needs of the application in which the method is used. According to Fig. 30, the sample values taken by the groups were accepted as $(2\ 158\ 168\ 178\ 255\ 0\ 254\ 32\ 128\ 130\ 1\ 52\ 45\ 248\ 12\ 30\ 56\ 21\ 78\ 20\ 25\ 128\ 136\ 0\ 0\ 1\ 24\ 12\ 178\ 13\ 17\ 29)_{10}$. These values are the iteration numbers. A sender who knows this key applies the data replacement function starting from the root hash code. The function repeats as many times as the numbers given in the groups in "iteration-key." According to the number of iterations in the example, the hash code is obtained by applying the data change function 2431 times starting from the root hash code. Keeping the number of iterations low will also shorten the implementation time of the methodology. Malicious persons can decrypt the encrypted hash code by

TABLE 6. An example of generating and decoding the encrypted hash code of $(0011)_2$ data with TWHF.

Iterations	Flow	Block Types	F/R/P	ED	CRD	Iterarions	Flow	Block Types	F/R/P	ED	CRD
1	**Flow 1 (DE)**	1.1 **(RF)(ICF)**				28	Flow 4	2.4 (CF) **(ICF)**	P	-	-
2		**1.3 (CF)(IRF)**	R	1	1	29		1.3 (CF) **(ICF)**			
3	Flow 2	1.1 (~~RF~~) (CF)**(ICF)**	P	-	-	30		4.2 (CF) **(ICF)**			
4		1.3 (CF) **(ICF)**				31	Flow 4	3.3 (CF) **(ICF)**	P	-	-
5	Flow 2.1 St. 1	1.1 (~~RF~~) (CF) **(ICF)**	P	-	-	32		2.4 (CF) **(ICF)**			
6		3.1 (CF) **(ICF)**				33	Flow 3.1 (St. 2)	1.3 (CF) **(ICF)**	**F/P**	-	**0**
7		1.1 (CF) **(ICF)**				34		2.3 (CF) **(ICF)**			
8	Flow 2.1 St. 2	1.1 (~~RF~~) (CF) **(ICF)**	F/**P**	-	-	35		3.1 (CF) **(ICF)**			
9		3.3 (CF) **(ICF)**				36		1.3 (CF) **(ICF)**			
10		2.4 (CF) **(ICF)**				37	Flow 2.1 (St. 1)	1.1 (~~RF~~) (CF) **(ICF)**	P	-	-
11		4.1 (CF) **(ICF)**				38		2.3 (CF) **(ICF)**			
12		1.3 (CF) **(ICF)**				39		3.1 (CF) **(ICF)**			
13	Flow 3	1.2 (CF) **(ICF)**	P	-	-	40		3.4 (CF) **(ICF)**			
14		1.3 (CF) **(ICF)**				41		3.2 (CF) **(ICF)**			
15	Flow 3.1 St. 1	1.3 (CF) **(ICF)**	P	-	-	42		4.1 (CF) **(ICF)**			
16		4.2 (CF) **(ICF)**				43		2.1 (CF) **(ICF)**			
17		3.1 (CF) **(ICF)**				44		1.1 (CF) **(ICF)**			
18		2.3 (CF) **(ICF)**				45	**Flow 1(DE)**	1.1 **(RF)(ICF)**	**R**	**1**	**1**
19		4.1 (CF) **(ICF)**				46		1.3 (CF) **(IRF)**			
20		1.2 (CF) **(ICF)**				47	Flow 2	1.1 (~~RF~~) (CF) **(ICF)**	P	-	-
21	Flow 3.1 St. 2	1.4 (CF) **(ICF)**	F/P	-	**0**	48		1.4 (CF) **(ICF)**			
22		2.1 (CF) **(ICF)**				49	Flow 3	1.4 (CF) **(ICF)**	P	-	-
23		3.4 (CF) **(ICF)**				50		1.1 (CF) **(ICF)**			
24		3.1 (CF) **(ICF)**				51	Flow 4	4.3 (CF) **(ICF)**	P	-	-
25		2.3 (CF) **(ICF)**				52		3.1 (CF) **(ICF)**			
26		3.2 (CF) **(ICF)**				53	**Flow 1(DE)**	1.1 **(RF)(ICF)**	**R**	**1**	**1**
27		1.3 (CF) **(ICF)**				54		1.3 (CF) **(IRF)**			

trying the total number of iterations in the key with a brute force attack without trying the iteration numbers one by one. This can be prevented by XORing a secondary key of the same length as the hash code generated at the end of

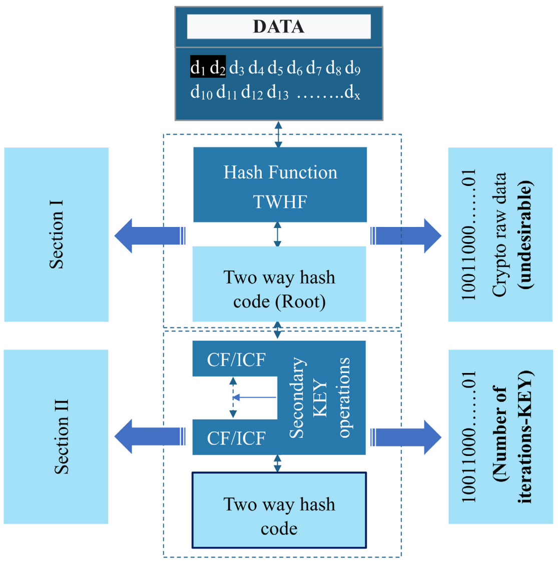

FIGURE 29. The model required to make it possible for TWHF to take on the properties of one-way functions.

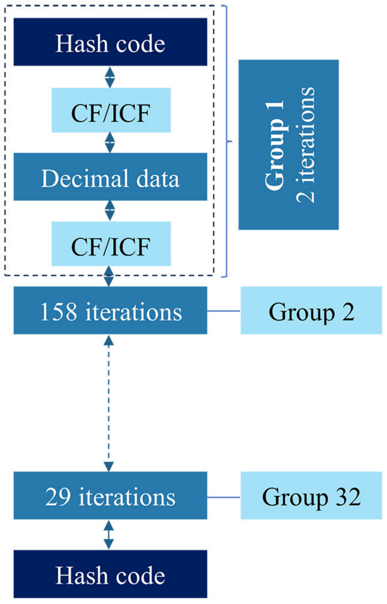

FIGURE 30. Iteration-key and secondary key workflow block diagram.

each group iteration. This statement explains that the XOR operation will be applied 32 times. By dividing the key into 32 groups, the total number of iterations was kept low, but the number of possibilities for brute force attacks was increased. This key logic, which is applied from the root hash code, can also be applied in different ways. Currently, the proposed key is sufficient to ensure that TWHF has the security properties of one-way hash functions.

