Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Review paper

Brand New Categories of Cryptographic Hash Functions: A Survey

B. Sefid-Dashti[1], J. Salimi Sartakhti[1,*], H. Daghigh[2]

[1]*Department of Computer Engineering, University of Kashan, Kashan, Iran.*

[2]*Faculty of Mathematical Science, University of Kashan, Kashan, Iran.*

Article Info

Article History:
Received 28 November 2022
Reviewed 28 December 2022
Revised 13 January 2023
Accepted 23 March 2023

Keywords:
Optical hash function
Memory-hard function
Bandwidth-hard function
Physical unclonable function
Quantum hash function
Application-specific hash function

*Corresponding Author's Email Address: *salimi@kashanu.ac.ir*

Abstract

Background and Objectives: Cryptographic hash functions are the linchpins of mobile services, blockchains, and many other technologies. Designing cryptographic hash functions has been approached by research communities from the physics, mathematics, computer science, and electrical engineering fields. The emergence of new hash functions, new hash constructions, and new requirements for application-specific hash functions, such as the ones of mobile services, have encouraged us to make a comparison of different hash functions and propose a new classification.

Methods: Over 100 papers were surveyed and reviewed in detail. The research conducted in this paper has included four sections; article selection, detailed review of selected articles, data collection, and evaluation of results. Data were collected as new hash function properties, new hash function constructions, new hash function categories, and existing hash function attacks which are used to evaluate the results.

Results: This paper surveys seven categories of hash functions including block cipher-based functions, algebraic-based functions, custom-designed functions, Memory-hard Functions (MHFs), Physical Unclonable Functions (PUFs), quantum hash functions and optical hash functions. To the best of our knowledge, the last four mentioned categories have not been sufficiently addressed in most existing surveys. Furthermore, this paper overviews hash-related adversaries and six hash construction variants. In addition, we employed the mentioned adversaries as evaluation criteria to illustrate how different categories of hash functions withstand the mentioned adversaries. Finally, the surveyed hash function categories were evaluated against mobile service requirements.

Conclusion: In addition to new classification, our findings suggest using PUFs with polynomial-time error correction or possibly bitwise equivalents of algebraic structures that belongs to post-quantum cryptography as candidates to assist mobile service interaction requirements.

Introduction

A cryptographic hash function is an integral part of a variety of applications such as digital signatures [1], authentication by static passwords, authentication by One-Time Passwords (OTP) [2], [3], data integrity [4], holographic encryption [5], Elliptic Curve Integrated Encryption Scheme (ECIES) [6], Merkle tree [7], WS-Security [8], [9], data anonymization [10], Blockchain [11]-[13], cryptocurrencies [14], [15], video similarity search [16], and hash chain based strong password authentication [17], to name a few. Nine uses of cryptographic hash functions have been reviewed by

Alkandari et al. [18] in detail.

Hash functions were introduced in the late 1970s [19]. Ever since, hash function has received great interest, which has led to the construction of a wide variety of hash functions as well as attacks that attempt to invert or forge hash values. For example, the five-year NIST SHA-3 competition which culminated in the selection of a hardware-effective algorithm, *Keccak* [20], [21], as the winner in 2012 demonstrated numerous hash functions and a security analysis of them. The Password Hashing Competition that ran from 2013 to 2015 culminated in the selection of an MHF algorithm, *Argon2* [22], as a winner. To address constrained environments such as the continuously growing Internet of Things (IoT), NIST Lightweight competition was initiated in 2018 and is due to end in the year 2022. In March 2021, 10 lightweight cypher submissions were selected for the final round of the competition, and 3 out of the 10 finalists provide lightweight hashing functionalities. *SM3* hash function [23] was introduced as a new Chinese standard in 2010. *Streebog* hash function [24] was introduced as a new Russian standard in 2013. In 2015, *Kupyna* hash function [25] was introduced as a new standard in Ukraine.

Importantly, designing variants of hash functions has been undertaken by research communities from the fields of physics, mathematics, computer science, and electrical engineering, and this has led to the introduction of new categories of hash functions. Moreover, attacks based on physics, mathematics, computer science, and electrical engineering have been developed to compromise the security of a wide variety of hash dependent applications.

This research surveys seven categories of hash functions, namely block cipher-based functions, algebraic-based functions, custom-designed hash functions, MHFs, PUFs, quantum hash functions, and optical hash functions. To the best of our knowledge, the last four mentioned categories have not been sufficiently addressed in most existing surveys [18], [19], [26]-[29].

On the other hand, with the proliferation of smartphones and tablets, mobile devices are introduced as a new computational platform for enterprise applications and other software systems, and are considered as major participants in IoT and related constrained environments (e.g. smart homes, smart cities). Although mobile devices are strong enough to consume and/or provide some services, they suffer from computational and communicational constraints on their resources and generally experience intermittent connectivity. A clear understanding of what exactly is needed from an application-specific hash function is an urgent requirement. Hence, we evaluate the surveyed categories against mobile software requirements. The remainder of this paper is structured as follows:

I. Due to significant developments in the literature and the use of cryptographic hash functions, today, new cryptographic hash functions impose more properties than traditional cryptographic hash functions. These supplementary properties are discussed in Section II.

II. SHA-3 competition (2007-2012) and Password Hashing Competition (2013-2015) fostered the design and analysis of processor-centric and memory-centric cryptographic hash functions, respectively. In turn, such events led to the introduction of new iterative and noniterative hash function constructions, six of which are overviewed in Section III. This section also reviews two hash function combiners.

III. New categories of hash functions including PUFs, quantum hash functions, and optical hash functions, MHFs along with the Bandwidth-hard Functions (BHF) subcategory, and some attacks affecting each category are presented in Sections IV and V. Section IV briefs on what affecting attacks entail. Investigated hash functions and the proposed seven-category classification are presented in the Section V. The attacks presented in Section IV are used as evaluation criteria in Section V.

IV. Mobile services suffer from computational and communicational problems. Hence, lightweight but not less secure cryptographic hash functions which secure interactions of resource constrained devices is an urgent need. Requirements which influence mobile services to choose some variants of cryptographic hash functions are presented in Section VI. In addition, Section VI discusses how each hash function category fits the mobile service requirements and why this research suggests PUFs with some enhancements and possibly bitwise equivalents of algebraic structures for mobile service consumption.

V. Section VII discusses the selection of appropriate hash functions for four application scenarios.

VI. Finally, in Section VIII the paper is summed up and conclusions are provided.

Definition

A hash function maps an input message of arbitrary length to a fixed length output which is called “hash,” “hash value,” or “message digest.” A hash function with *n*-bit output length is called an *n*-bit hash function. A good hash function produces random and uniform outputs. An output sequence resulted from applying a hash function in succession is called a “has chain” [2].