If the TWHF given in Section I is applied and the decimal data in the last layer of the pattern is considered to be hash code, there is no obstacle to obtaining the original data from this data. However, the study also aims to ensure that TWHF has the security properties of one-way hash functions. For this purpose, after obtaining the root hash code of the original data, the pattern should be continued as many times as the number of iterations. In this case, the last layer of the pattern is the hash code has the security properties of one-way functions. This is because there is no meaningful relationship between the obtained hash code and the original data and the root hash code must be reached for the decryption process to work. The key must be delivered to the recipient via a secure communication channel in advance.

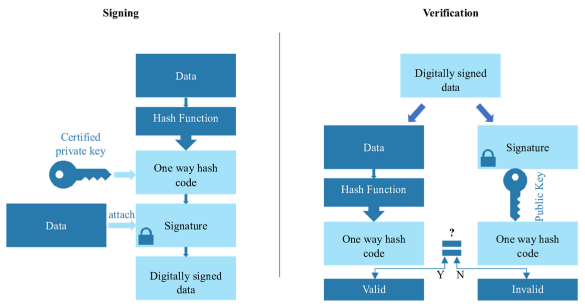

FIGURE 31. Use of current hash functions in digital signature workflow.

VI. TWHF POSSIBLE USAGE AREAS

A. DIGITAL SIGNATURE

Fig. 31 shows the current use of the digital signature workflow with one-way hash functions [47]. The purpose of a digital signature is to guarantee data integrity and authentication. In digital signature applications, the encrypted hash code of the original data is sent to the receiver along with the original data. At the receiver side, the original data were hashed again. The encrypted hash code is decrypted using a public key and the hash code is obtained again. If the results of the comparison of the hash codes are the same, the authenticity and integrity of the data are confirmed. The main process that enables authentication is encryption of the hash code with the sender's private key. Therefore, encryption plays a significant role in authentication. The hash function focuses more on the integrity of the data. The transmission of the original data with hash code to the receiver may have some disadvantages. One is the increased use of internet bandwidth and the other is the compromise of the confidentiality of the original data. For these reasons, it is more convenient to send only hash code to the receiver.

Fig. 32 shows the possible use of the TWHF method in the digital signature workflow. In a digital signature application using TWHF, the original data are not sent. Only encrypted hash code is sent to the receiving party. The original data can be retrieved from the hash code because the hash function is two-way. Thus, the confidentiality of the original data is fully guaranteed. Authentication is achieved by encrypting the hash code with the sender's private key and decrypting it with the public key, as in the current implementations. The iteration key and secondary key in the TWHF method can optionally be unused depending on the needs of the application where TWHF is used.

If the data arrives at the receiver altered for any reason, it cannot be decrypted due to the mathematically interconnected number pattern of TWHF and data integrity is determined to be corrupted. It can be said that TWHF will contribute to less bandwidth utilization of networks in applications where data exchange is frequent and authentication is required.

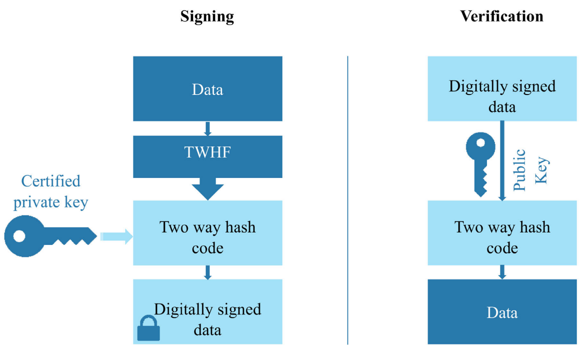

FIGURE 32. Use of TWHF in digital signature workflow.

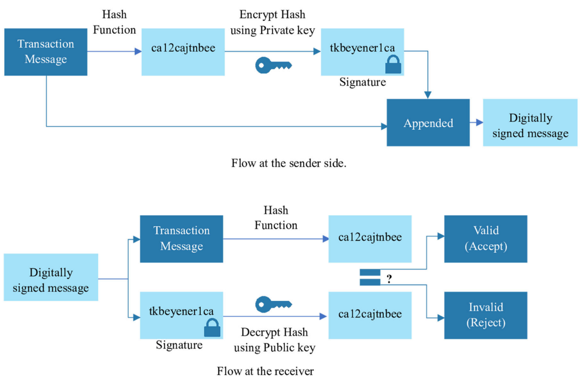

FIGURE 33. The use of current hash functions in blockchain workflow.

B. BLOCK CHAIN APPLICATIONS

Fig. 33 shows the use of a blockchain workflow with one-way hash functions [49]. Transaction messages are the smallest structures in the block system. Their role is to maintain information and records. The purpose of using encrypted hash functions in blockchain applications is to ensure data integrity and authentication as in digital signature applications. Using the hash function, the hash codes of the transaction messages on the sender side are obtained and encrypted with the sender's private key. The process continues by appending the original transaction messages to encrypted hash codes and sending them to the receiver. Thus, transaction messages are digitally signed. At the receiving end, the hash codes of the encrypted transaction messages are obtained using the public key. The authentication process continues by hashing the original transaction messages coming to the receiver side. If the hash codes obtained by the two different processes match each other when compared, data integrity and authentication are ensured. Upon authentication, the transaction messages are added to the chain structure as a new block. Fig. 34 shows the possible use of blockchain workflow with TWHF. In a blockchain application using TWHF, the process is as follows. The encrypted hash codes of the transaction messages are obtained using TWHF. The original transaction messages are not sent to the receiving side, as is the case with the current apps. In this phase, only the encrypted hash codes of transaction messages are sent to the receiver. This enables a more efficient use of the network's bandwidth and storage on the receiving side. Moreover, the confidentiality of the original transaction messages is more secure than that in traditional applications. This is because the original transaction messages are not additionally sent to the receiving side. The encrypted hash codes coming to the receiver side are extracted using a public key and the data are authenticated. The patterned nature of TWHF ensures data integrity. In the possible blockchain implementation of the TWHF method, the use of an iteration key and a secondary key is optional.

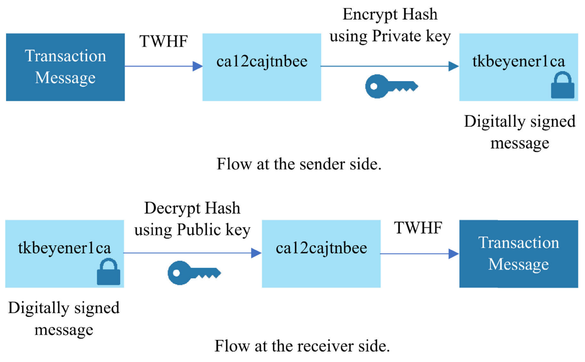

FIGURE 34. TWHF-Blockchain usage.

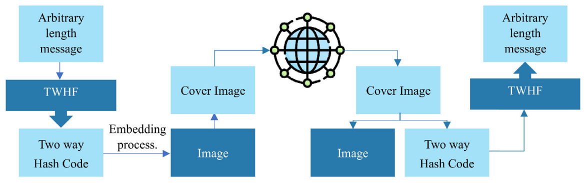

FIGURE 35. Use of TWHF in image steganography.

C. STEGANOGRAPHY

In image steganography, the message to be sent to the receiver is embedded in the image, which is called a cover object. The data embedded in the image cause a change in the pixels of the image. In other words, the original image is altered and distorted. The goal of image steganography studies is to minimize this distortion. Therefore, the more similar the cover image is to the original image, the less likely it is to attract third parties' attention. When this occurs, the message securely reaches its receiver. With the use of TWHF, the intended goal of steganography is achieved in an optimized way. This is because large amounts of bit-length data are digested and converted into fixed-length data. With some exceptions, the digestion of data is better than the output obtained by using data compression algorithms. This way the cover image is obtained with the least possible distortion and is more similar to the original image. In particular, better results are obtained in direct proportion to the increase in bit length. This is because the distortion of the image does not change even if the bit length increases.

Fig. 35 shows the possible use of TWHF in image steganography. In image steganography using TWHF, the image and the two-way hash code are parsed when the cover image reaches the receiver. The original message is then obtained again at the receiver side using two-way hash code. This is not possible with the hash functions that are currently used.

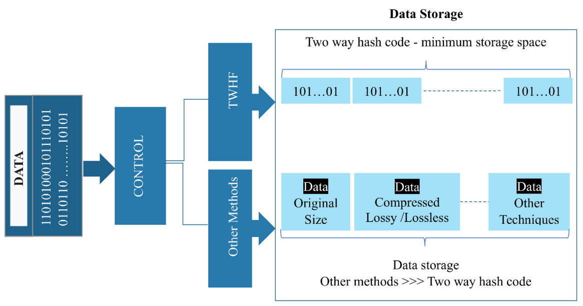

FIGURE 36. TWHF and data Storage.

D. TWHF AND DATA STORAGE

One of the areas where TWFH will benefit the most is that it will significantly reduce hardware costs by enabling data to be stored in data storage devices to take up very little space. Fig. 36 shows a scenario for the use of TWHF in data storage devices. According to this scenario, not all data must be stored using TWHF. Depending on the selection made by the control unit, the parsed data can be stored in the form of hash codes by applying the TWHF. Storing data in this manner will significantly reduce hardware costs. In addition, the reduced need for hardware devices contributes to the reduction in electricity consumption. Moreover, in an Internet environment, where data are exchanged intensively, the need for bandwidth can drop significantly. This is due to the idea that data travels through the network with hash codes rather than the original bit lengths or bit lengths that have been slightly reduced by other techniques.

VII. RESULTS

A. RESULTS OBTAINED IN THE ENCRYPTION/HASH CODE GENERATION PROCESS FOR THE CHARACTER "A"

In this study, the number of "fake" data, the number of "real" data, the length of encrypted data, the total number of iterations, and other statistical data were obtained with an applications in Python. For the statistical data, the character "A" was chosen as the plain text to be encrypted and hashed, and the decimal data of $(1\,000\,000.00000000)_{10}$ were chosen as the initial data. Statistical data are shown in Table 7.

The numerical values of "real" and "fake" decimal data are listed in Table 7. In the number pattern generated to encrypt the character "A" whose binary equivalent is $(01000001)_2$ and to generate a hash code, 4 "real" and 12 "fake" data were observed. The first "real" data was found to be type 1.3 $(463\,078.30547297)_{10}$. For this data, the reduction amount was one bit. Because the first bit of the character "A" is $(0)_2$, this data bit was stored into the data "real", whose value is $(463\ 078.30547297)_{10}$. $(12\ 652\ 818.13963791)_{10}$ decimal data was determined as the next "real" data. For this data, the amount of reduction was determined to be 2 bit. Because the next bits of character "A" are $(10)_2$, the bits loaded into the "real" data must be $(10)_2$.

$(4\ 199\ 888.77380527)_{10}$ decimal data was found to be the 3rd "real" data. The amount of reduction for this "real" data is found to be one bits and the bit $(0)_2$, which is the next data of character "A," was stored.

$(5\ 621\ 825.95322159)_{10}$ was determined to be the last "real" data. The last data to be stored must be $(0001)_2$. The amount of reduction of the last "real" data is four bits. Therefore, the remaining four bits of the character "A" were stored to the end "real" data.

To avoid increasing the number of rows, the block data characterized as "pass" were not included in Table 7. Therefore, "real" and "fake" data were shown consecutively in Table 7. However, they were not ordered in this way in the number pattern. In reality, these data were positioned among the data characterized as "pass." According to Table 7, the encrypted data/hash code was found to be $(10011100000100010111100100011111111 0100110101011000)_2$ and the crypto raw data was $(1010001001000000)_2$.

Additional statistical data obtained during the encryption and hash code generation process for the character "A" was shown in Table 8. As shown in Table, 7680 iterations were performed to encrypt the character "A." This number also shows that 7680 decimal data were generated in the number pattern. Four of these data were found to be "real" and twelve were found to be "fake." In this case, the sum of "real" and "fake" decimal numbers, i.e. the bit length of the crypto raw data, was calculated as 16. The size of the "crypto raw data" was reduced to 15 bits using a reduction function (Huffman). A total of 17 bits of data were added to the four "real" decimals. It was determined that eight of the added 17 bits belong to the plain text added to the integer part of the decimal data, and the remaining nine bits are control bits added to the decimal parts of the decimal data. Control bits reduced the number of "fake" data by about 7.4 times, from 89 to 12. Table 9 shows the statistical values obtained for three different texts.