In addition to message, some hash functions may either accept a salt or a secret key. The former is called a salted hash function. Salts are randomly generated for each input message and are used for password hashing. The latter is usually used to build message authentication codes (MACs) and is called a keyed hash function. In other words, it serves as a checksum. In contrast to salts, keys are secrets and are not supposed to vary for different

messages.

Depending on the application, a hash function *h* may need to support some or all of the following properties:

I. It maps arbitrary length input *x* to *h(x)* efficiently. An efficient implementation may be achieved in software or hardware or both.

II. One-way property or pre-image resistant property: For any given y in the image of h, it is not computationally feasible to find a message x such that $y = h(x)$.

III. Second pre-image resistant property: For any given message x, it is not computationally feasible to find a message x' such that $x \neq x'$ and $h(x) = h(x')$.

IV. Collision resistant property: It is not computationally feasible to find a pair x and x' such that $x \neq x'$ and $h(x) = h(x')$.

V. Second collision resistant property: An attacker should not be able to use a given collision $h(x_1) = h(x_1')$ to find another collision $h(x_2) = h(x_2')$.

VI. Hiding property: Given $h(r||x)$ so that r is chosen from a high min-entropy probability distribution and $||$ denotes concatenation of values, it is not computationally feasible to find x [30]. This property is a variant of one-way property and originates from blockchain terminology.

VII. Puzzle friendliness property: Given r and $h(r||x)$ so that r comes from a spread-out set and h is an n-bit hash function, it is computationally infeasible to find x in time significantly less than 2^n [30]. Bitcoin mining is a race to solve such a computational puzzle.

VIII. Chosen-Target-Forced-Prefix (CTFP) preimage resistance property: Committing a hash value h, without knowing the prefix of the message that will be hashed should be difficult [31], [32].

IX. Chosen-Target-Forced-Midfix (CTFM) preimage resistance property: Committing a hash value h, without knowing any part of the message that will be hashed should be difficult [33].

X. Application-Specific Integrated Circuit (ASIC) resistance property: It should not be easy to compute on ASIC machines [34].

XI. Robustness property, aka robust video hashing property: For a given pair x and x' such that $x \neq x'$, $h(x) = h(x')$ as long as x and x' represent the same video content s, even though they represent it in different manners. In plain English, a hash function h used for video hashing should be robust against content-preserving changes such as encoding and blurring [16], [35].

The first four properties are mentioned in many references, but the rest are more or less new. *Property I* emphasizes that a hash function may be used by resource-constrained devices or to provide a fingerprint for a possibly very large file. An example of this property is a parameter provided by SHA-3 hash function to trade-off security and performance [20], [21]. As another example, some hash functions such as MD-6 provide parallel implementation to speed up hashing a long message on multicore processors [36].

A hash function that supports *Properties I* and *II* is called a *one-way hash function* [1], [37]. A *cryptographic hash function* is a one-way hash function that provides second pre-image and collision resistant properties.

Since the introduction of cryptographic hash functions in the late 1970s, lots of hash functions have emerged that support pre-image resistance and second pre-image resistance properties; providing collision resistance, however, is more challenging. Fortunately, while few of hash function applications, such as digital signature, rely on collision resistance, for others providing pre-image resistance and second pre-image resistance properties is sufficient [2], [19]. Incidentally, there are collision-free hash functions as well, such as the lattice-based hash function proposed by Goldreich et al. [38].

Regarding the special ways that hash functions are employed in blockchain, hiding and puzzle friendliness properties are defined. *Properties VI* and *VII* harden bitcoin mining by reducing its surface of vulnerability, but as bitcoin lacks *Property X*, there are ASIC machines which speed up mining with reduced cost per bitcoin mined. *Properties VIII* and *IX* are preventive criteria to resist against herding attack (Section IV-A-4). Finally, *Property XI* focuses video hashing design on semantic content changes [16], [35] extracted from segmented video structural elements such as video shots [39]. Illustrated with UML class diagram, Fig. 1 depicts how hash functions, one-way hash functions, and cryptographic hash functions are subsequently extended (denoted by UML Generalization relationship) by adding pre-image property and both second pre-image and collision resistant properties, respectively. Fig. 1 further shows how application-specific hash functions, such as blockchain specific and video hash functions, enrich the required properties to satisfy application-specific requirements.

One may wonder whether a practical hash function without one-way property exists. Murmur hash [40] is an example of a hash function which is not designed for one-wayness. Non-cryptographic hash functions (NCHFs) [41] provide fast lookup capability. This paper concentrates on cryptographic hash functions, referred to hereafter as hash functions.

Constructions and Combiners

Designing a hash function entails making important decision on how to mix input message bits all together. While a large number of hash functions exists, they all have been designed based on a handful of constructions.

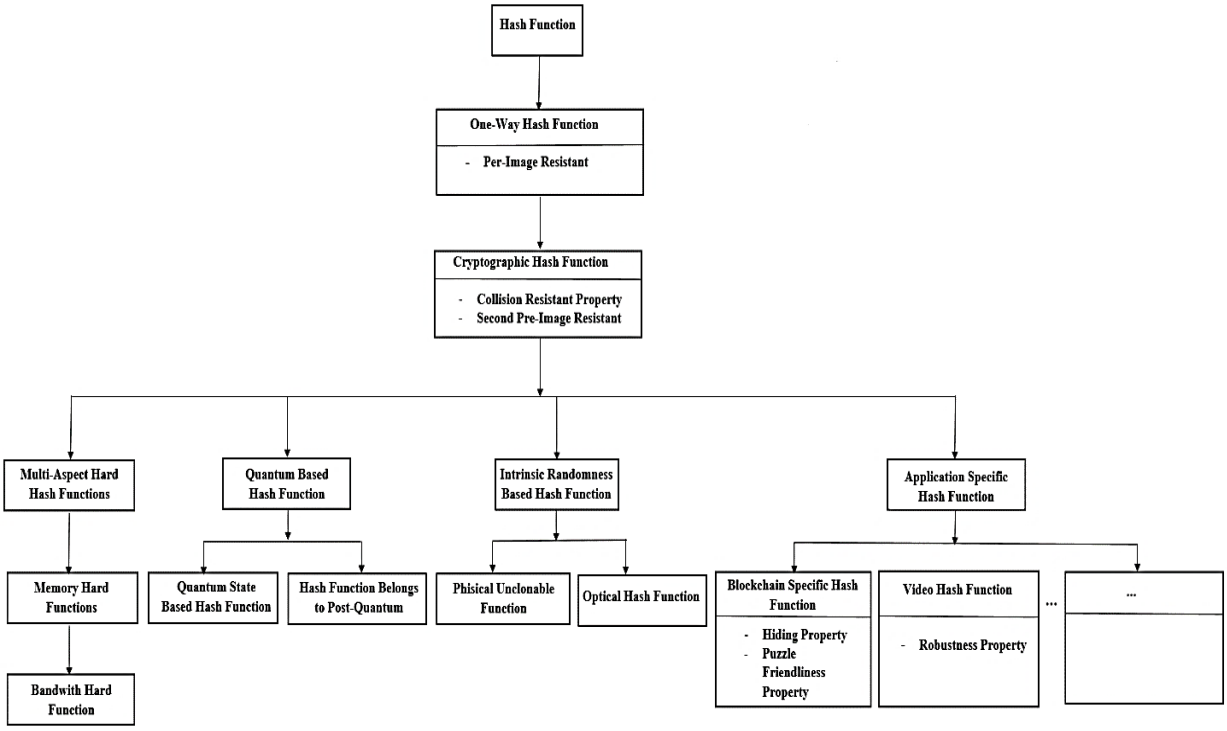

Fig. 1: Hierarhy of hash functions.

Hash function constructions are important in combining all bits of arbitrary-length messages in a way that holds properties such as collision resistance. These constructions split an arbitrary-length message into equal-sized blocks and iterate through the blocks to combine block bits all together. Some constructions combine block bits themselves, while others apply a compression function on each block and combine the results.

A compression function is a one-way function which takes a fixed length block of message along with a chaining variable as input, mixes the bits of input with each other, and returns a shorter, fixed-length output.

The way that a hash function construction combines the results of its underlying compression function is called *domain extension* [42]. For some *domain extensions*, if the underling compression function has a security property such as collision resistance, that *domain extension* can produce hash functions that retain that property. For example, it is proven that hash functions based on the Merkle-Damgård construction (Section III-B-1) which use a fixed initial value along with an appropriate padding are collision resistant as long as their corresponding compression function is collision resistant.

Moreover, some other *domain extensions* such as the Zipper hash construction [43] (Section III-B-3) produce hash functions which hold properties such as collision resistance regardless of their underlying compression function.

This section reviews four iterative and two noniterative hash function constructions. The former includes most common constructions such as Merkle-Damgård and Sponge, while the latter includes tree style and graph-based hash function constructions. Other rarely used hash function constructions, such as Wide-pipe and the Hash Iterated Framework, are not included in this survey. Both of the omitted constructions aim to solve internal collision problems. The former uses output transformation, while the latter uses a salt and a counter to achieve this goal [44].

Finally, this section reviews two *hash function combiners* (simply *combiner* henceforth). A *combiner* combines the output of two hash functions or the output of the compression functions of two hash functions [45]. As an example, bitcoin uses double SHA-256 (i.e. SHA-256(SHA-256(*message*))) and a combination of RIPEMD-160 and SHA-256 (i.e. RIPEMD-160(SHA-256(*message*))) that are examples of combining hash functions in a sequential order. As another example, a combination of MD-5 and SHA-1 was used by SSL/TLS [46]. Concatenation combiners and XOR combiners are also used [45]. *Merkle tree* and *Zipper hash* combiners are reviewed in this section. The former combines the outputs of a hash function in tree style, while the latter combines the outputs of two different compression functions in reverse order.

A. Noniterative Constructions

This section first reviews Merkle tree and then

discusses tree- and graph-based constructions. These constructions map arbitrary-length input to tree leaves or graph walks and process the resulting tree or graph.

1. Merkle Tree: Merkle tree [7] is a combiner and uses a binary tree structure to allow the integrity of large data sets to be verified quickly. One of its recent applications is bitcoin. Fig. 2 depicts an example of a Merkle tree [14]. The tree's leaves are data blocks we want to hash. The hash of each leaf node is stored in its immediate parent node. Then, the hash of each pair of nodes is concatenated and hashed together, until there is one root hash known as the Merkle root [14]. Data integrity of a block is verified by checking hashes from that block to the root node (Fig. 3 [14], [30]). A tree consisting of n nodes requires verifying about $\log n$ items [30], including verifying hash of that data block and its sibling-node (if it exists), and then proceeds upward until it reaches the top.

2. *MD-6 Tree style construction:* MD-6 [36] uses a 4-ary tree structure to achieve parallelism along with alternative sequential mode. As a source of parallelism, each round of its compression function uses 16 parallelizable loops. Moreover, it parallelizes a quaternary Merkle tree-like structure with a height adjustment parameter (L). Regarding L, there are three modes of operation:

- L = 64 as the default and means fully tree-based mode.
- L = 0 means sequential mode and uses a Merkle-Damgård construction.
- Specifying a number greater than 0 and less than 64 means hybrid mode. First using L level tree, and then sequential mode.

Fig. 4 shows an example of an MD-6 tree [36]. MD-6 uses a 4-to-1 compression function at each internal node of the tree. Tree leaves store blocks of data to be hashed, and internal nodes store the results of applying compression function on the concatenated data of four child nodes. The compression function at the root node is flagged to return truncated result as a MD-6 hash value.

MD-6 was submitted to the SHA-3 competition, but due to an error found in its security proof against differential attacks [19], it did not proceed to the second round of that competition.

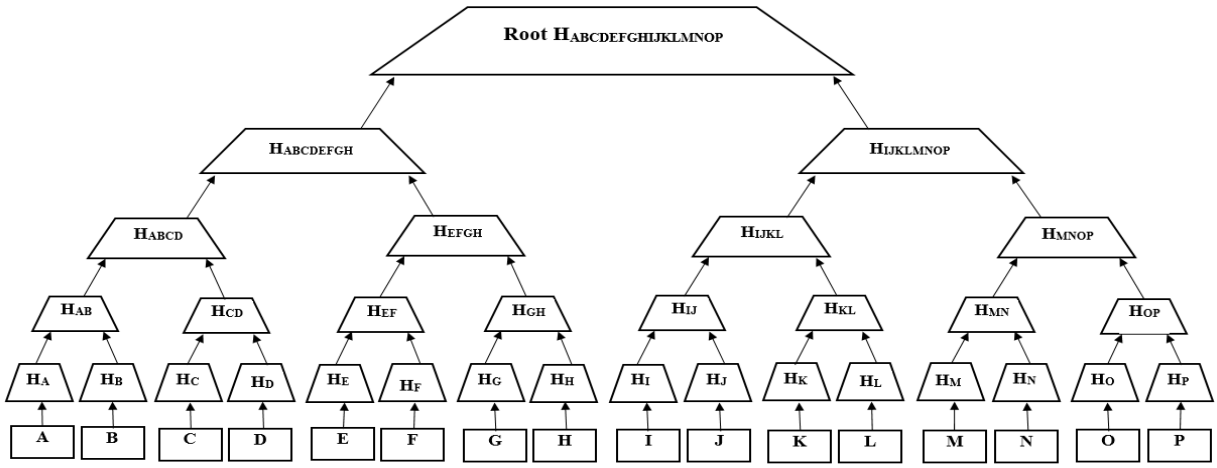

Fig. 2: An example of a Merkle tree construction [14].