B. ANALYSING THE IMPACT OF CONDITIONS ON METHODOLOGY

In the TWHF methodology, the number of "fake" data changes when certain conditions are changed or removed. Achieving the ideal operating state of the TWHF depends on the complete elimination of "fake" data. Therefore, the impact of certain conditions on the TWHF methodology was

TABLE 7. **"Real/fake" data values obtained during the encryption of the "A" character.**

Data to be encrypted = "A" (65)10 = $(01000001)_2$ Initial data : $(1\ 000\ 000.00000000)_{10}$ FT : 00 01 10 11 ["1" "00" "010" "011"]						
Sequence No	Fake/Real	Decimal data (Decimal number system)	Reduction Amount (bit)	Added Data (Binary)	Added Control Data (Binary)	Crypto Raw Data
1	Fake	14 024 721.46273759	-	-		0
2	Fake	8 462 144.31595595	-	-		0
3	Fake	72 540.25308223	-	-		0
4	Fake	7 708 704.72681047	-	-		0
5	Fake	5 292 350.10560543	-	-		0
6	Fake	460 893.48845001	-	-		0
7	**Real**	**463 078.30547297**	**1**	**0**	**1**	**1**
8	Fake	9 522 640.89161647	-	-		0
9	Fake	14 495 746.91813303	-	-		0
10	**Real**	**12 652 818.13963791**	**2**	**10**	**11**	**1**
11	Fake	399 390.61647479	-	-		0
12	Fake	248 325.18498111	-	-		0
13	Fake	97 358.95540271	-	-		0
14	**Real**	**4 199 888.77380527**	**1**	**0**	**1111**	**1**
15	Fake	8 701 196.27951145	-	-		0
16	**Real**	**5 621 825.95322159**	**4**	**0001**	**11**	**1**
Decimal number pattern end data (cipher text and hash code)				$(1001110000010001011110010001111111101001\ 10101011000)_2$		
				$(10\ 228\ 089.33508696)_{10}$		

analyzed. For this purpose, the process of encryption and hash code generation for the character "A" was performed by removing and changing some conditions. The results are shown in Table 10.

Condition 2 required a data load block to end at 1.3. Condition 2 was changed such that the block was terminated with type 1.1 instead of 1.3. When the process was operated, the number of "fake" data points was obtained as 47 with control data and 225 without control data. Compared to the current situation, there is a 3.9-fold increase in the number of "fake" data compared to the situation with control data and a 2.5-fold increase compared to the situation without control data.

When the TWHF process was applied with the elimination of Condition 3, the number of "fake" data was found to be 27 with control data and 201 without control data. In both cases, an increase of approximately 2.25 compared to the current situation was observed. The data obtained from Table 10 shows that the current situation yielded better results. These results give clues that the number of "fake" data can be reduced or completely eliminated in future studies on TWHF.

C. TWHF SECURITY ANALYSIS

Tests were performed to analyze the security of TWHF and some one-way hash functions. In the first test, the hash code generated for the word "crypto" was accepted as the original hash code. Then, each bit of the word "crypto" which is 48 bits long, was changed starting from the MSB side and 48 different hash codes were generated. The process of changing the bits is "1" for "0" and "0" for "1". The 48 hash codes generated were compared with the original hash code at the bit level. The differences of each hash code compared to the original hash code in terms of percentage ratio have been found. These values give the reflection of a change in one bit of the original data in the generated hash codes. The graph in Fig. 37 shows the avalanche effect of hash functions.

The average difference was 50.57% for the 52-bit output of the TWHF. The values found for other hash functions were MD5 49.42%, SHA 256 49.70%, SHA 512 50.31%, Keccak 256 49.90% and Keccak 512 49.87%. In this test, TWHF performed better than the one-way hash functions.

Another test for security analysis is the permutation test for the word "crypto". For this purpose, a hash code was first generated for the word "crypto." This generated code was accepted as the original hash code. Subsequently, 719 different words have been derived by changing the letters of the word "crypto." Some of these words can be exemplified as "crypot," "crytpo," "crytop" etc. Hash code was generated for each of the 719 derived words. These hash codes were compared with the original hash code at the bit level. As a result of the comparison, the percentage of how much the hash codes of the derived words differ from the original hash code was obtained. A graph drawn according to these values is shown in Fig. 38. In addition, the average difference values were obtained as TWHF 50.07%, md5 49.77%, SHA 256 49.87%, SHA 512 49.91%, Keccak 256 49.63%, and Keccak 512 49.98%. TWHF, also performed better in this test.

The randomness of TWHF and the randomness of some one-way hash functions were measured by NIST tests. The 48 different hash codes generated for the avalanche effect test were also used in the NIST tests. The results are shown in Table 11. According to the table, all hash functions failed the Maurer's Universal Statistical test. The TWHF passed 14 of the randomness tests.

TABLE 8. Other statistics obtained during the encryption of "A" characters.

Total number of iterations	7680
Crypto raw data (16 bits)	$(1010001001000000)_2$
Function output when applying the reduction function to the crypto raw data (15 bits)	$(010010101000111)_2$
Number of "real" data	4
Number of "fake" data	12
Total number of bits of plain text loaded into the integer part of the real data	8 bit
Total bit length of the control data loaded into the decimal part of the real data	9 bit
Total number of bits loaded into real data Plaintext data bits. + control data	8+9= 17 bit
Number of fake data-I (when control data are applied)	12
Number of fake data - II (when control data are not applied)	89
Number of fake data - II / Number of fake data – I	89/16=7.41

TABLE 9. Statistics for different texts.

Initial Data : $(5,000,000.00000000)_{10}$						
Frequency Table : ["1" "00" "010" "011"]						
Plain Text	Plain Text Binary Length	Encrypted Raw Data Length	Reduced Crypto Data Length (Huffman)	Total Number of Iterations	Real Number of Data	Fake Number of Data
In this study, an original cryptography structure has been developed (Text 1)	544	1940	1446	913 264	328	1612
In this study, an original cryptography structure has been developed. Other crypto algorithms use similar structures (Text 2)	928	3325	2443	1 582 230	562	2763
In this study, an original cryptography structure has been developed. Other crypto algorithms use similar structures. These are feistel, permutation public key structures (Text 3)	1360	4904	3603	2 336 048	814	4090

TABLE 10. Impact of condition 2 and condition 3 on TWHF.