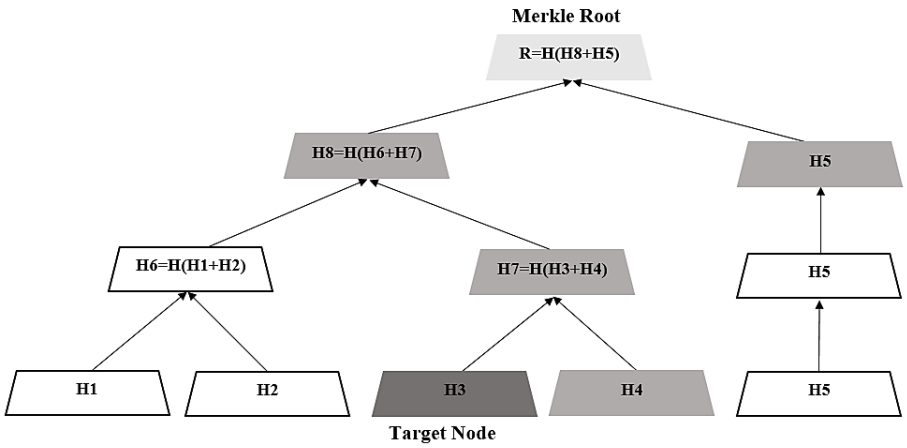

Fig. 3: Verifying hashes from a block to the root node [14], [30].

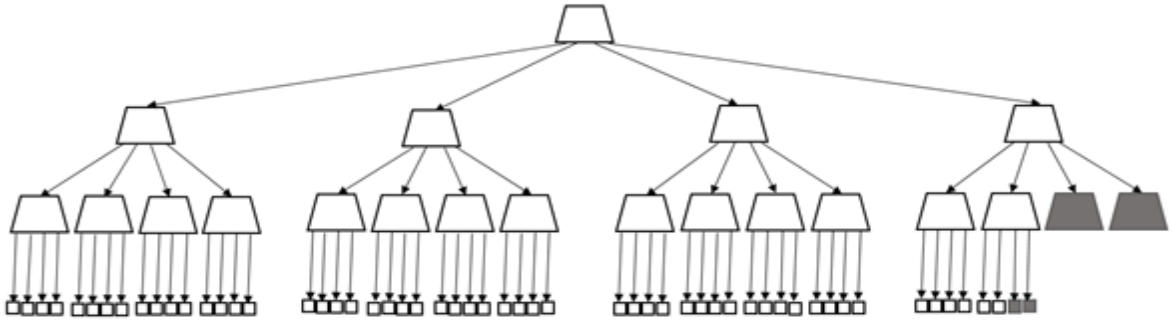

Fig. 4: An example of an MD-6 tree construction [36].

3. Graph based constructions: There are a number of hash functions defined based on Cayley graphs which are expanders too. A Cayley graph is one that encodes a group based on its generator set. An expander graph is a sparse but highly connected graph, so that each small set of vertices has many neighbors. Cayley graphs which map non-Abelian finite groups and are expanders were used to design hash functions. An example is the elliptic curves-based graph hash function defined by Charles et al. [47]. Regarding the hardness of finding cycles in an expander graph, this graph hash function used the input message to walk around an expander graph and defined collision-resistancy as equivalent to finding a cycle in such a highly connected graph.

In addition, the preimage resistance of some graph hash functions depends on the hardness of the Factorization problem in non-Abelian groups [48].

B. Iterative Constructions

These constructions iterate through an arbitrary-length input to compute bitwise operations such as XOR on fixed-length blocks of that input. Each iteration mixes an input block with either an initial value or the output of its previous iteration. The input message will be padded if its length is not an integer multiple of the block size. Hash functions based on such constructions are known as iterated cryptographic hash functions [49].

1. Merkle-Damgård construction: Merkle-Damgård construction [50], [51] was used by known hash functions MD-5, SHA-1, and SHA-2. It allows the construction of collision-resistant hash functions from collision-resistant compression functions when fixed initial values are used and the length of the input message is appended to it [19]. The same, however, is not true about pre-image resistance and second pre-image resistance properties [52]. Fig. 5 represents this construction [28]; M_i labelled boxes represent message blocks, *F* labelled trapezoids represent compression functions, solid lines represent dataflows, and other symbols intuitively represent initial value and output digest. This notation is common in cryptography literature with some exceptions that are considered irrelevant.

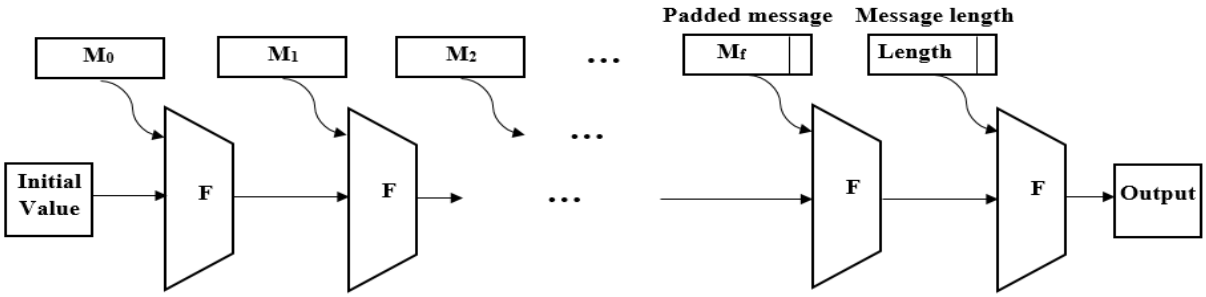

Fig. 5: Merkle-Damgård construction [28].

2. Shoup construction: The Shoup construction aims to achieve pre-image resistance and is depicted in Fig. 6 [53], [54]. It is similar to the Merkle-Damgård construction along with some mask bits that are XORed with the results of the compression function at each iteration [53], [54]. Bitwise XOR operations are represented by the $\oplus$ symbol.

3) Zipper hash construction: Zipper hash combines the results of two different compression functions in reverse order. Hence, it is a hash function combiner and a hash function construction as well. Regarding the second collision-resistant property, this construction aims to prevent the use of a successful attack on a compression function to attack a hash function which applies it. It

employs two compression functions, f_0 and f_1, which process input blocks in the reverse order [43].

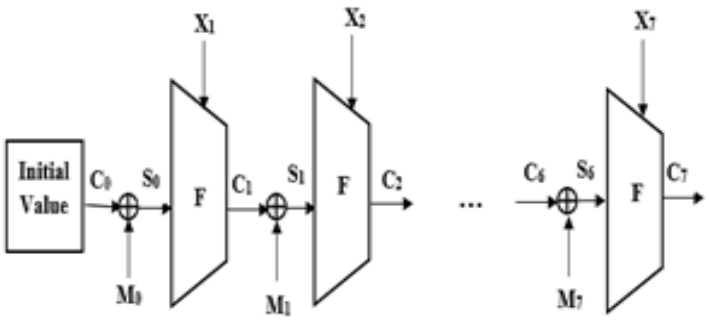

Fig. 6: Shoup construction [53], [54].

It is depicted in Fig. 7 [43] and includes the follow steps:

Step 1: Pad input message M so that the length of padded message $P(M)$ is a multiple of block size (i.e. input size of compression functions f_0 and f_1). Say blocks $M_1, M_2, \dots, M_t$.

Step 2: Compute h_1 as $f_0(M_1, Initial\ Value)$, and $h_2, \dots, h_t$ as $h_i = f_0(M_i, h_{i-1})$.

Step 3: Compute h'_1 as $f_1(M_t, h_t)$, and $h'_2, \dots, h'_t$ as $h'_i = f_1(M_{t-i+1}, h'_{i-1})$.

Step 4: Compute output transformation function *g*(h'_t) as hash value of input message.

The output transformation function is represented by a *g* labelled trapezoid.

A second pre-image attack on *Zipper hash* was introduced [42], although the time complexity of this attack was not much better than the time complexity of the brute force attack (i.e. O (2^n)). In addition, *Herding attack* (Section IV.A.4) was extended to attack the *Zipper hash* and other hash function constructions which process each message block more than once [55].

4: Sponge construction: The Sponge construction [20], [21] is used by the Keccak hash function which won the SHA-3 competition. This construction takes the padding algorithm as input and adds zero initiated bits which are called capacity (*c*) to the processing bits of each iteration which are called bit-rate (*r*). The ratio of capacity bits to bit-rate determines the balance between security and performance [21].

Fig. 8(a) shows how an input message is padded and processed by appending capacity bits to each block in each iteration [21]. Accomplishing such iteration through all blocks is called the *absorbing phase* which processes *b= r + c* bits at each iteration. In addition, the Sponge construction allows users to customize the output size. If the length of required output (*l*) is not greater than *b*, then the first *l* bit of *b* is returned as output; if *l>b,* however, then the *squeezing phase* begins, so that the first *r* bit of the output of all *squeezing* iterations are concatenated and returned as output. Fig. 8(b) shows the squeezing phase [21].

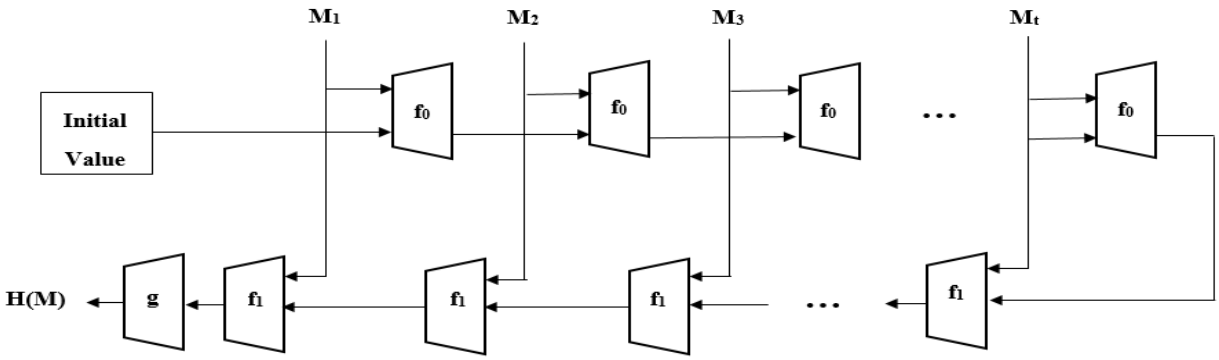

Fig. 7: Zipper hash construction [43].

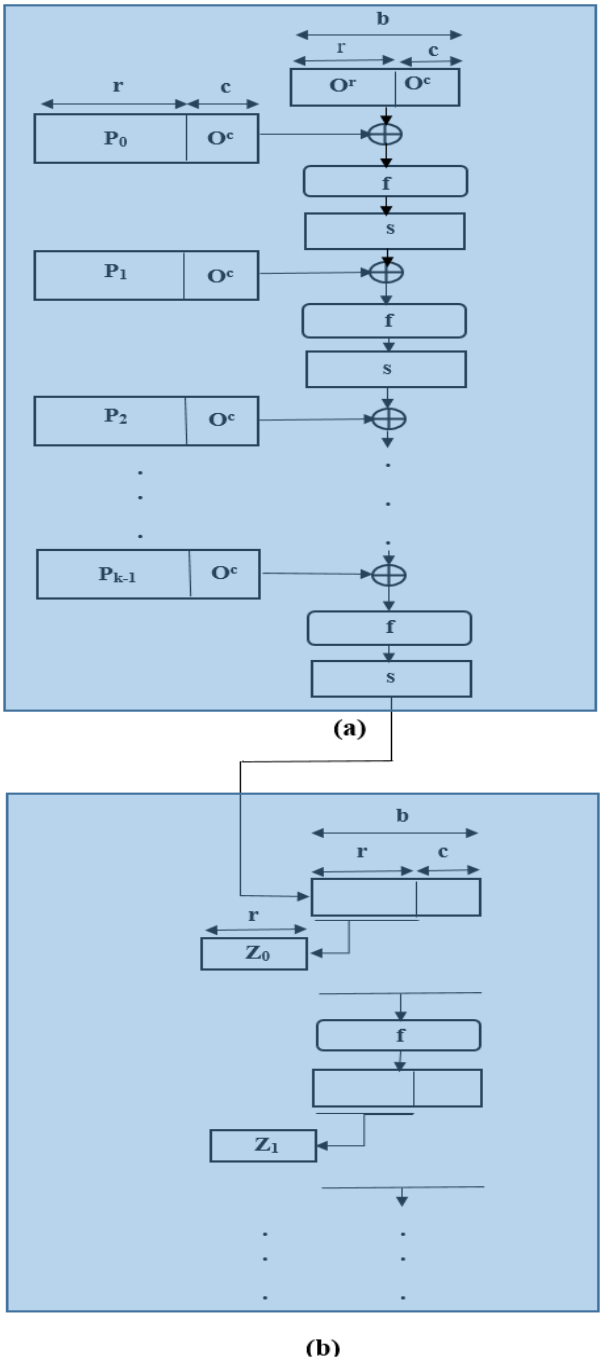

Fig. 8: Sponge construction, (a) input padding (b) squeezing output [21].