	Current Status	In case block 1.1-1.3 becomes 1.1-1.1 (Condition 2)	In case the 3rd condition is removed. (Condition 3)
Number of fake data-I (when control data are applied)	12	47	27
Number of fake data - II (when control data are not applied)	89	225	201

VIII. DISCUSSION

In this study, we investigated whether hash functions, which are one-way in the literature, can be two-way. In this context, the benefits of two-way hash functions and their ability to fulfil the security features offered by one-way hash functions are discussed. For this purpose, the TWHF methodology was developed, which involves generating an encrypted hash code from the original data and retrieving the original data from the encrypted hash code. In the process of generating an encrypted hash code, TWHF generates a pattern of decimal numbers with a mathematical relationship between them and stores the original data in the pattern in the form of small bits. In the decryption process, the last decimal number of the number pattern is taken as the hash code and the original data is retrieved by utilising the mathematical relationships between the decimals.

In this study, answers to some research questions were obtained. These questions and the answers given according to the results can be summarized as follows.

Can hash functions, which are today one-way, be two-way? This study aims to make it possible to retrieve the original data, which is impossible for one-way hash functions in terms of computation time. The encrypted hash code generation process of the proposed methodology was stable. However, during the decryption process, an average of 0.17% of "fake" data were observed according to the data in Table 9. In other words, these data show that there is uncertainty in the decoding of 0.17% of the number pattern. The uncertainty was temporarily resolved by means of additional data characterized as "crypto raw data" in the current state of the study.

What is the purpose of providing two-way properties to hash functions? Although the one-way nature of hash functions provides an advantage in terms of security for some applications, it may also have disadvantages. These disadvantages can be explained by the fact that in addition to the hash code, the original data must also be sent to the recipient, or the recipient must have the original data

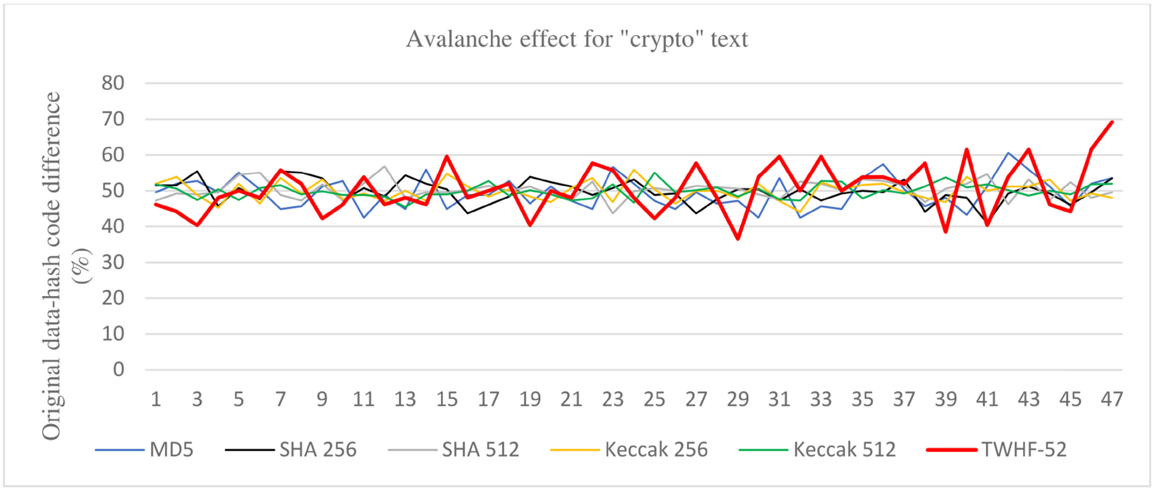

FIGURE 37. Avalanche effect for "crypto" text.

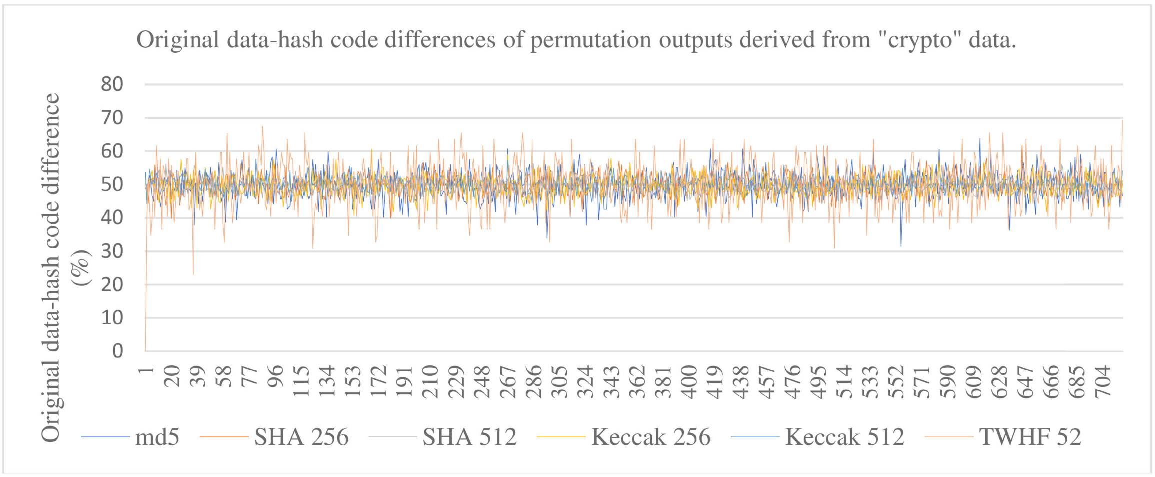

FIGURE 38. Original data-hash code differences of permutation outputs derived from "crypto" data.

in advance. However, in a two-way hash function with the security features of one-way hash functions, the original data do not need to be sent to the receiver. This defines a higher level of security.

What are the possible uses of the two-way hash functions? What are the benefits of using two-way hash functions in these areas compared with one-way hash functions? Two-way hash functions can be used in digital signature and blockchain applications where one-way hash functions are used. When used in these areas, the original data do not need to reach the recipient side and therefore offer a higher level of security. In addition, the two-way nature of hash functions allows them to be used for steganography and data storage. In steganographic applications, distortion of the cover object can be minimized. The fact that hash functions present the data in the form of a summary and that the output length does not change even if the bit length of the original data increases makes it much more effective in steganography. The benefits of keeping data summarized in storage areas and accessing the original data losslessly when needed are incredible.