Attacks and Adversaries

How a hash function resists different attacks is the most important criterion for gaining wide acceptance. Loosely speaking, there are four categories of approaches to make an attack on a hash function: generic attacks, cryptanalysis attacks, quantum adversaries, and implementation specific adversaries. This section briefly describes these categories, and Table 1 depicts the target, method, and complexity of each attack category. The parameter *n* used in the last column of Table 1 denotes the length of input message which will be hashed.

A. Generic Attacks

Generic attacks are slow, but they apply to all hash functions, regardless of their algorithms and corresponding implementations. Thus, these attacks define a lower band for the output length of secure hash functions [56]. These attacks call a hash function or its compression function a number of times and seek relationships between the results. As a generic attack uses the black box model, it may cause exponential time complexity in the form of $\theta(2^{(n-k)/a})$, where *n* is output length of hash function, *a* indicates the possible order reduction by statistical methods (e.g., birthday attack and herding attack), and *k* is the order reduction achieved in the cost of $\theta(2^k)$ space (herding attack and rainbow tables). See Table 2.

1: Brute force attacks: A brute force attack on an *n*-bit hash function evaluates that function on $\theta(2^n)$ distinct input values to find (second) pre-images; considering multiple targets, say $\theta(2^t)$ targets, the cost can be reduced to $\theta(2^{n-t})$, while this degradation can be answered by parameterization of the hash function [19]. Furthermore, in some cases such as password hashing, rainbow tables, which are cached tables of precomputed hash values, may accelerate these attacks and trade increased space usage with decreased time. But random salting [57] and automatic padding [58] prevent such lookup table creations.

A brute force attack shows the worst case to find a pre-image or second pre-image on an *n*-bit hash function. It determines a lower bound for the output length of hash function to resist pre-image and second pre-image attacks (Similarly, birthday attack defines a lower bound for the output length of hash function to resist against collision attacks). For example, 224 bits is the lower bound used by SHA-2 and SHA-3 hash functions.

2: Birthday attacks: These algorithms find a collision based on the so-called birthday paradox in the cost of $\theta(2^{n/2})$ with a probability greater than ½.

Seemingly unintuitive, the birthday paradox states that 23 people are sufficient to have a shared birthday occurrence with ½ probability, i.e. the probability of finding a shared birthday (i.e. collision) for *t* people whose birthdays are independently distributed among the n = 365 days of a non-leap year is $\theta(t^2/n)$ if $t < n^{1/2}$ and is a constant value otherwise [56]; the exact value is computed by the possibility that each investigated person does not share their birthday with previously investigated persons and subtracting that product value from 1 [1]; this probability is denoted in (1).

$$p = 1 - \prod_{i=0}^{t-1}(365 - i)/365. \quad (1)$$

This attack stores $O(2^{n/2})$ values, and it may be possible to trade off required time against memory as described by Katz and Lindell [56].

Table 1: Categories of Attacks on Hash Functions; Targets and Methods. *n, a, k, b, c, d and e* present output length of hash function, possible order reduction by statistical methods, order reduction achieved in the cost of $\theta(2^k)$ space, polynomial time constant value, polynomial time constant value, polynomial time constant value, sub-exponential time constant value, and , polynomial time constant value respectively

Category	Target		Method Elements	Time Complexity
Generic attacks	The output of hash function (hash value) or the output of compression function		Statistical methods and probability theory	$\theta(2^{(n-k)/a})$; where k and a are constant values
Cryptanalysis	Steps of algorithm		Detecting no random behavior in parts of a hash algorithm	From $\theta(n^b)$ to $\theta(2^n)$, where b is a constant value
Quantum adversaries	Steps of algorithm		Quantum solution for classically non-polynomial steps of algorithm, such as Integer Factorization and Discrete Logarithm.	From $\theta(n^c)$ to $\theta(2^{n/d})$, where c is a constant value
Implementation specific attacks	Physical security attacks	Dependency of Time and power consumption to executed operations and processed data. Electromagnetic fields which are emitted by processors.	Time measurements to verify the correlation between a partial key value and the expected running time, power traces, and also measuring near- and far- field of processors	$\theta(n^e)$, where e is a constant value – few declared time complexities.
	Software implementation attacks	Steps of algorithm implemented in a vulnerable programming language or in a vulnerable manner.	For example, buffer overflow for algorithms implemented in C language without boundary checking	

To counter this attack, one may use Universal One-Way Hash Functions (UOWHF), which are a class of hash functions that are indexed by a parameter (key) and select function instance based on selected challenge input [2].

3: Meet-in-the-middle attacks: These attacks apply to multiple encryption schemes such as double DES and find matches between encrypted values of one scheme and decrypted values of another scheme [59]. Derived from encryption, these attacks were applied for finding pre-images of reduced variants of common hash functions such as MD4 [60], [61], MD-5 [61], SHA-1 [62], [63] and SHA-2 [60], [64]. For example, Aoki et al. [64] divided the steps of the compression function and used a pre-image of the compression function to gain a pre-image of the hash function. As another example, Knellwolf and Khovratovich [62] employed the meet-in-the-middle technique along with differential cryptanalysis (differential cryptanalysis is discussed in section IV-B-3) to attack SHA-1.

4: Herding attack: A herding attack, aka the Nostradamus attack, finds (second) pre-images on a hash function by searching collisions among precomputed compression functions.

It uses the birthday paradox to find the mentioned collisions and constitutes a diamond-shaped network of these collisions to determine a hash value that can be declared as a commitment to some predictions about the future.

At a point in the future, a second pre-image of that value which includes some happened events will be published as evidence to support that assertion [65], [66]. This attack finds a suffix that can be appended to a message, so that the concatenated message results in a hash value which is equal to the hash value claimed by attacker.

Mennink [32] improved the flexibility of the attack by adjusting trade-off between the speed of attack and the length of the (second) pre-image.

The herding attack was designed to target hash functions based on the Merkle–Damgard construction. Moreover, Andreeva et al. [55] showed the success of herding attacks on four other hash function constructions, namely concatenated, zipper, hash-twice, and tree hash constructions. constructions.

B. Cryptanalysis

Cryptanalysis exploits logical weaknesses in a hash algorithm to invert or forge hash values [67].

These attacks are generally more efficient than generic attacks, but their applicability is limited to either a specific hash function or a specific implementation of a hash function.

This section overviews four attacks in this category: length extension attack, algebraic cryptanalysis, differential cryptanalysis, and rebound attack.