Can two-way hash functions provide the security properties of one-way hash functions? Whether the 52-bit low-length output of the TWHF method has a significant relationship with the input data is tried to be revealed by the tests performed within the scope of the study. Graphs of the tests performed for this purpose are shown in Fig. 37 and Fig. 38 In the first of the tests, the results of which are visualised in Fig. 37 and show the avalanche effect of TWHF, an average difference rate of 50.57% was obtained. This result is better than the results obtained using some common one-way functions. In the second test, which is visualized in Fig. 38, the difference rate of the hash codes of 719 different texts generated by permutation for the text "crypto" compared to the hash code of the original text was found to be 50.07% on average. This average difference percentage is slightly better than the average difference

TABLE 11. NIST tests of hash codes generated by TWHF and one-way functions for "crypto" text - P Value.

Hash Functions	**MD5**		**SHA256**		**SHA 512**		**KECCAK 256**		**KECCAK 512**		**TWHF-52**	
Frequency Test (Monobit)	R	0.28	R	0.069	R	0.527	R	0.759	R	0.021	R	0.606
Frequency Test within a Block	R	0.671	R	0.626	R	0.680	R	0.798	R	0.580	R	0.518
Run Test	R	0.628	R	0.537	R	0.157	R	0.348	R	0.746	R	0.582
Longest Run of Ones in a Block	R	0.019	R	0.203	R	0.422	R	0.344	R	0.188	R	0.776
Binary Matrix Rank Test	R	0.161	R	0.118	R	0.805	R	0.107	R	0.810	R	0.481
Discrete Fourier Transform (Spectral) Test	R	0.649	R	0.235	R	0.989	R	0.100	R	0.584	R	0.956
Non-Overlapping Template Matching Test	R	0.695	R	0.950	R	0.152	R	0.121	R	0.768	R	0.557
Overlapping Template Matching Test	R	0.085	R	0.768	R	0.535	R	0.641	R	0.805	R	0.254
Maurer's Universal Statistical test	N-R	-1	N-R	-1	N-R	-1	N-R	-1	N-R	-1	N-R	-1
Linear Complexity Test	N-R	$1.048\text{x}10^{-5}$	R	0.353	R	0.534	R	0.766	R	0.430	R	0.985
Serial test:	R	0.250	R	0.557	R	0.075	R	0.699	R	0.809	R	0.758
Approximate Entropy Test	R	0.011	R	0.010	R	0.231	R	0.092	R	0.420	N-R	$2.33\text{x}10^{-7}$
Cummulative Sums (Forward) Test	R	0.163	R	0.063	R	0.871	R	0.981	R	0.017	R	0.225
Cummulative Sums (Reverse) Test	R	0.573	R	0.136	R	0.344	R	0.769	R	0.011	R	0.566
Random Excursions Test – State : +1	R	0.156	R	0.161	R	0.053	R	0.136	R	0.367	R	0.382
Random Excursions Variant Test – State : -1	R	0.317	N-R	0.004	N-R	0.010	R	0.666	R	0.779	R	0.570

percentage of one-way hash functions. The final security test of the TWHF is the NIST randomness test, the results of which are listed in Table 11. This test involves the randomness of 48 different hash codes for 48 different data obtained by replacing each bit of the 48-bit "crypto" text in turn. According to this test, all hash functions including TWHF fail Maurer's Universal Statistical test. TWHF also failed the approximate entropy test but passed the other 14 randomness tests.

The test results strongly suggest that the TWHF can provide the security properties of one-way hash functions. However, due to its two-way nature, the "key" structure should be included in the TWHF methodology. With this key, the "root hash code" should be accessed first and the process of decoding the original data should start from this point. Otherwise, starting the decryption process without this security phase may render the TWHF method unreliable. A visual representation of this situation is shown in Fig. 29.

A. OPEN QUESTIONS

It can be said that TWHF can provide a private key in its current methodology and only the owners of this key will have access to the root hash code. However, it is necessary to find a solution for the average uncertainty of 0.17% that occurs in the process after the root hash code. In the current state of the TWHF, the uncertainty rate has been considerably reduced under various conditions; however, it has not been completely eliminated. The effects of the conditions on the TWHF method are listed in Table 10. The fact that uncertainty cannot be eliminated in the current situation has temporarily revealed the existence of data characterized as "crypto raw data." If the need for these data is removed, the TWHF can reach its ideal level of operation.

The security results of TWHF are similar to the results given by existing one-way hash functions. However, in the current state of TWHF, the operating speed is low compared to one-way hash functions. This is because TWHF includes operations to hide the original data as well as summarising the original data. Therefore, researchers interested in this subject should also conduct studies to increase the TWHF speed.

IX. CONCLUSION

All hash functions in the literature are one-way. The contribution of this study to the literature is that it discusses the concept of two-way hash functions and provides a methodology for this. In this study, studies have been conducted on the areas where two-way hash functions can be used, whether they can have the security features of one-way functions and their applicability.

TWHF combines the benefits of data compression, hash functions, and cryptographic algorithms into a single methodology with the originally developed functions, concepts, and workflow. TWHF aims to express data of arbitrary length with a hash code, and to obtain the original data from the hash code when needed. With this feature, TWHF has the potential to be used in many areas such as data storage, data confidentiality, data integrity, digital signature applications, blockchain, and steganography.

The strengths of the TWHF are derived from its methodology. In this context, a novel change function, data/block classifications, and an original workflow integrating them with Huffman coding are presented. The change function was used to find the data suitable for Huffman encoding. Data were divided into four main classes that contributed to the literature. Another important contribution of this study is the classification of blocks formed by data and their integration into the workflow of the proposed method.

The analysis of TWHF and the results obtained can be summarised as follows.

When the security test results of common one-way hash functions are compared with TWHF, it can be said that the results are satisfactory.

It is thought that TWHF can be used more effectively in areas where one-way functions are used, owing to its two-way feature. In addition, the two-way nature of the TWHF method may make it possible to use in areas where one-way hash functions cannot be used.

There are no problems with the TWHF encrypted hash code generation process. During the decryption process, uncertainty was detected in some data and these data were charactered as "fake." The uncertainty was reduced by a factor of approximately 7.4 using some conditions and control data.

This study is of an unsolved type of problem and is open to further development. This paper discusses all aspects of two-way hash functions, the existence of this function and aims to provide guidance for its development. In the future, it is extremely important to improve the methodology presented in this study with the involvement of more researchers. In addition, it is foreseen that publication of the study in this form will cause researchers interested in the subject to initiate new research and thus obtain more effective results regarding the methodology.