The first exploits the lack of output transformations, and the second breaks codes by solving equivalent equations. The others detect primitives that hold properties leading to a non-random behavior through a number of rounds. Some cryptanalysis attacks operate in polynomial time (e.g., length extension attack and differential cryptanalysis), while others operate in exponential time and space complexity (e.g., rebound attack). See Table 2.

1: Length Extension Attack: Some hash function constructions, such as the known Merkle–Damgård one, process subsequent blocks, mix the results subsequently, and provide the internal state of the processed blocks as a hash value.

Exposure of the internal state makes the hash function vulnerable to length extension attack.

Message authentication is an example of an application which is susceptible to length extension attack. Applications may authenticate messages by prepending a secret value to the message and computing the hash of the concatenated message at both sides (i.e. sender and receiver) [67].

Such applications are susceptible to length extension attack if they use a vulnerable hash algorithm and the attacker has access to the message and its hash value, and they know or guess the length of the secret, although they do not know the secret itself.

This attack is implemented by initiating the hash algorithm with a given internal state, which is the hash value of a secret prepended to a message, and appending attacker data as subsequent blocks by subsequently feeding the algorithm.

Next, the attacker will submit the computed hash value along with a concatenated message that involves the original message, padding of the original algorithm, and attacker data to the receiver.

Output transformation is a solution to resist length extension attack [19] that is employed by hash functions such as Modular Arithmetic Secure Hash (MASH) [2] and MD-6 [36].

2: Algebraic Cryptanalysis: Algebraic cryptanalysis is a method for attacking hash functions by solving polynomial systems of equations [68]. Some hash functions are reduced to instances of a satisfiability problem [69]. Such encoding of cryptographic algorithms and the subsequent reasoning is called *logical cryptanalysis* [70]. There are many examples of this type of attack to find second pre-images on round reduced variants of MD-4 [71], MD-5 [71] and SHA-1 [71], pre-images on a round reduced variant of MD-4 [72], and Keccak [73], [74].

3: Differential Cryptanalysis: Differential cryptanalysis seeks the relation between input differences and corresponding output differences. It is quite common to see eXclusive OR (XOR) as the difference operator. In addition, operators such as modular subtraction have been used to successfully attack MD-5 [75] and SHA-1 [76] hash functions.

C. Quantum Adversaries

This section discusses quantum adversaries. Companies such as IBM, Google, D-Wave, and Microsoft have developed quantum computers using various types of qubits.

D-Wave practical quantum devices have attracted research interest [77]. While up to eight trapped-ion qubits, about ten nuclear magnetic resonance qubits, and about ten optic qubits were considered as the maximum number of qubits in 2010 [78], in 2017, D-Wave announced and shipped its new commercial quantum computer equipped with 2000 qubits [79] (D-Wave uses Adiabatic quantum computation instead of gate-based quantum computation).

In 2019, D-Wave announced a new 5000 qubit device too.

Moreover, Microsoft announced that the company is going to offer a full-fledged topological quantum computing system which includes hardware, software, and programming languages, so that a free preview of the programming language which supports simulation of up to 30 logical qubits on personal computers (or up to 40 logical qubits on Azure) would be released by the end of the 2017 [80]. Microsoft Quantum Development Kit including Q# programming language is a released part of this stack.

Moreover, programming languages and software development kits (SDKs) such as Google qsim [81], IBM Qiskit [82], D-Wave Ocean [83], Scaffold [84], Quipper [85], and Microsoft LIQUi|> [86] facilitate the transition from high-level quantum algorithms to low-level gate representation, different architectures, error correction, and so on.

The emergence of these commercial quantum computers (D-Wave and in future Microsoft) connoted the existence of both opportunity of quantum cryptography schemes and threat of quantum adversaries.

Tackling the latter is referred to as post-quantum cryptography.

Table 2: Categories of Hash Functions – Analysis and applicability of attacks

Category	Algorithm	Advantages	Disadvantages	Applicable attacks
Hash functions based on a block cipher	A block cypher algorithm strengthened with one-wayness.	Reuse of existing block cypher effort and benefit from compact implementation due to the cypher reuse	Each function of these category maps an invertible block cypher algorithm to a noninvertible hash function. The second vulnerability map is between 64- and 128-bit block length of block cyphers and 224- to 1024-bit length of hash values required for securing against generic attacks. Furthermore, any vulnerability on the underlying block cypher may lead to a vulnerability on the associated hash function.	There are successful known side channel attacks for block cyphers such as DES and AES. These attacks are candidates for attacking the corresponding hash functions.
Hash functions based on algebraic structures	Reduction of pre-image resistance, second pre-image resistance, and collision search to computationally hard problems such as Integer Factorization and square root.	Provable security	Computation of these hash functions includes operations such as modular multiplication that are time-consuming and impose dependency between processed data and consumed time and power. Hence, many of the hash functions in this category are slow and are prone to side channel attacks.	Firstly, some computationally hard problems such as Integer Factorization which belong to BQP computational complexity class have known attacks for specific values of parameters. Secondly, due to the use of data dependent operations such as modular multiplication, these hash functions are prone to side channel attacks.
Custom designed hash functions	Designed from scratch and usually use bitwise operations to cause confusion and diffusion.	Usually have better performance than algebraic-based hash functions	The lack of provable security is commonplace in these hash functions.	The use of bitwise operations (e.g., XOR, AND, and circular shifts) improves their performance and reduces their vulnerability against side channel attacks. In different categories, however, successful attacks on these hash functions are reported.
Physical unclonable functions	Instead of explicit algorithms, they benefit from intrinsic randomness resulted from manufacturing variations.	The absence of any explicit algorithms makes cryptanalysis and quantum adversaries inapplicable. In addition, PUFs are tamper resistant. Hence, active physical attacks are inapplicable, too.	The use of PUFs requires skill in hardware description languages such as VHDL. In essence, PUFs map fixed length inputs to fixed length outputs, while arbitrary length input is desired. Moreover, they require error correction enhancements.	Some variants of side channel attacks (i.e. power analysis and timing attacks) enhanced by machine learning are used to read the output of these functions.
Quantum hash functions	Problems that are zero-knowledge against general quantum attacks are implemented by either a classical or a quantum algorithm.	Security is proved by reduction to computationally hard problems which belong to NP-BQP. That is, they belong to NP computational complexity class but not BQP.	Problems such as Approximate Closest Lattice Vector require considerable computational resources even for the computation of a hash value.	Being new, there are no reported attacks on these hash functions, and there is no evident proof of their strength against cryptanalysis.
Memory hard functions	Algorithms with huge amounts of memory usage and memory considerations including input-independent memory addressing, input-dependent memory addressing, and planned number of passes over the memory	Increasing the cost of ASIC-based attacks in terms of memory usage and energy consumption	As they intentionally lack the efficient input-to-output mapping property (Property I in Section 2), many resource constrained devices cannot afford to use these functions.	Cryptanalysis attacks which employ time-memory trade–off may target MHFs with input-independent memory addressing. Furthermore, side channel attacks may target MHFs with input-dependent memory addressing.
Optical hash functions	Use “confusion” and “diffusion” of modulated light instead of computation of a compression function	Low processing amounts due to taking advantage of natural randomness instead of computation	Complexity of setting up an optoelectronic system in a noisy environment	Being new, there are no reported attacks on these hash functions, and although there is no explicit algorithm, there is no evident proof of their strength against cryptanalysis.