REFERENCES

[1] B. M. P. Waseso and N. A. Setiyanto, "Web phishing classification using combined machine learning methods," *J. Comput. Theories Appl.*, vol. 1, no. 1, pp. 11–18, Aug. 2023.

[2] R. Barman, S. Deshpande, N. Kulkarni, S. Agarwal, and S. Badade, "A review on lossless data compression techniques," *Int. J. Sci. Res. Eng. Trends*, vol. 7, no. 1, pp. 143–148, Feb. 2021.

[3] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, no. 2, pp. 119–140, Feb. 2021.

[4] A. Gupta and S. Nigam, "A review on different types of lossless data compression techniques," *Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol.*, vol. 7, pp. 50–56, Jan. 2021.

[5] P. William, A. Choubey, G. S. Chhabra, R. Bhattacharya, K. Vengatesan, and S. Choubey, "Assessment of hybrid cryptographic algorithm for secure sharing of textual and pictorial content," in *Proc. Int. Conf. Electron. Renew. Syst. (ICEARS)*, Tuticorin, India, Mar. 2022, pp. 918–922.

[6] J. Verma, M. Shahrukh, M. Krishna, and R. Goel, "A critical review on cryptography and hashing algorithm SHA-512," *Int. Res. J. Modern. Eng.*, vol. 3, no. 12, pp. 1760–1764, Dec. 2021.

[7] T. Koroglu and R. Samet, "Secure steganographic data transmission method for periodically updated data," *J. Modern Tech. Eng.*, vol. 7, no. 3, pp. 153–171, Dec. 2022.

[8] Z. Alqad, M. Oraiqat, H. Almujafet, S. Al-Saleh, H. Al Husban, and S. Al-Rimawi, "A new approach for data cryptography," *Int. J. Comput. Sci. Mob. Comput.*, vol. 8, no. 9, pp. 30–48, Sep. 2019.

[9] M. N. B. Anwar, M. Hasan, M. M. Hasan, J. Z. Loren, and S. T. Hossain, "Comparative study of cryptography algorithms and its' applications," *Int. J. Comput. Netw. Commun. Sec.*, vol. 7, no. 5, pp. 96–103, May 2019.

[10] F. Maqsood, M. Ahmed, M. Mumtaz, and M. Ali, "Cryptography: A comparative analysis for modern techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 6, pp. 442–448, 2017.

[11] M. Barakat and C. E. V. T. Hanke. (2018). *An Introduction to Cryptography*. [Online]. https://agag-ederc.math.rptu.de/~ederc/download/Cryptography.pdf

[12] O. G. Abood and S. K. Guirguis, "A survey on cryptography algorithms," *Int. J. Sci. Res. Publications (IJSRP)*, vol. 8, no. 7, pp. 495–516, Jul. 2018.

[13] M. N. Alenezi, H. Alabdulrazzaq, and N. Q. Mohammad, "Symmetric encryption algorithms: Review and evaluation study," *Int. J. Commun. Netw. Inf. Sec.*, vol. 12, no. 2, pp. 256–272, Aug. 2020.

[14] M. A. Al-Shabi, "A survey on symmetric and asymmetric cryptography algorithms in information security," *Int. J. Sci. Res. Publications (IJSRP)*, vol. 9, no. 3, pp. –589, Mar. 2019.

[15] D. R. I. M. Setiadi and N. Rijati, "An image encryption scheme combining 2D cascaded logistic map and permutation-substitution operations," *Computation*, vol. 11, no. 9, p. 178, Sep. 2023, doi: 10.3390/computation11090178.

[16] E. Winarno, K. Nugroho, P. W. Adi, and D. R. I. M. Setiadi, "Combined interleaved pattern to improve confusion-diffusion image encryption based on hyperchaotic system," *IEEE Access*, vol. 11, pp. 69005–69021, 2023, doi: 10.1109/ACCESS.2023.3285481..

[17] A. Babalola, E. Bamidele, and O. Esther, "Cryptography: A review," *Int. J. Adv. Eng. Manag. (IJAEM)*, vol. 3, no. 10, pp. 1385–1391, Oct. 2021.
[18] V. S. Shetty, R. Anusha, D. Kumar, and P. Hegde, "A survey on performance analysis of block cipher algorithms," in *Proc. Int. Conf. Inventive Comput. Technol. (ICICT)*, Feb. 2020, pp. 167–174.
[19] W. Yihan and L. Yongzhen, "Improved design of DES algorithm based on symmetric encryption algorithm," in *Proc. IEEE Int. Conf. Power Electron., Comput. Appl. (ICPECA)*, Shenyang, China, Jan. 2021, pp. 220–223.
[20] B. Padmavathi and S. R. Kumari, "A survey on performance analysis of DES, AES and RSA algorithm along with LSB substitution," *IJSR, India*, vol. 2, no. 4, pp. 170–174, Apr. 2013.
[21] S. Sharma, K. N. Patel, and A. Siddhath Jha, "Cryptography using blowfish algorithm," in *Proc. 3rd Int. Conf. Adv. Comput., Commun. Control Netw. (ICACN)*, Greater Noida, India, Dec. 2021, pp. 1375–1377.
[22] B. V. Nair, "Survey on asymmetric key cryptographic algorithms," *Int. J. Sci. Res. Sci., Eng. Technol.*, vol. 2, pp. 404–408, Apr. 2020.
[23] R. Sood and H. Kaur, "A literature review on RSA, DES and AES encryption algorithms," *Emerg. Trends Eng. Manag.*, pp. 57–63, Feb. 2023, doi: 10.56155/978-81-955020-3-5-07.
[24] S. F. Yousif, "Secure voice cryptography based on Diffie–Hellman algorithm," in *Proc. 2nd Int. Sci. Conf. Eng. Sci.* Diyala, Iraq: Materials Science and Engineering, Feb. 2021, Art. no. 012057, doi: 10.1088/1757-899X/1076/1/012057.
[25] M. R. Anwar, D. Apriani, and I. R. Adianita, "Hash algorithm in verification of certificate data integrity and security," *Aptisi Trans. Technopreneurship (ATT)*, vol. 3, no. 2, pp. 65–72, Sep. 2021.
[26] B. Madhuravani and D. S. R. Murthy, "Cryptographic hash functions: SHA family," *Int. J. Innov. Tech. Expl. Eng.*, vol. 2, no. 4, pp. 326–329, Mar. 2013.
[27] A. A. Alkandari, I. F. Al-Shaikhli, and M. A. Alahmad, "Cryptographic hash function: A high level view," in *Proc. Int. Conf. Informat. Creative Multimedia*, Kuala Lumpur, Malaysia, Sep. 2013, pp. 128–134.
[28] A. Zellagui, N. Hadj-Said, and A. Ali-Pacha, "Comparative study between Merkle–Damgård and other alternative hashes construction," in *Proc. Sec. Conf. Inf. App. Math.*, Guelma, Algeria, Jun. 2019, pp. 30–34.
[29] Z. Al-Odat and S. Khan, "Constructions and attacks on hash functions," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Las Vegas, NV, USA, Dec. 2019, pp. 139–144.
[30] S. Debnath, A. Chattopadhyay, and S. Dutta, "Brief review on journey of secured hash algorithms," in *Proc. 4th Int. Conf. Opto-Electron. Appl. Opt. (Optronix)*, Kolkata, India, Nov. 2017, pp. 1–5, doi: 10.1109/OPTRONIX.2017.8349971.
[31] Z. Al-Odat, A. Abbas, and S. U. Khan, "Randomness analyses of the secure hash algorithms, SHA-1, SHA-2 and modified SHA," in *Proc. Int. Conf. Frontiers Inf. Technol. (FIT)*, Islamabad, Pakistan, Dec. 2019, p. 316.
[32] R. Martino and A. Cilardo, "A flexible framework for exploring, evaluating, and comparing SHA-2 designs," *IEEE Access*, vol. 7, pp. 72443–72456, 2019, doi: 10.1109/ACCESS.2019.2920089.
[33] A. K. Sharma and S. K. Mittal, "Cryptography & network security hash function applications, attacks and advances: A review," in *Proc. 3rd Int. Conf. Inventive Syst. Control (ICISC)*, Coimbatore, India, Jan. 2019, pp. 177–188.
[34] K. Rajeshwaran and K. Anil Kumar, "Cellular automata based hashing algorithm (CABHA) for strong cryptographic hash function," in *Proc. IEEE Int. Conf. Electr., Comput. Commun. Technol. (ICECCT)*, Coimbatore, India, Feb. 2019, pp. 1–6, doi: 10.1109/ICECCT.2019.8869146.
[35] N. Kheshaifaty and A. Gutub, "Preventing multiple accessing attacks via efficient integration of captcha crypto hash functions," *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)*, vol. 20, no. 9, pp. 16–28, Sep. 2020.
[36] Y. Li and G. Ge, "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security," *Multimedia Tools Appl.*, vol. 78, no. 13, pp. 17973–17994, Jan. 2019.
[37] A. Mohammed Ali and A. Kadhim Farhan, "A novel improvement with an effective expansion to enhance the MD5 hash function for verification of a secure E-document," *IEEE Access*, vol. 8, pp. 80290–80304, 2020, doi: 10.1109/ACCESS.2020.2989050.
[38] A. Abouchouar, F. Omary, and K. Achkoun, "New concept for cryptographic construction design based on noniterative behavior," *IAES Int. J. Artif. Intell. (IJ-AI)*, vol. 9, no. 2, pp. 229–235, Jun. 2020.
[39] H. Abroshan, "A hybrid encryption solution to improve cloud computing security using symmetric and asymmetric cryptography algorithms," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, pp. 31–37, 2021.
[40] X. Wang, S. Wang, N. Wei, and Y. Zhang, "A novel chaotic image encryption scheme based on hash function and cyclic shift," *IETE Tech. Rev.*, vol. 36, no. 1, pp. 39–48, Jan. 2019.
[41] M. Alotaibi, D. Al-Hendi, B. Alroithy, M. AlGhamdi, and A. Gutub, "Secure mobile computing authentication utilizing hash, cryptography and steganography combination," *J. Inf. Secur. Cybercrimes Res.*, vol. 2, no. 1, pp. 73–82, 2019.
[42] K. Koptyra and M. R. Ogiela, "Imagechain—Application of blockchain technology for images," *Sensors*, vol. 21, no. 1, p. 82, Dec. 2020, doi: 10.3390/s21010082.
[43] X. Liu, P. An, Y. Chen, and X. Huang, "An improved lossless image compression algorithm based on Huffman coding," *Multimedia Tools Appl.*, vol. 81, no. 4, pp. 4781–4795, Jun. 2021.
[44] M. A. Rahman and M. Hamada, "Lossless image compression techniques: A state-of-the-art survey," *Symmetry*, vol. 11, no. 10, p. 1274, Oct. 2019, doi: 10.3390/sym11101274.
[45] R. Arshad, A. Saleem, and D. Khan, "Performance comparison of Huffman coding and double Huffman coding," in *Proc. 6th Int. Conf. Innov. Comput. Technol. (INTECH)*, Dublin, Ireland, Aug. 2016, pp. 361–364.
[46] L. J. Paterson and C. A. Glasbey, "An illustration of rounding error on computers," *Math. Gazette*, vol. 69, no. 448, pp. 128–131, Jun. 1985.
[47] S. Singh, M. Sarfaraz Iqbal, and A. Jaiswal, "Survey on techniques developed using digital signature: Public key cryptography," *Int. J. Comput. Appl.*, vol. 117, no. 16, pp. 1–4, May 2015.
[48] A. S. Rajasekaran, M. Azees, and F. Al-Turjman, "A comprehensive survey on blockchain technology," *Sustain. Energy Technol. Assessments*, vol. 52, Aug. 2022, Art. no. 102039, doi: 10.1016/j.seta.2022.102039.

TIMUCIN KOROGLU received the B.Sc. degree from the Department of Computer Systems Education, Gazi University, and the M.Sc. degree from the Department of Electrical and Electronics Engineering, Pamukkale University, in 1995. He is currently pursuing the Ph.D. degree with the Department of Computer Engineering, Ankara University. He is a Lecturer with the Department of Computer Programming, Pamukkale University. His current research interests include cyber security and web applications.

REFIK SAMET received the Ph.D. degree from the Department of Fault-Tolerant Multicomputers and Multiprocessor Systems, the Russian Academy of Sciences, and the Institute of Control Sciences. He is currently a Professor with the Department of Computer Engineering, Ankara University. He has published several papers in international journals and conferences. His research interests include parallel systems, cyber security, computer forensics, machine-learning networks, and mobile applications.