In brief, quantum computing upsides include:

I. Significant speedup: There are quantum algorithms for some computationally hard problems such as Factoring and Ground State Estimation that are exponentially faster than the best classical algorithms for those problems [87]. Such problems belong to the Bounded-error Quantum Polynomial (BQP) computational complexity class which can be solved efficiently on a quantum computer with a bounded probability of error [88].

Their disadvantages include:

I. Error correction: Resisting communication channel noise errors such as bit-flip errors and phase errors, and tolerating computational faults such as faulty logic gates are necessary and are achieved through error correction techniques such as employing redundant qubits [88].

II. Scalability problems: Existence of noise and entanglement phenomena cause scalability problems [89].

Shor [90] introduced polynomial time algorithms for Factorization and Discrete Logarithms on quantum computers. Grover's quantum searching algorithm [91], [92] can find a 256-bit AES key in about 2^{128} quantum operations [93] and is used to find hash pre-images [94]. Furthermore, there are quantum attacks to find hash collisions [95].

In contrast to problems such as Factorization and Discrete Logarithms which have polynomial time quantum algorithms [90], post-quantum cryptography [96] tends to introduce problems that cannot be solved by quantum computers in polynomial time. Watrous [97] proved that problems such as Graph Isomorphism and Graph 3-coloring are zero-knowledge against general quantum attacks. Kashefi and Kerenidis [98] defined several quantum one-way functions such as Graph Non-Isomorphism, Approximate Closest Lattice Vector, and Group Non-Membership and generalize their results for any hard instance of Circuit Quantum Sampling problem as a candidate quantum one-way function.

D. Physical Security: Side Channel Attacks

Classical cryptanalysis views steps of algorithms as transformation of inputs to outputs. Conversely, physical security views specific characteristics imposed by an implementation of those steps which are running on a specific processor in a specific environment. Physical attacks may or may not depackage the chip; such situations are called invasive or noninvasive attacks, respectively. In addition, physical attacks may or may not try to tamper with the proper functioning of the device and are called active or passive attacks, respectively [2]. Side-channel attacks, or environmental attacks, exploit dependency of information such as running time, power consumption, and electromagnetic emissions of operated data and performing instructions to (statistically) learn about an algorithm's internal state [2], [99] or expose the device's secrets. The SHA-3 finalists were evaluated against three variants of side channel attack: timing attack, power analysis, and electromagnetic analysis. The evaluation declared the sufficient security margin of all finalists and found collisions on the round reduced variant of Keccak [99]. Cryptographic algorithms prevent such attacks by avoiding the use of data-dependent or power-dependent operations such as multiplications, data-dependent rotations, and table lookups.

PUFs (See Section V-D) are tamper resistant variants of hash functions, but there are polynomial time side channel attacks on PUFs [100] that enable the attacker to read the generated output value.

In addition to physical security, there are adversaries which consider an implementation of a security primitive from the viewpoint of software and programming language flaws. The buffer overflow found on the C language implementation of MD-6 is an instance of such software implementation attacks [19].

Hash Function Categories

This section describes cryptographic hash functions in seven categories and analyses the strengths and vulnerabilities of each category (See Table 2). The proposed seven-category classification includes hash functions based on a block cipher, hash functions based on algebraic structures, custom-designed hash functions, PUFs, quantum hash functions, MHFs, and optical hash functions. To the best of our knowledge, the last four mentioned categories have not been sufficiently addressed in most existing surveys [18], [19], [26]-[29].

A. Hash Functions Based on Block Ciphers

Developed mostly based on DES and AES, these hash functions reuse underlying block ciphers to achieve a compact implementation. The main challenges of these hash functions lie in designing a noninvertible construction based on an invertible block cipher. The SHA-3 finalist BLAKE [101] and Russian standard hash Streebog [24] are two known hash functions of this category.

B. Hash Functions Based on Algebraic Structures

Most hash functions in this category use computationally hard problems such as Factorization, Discrete Logarithm, Knapsack, Lattice Problems, and Elliptic Curves and prove their security by reduction [102]. Some of these hash functions, though, allow the insertion of trapdoors to construct collisions by the person who chooses the design parameters [2]. The functions based on modular arithmetic suffer from being slow. There are many attacks for specific instances of hard problems, such as RSA [103]. As an example, collision resistancy of Very Smooth Hash (VHA) [104] is reduced to find nontrivial

modular square roots, but this function is not pre-image resistant [105]. Modular Arithmetic Secure Hash (MASH) was published as an International Organization for Standardization (ISO) standard on December 1998 and was reviewed and re-confirmed as current version of standard in 2022 [106]. It has strong output transformation but its security is not supported by a mathematical proof. Finite field is used to define some hash functions [107]. A recent survey on hash functions based on computational problems defined on lattices was provided by Mishra et al [108]. Furthermore, hash functions based on Cellular Automata [109] are newly introduced members of this category.

Finally, another important family of hash functions comprises chaos-based hash functions. A chaotic system behaves in an unpredictable but deterministic manner and is highly sensitive to initial conditions, so a very small change in its initial state may have a large effect on its later state. A chaotic map is a mathematical function which states such a chaotic behavior in one- or multi-dimensions. As an example, Teh et al. [110] presented a compression function based on a one-dimensional chaotic map and used Merkle–Damgard construction to process arbitrary-length messages.

C. Custom-Designed Hash Functions

Known cryptographic hash functions including MD-2, MD-4, SHA-1, SHA-2, and SHA-3 (Keccak) are instances of this category. These algorithms are designed independent from other security primitives. Although these hash functions do not provide provable security and their security depends on confusion and diffusion, the use of bitwise operations such as XOR, AND, and circular shifts leads to low processing time and partial security against side channel attacks, even though there are some reports of such attacks [111].

D. Physical Unclonable Functions (PUFs)

PUFs are hardware based security primitives and provide challenge response behavior based on manufacturing variations that occur on a small scale. Their intrinsic unpredictablity stems from random elements (e.g., various gate delay) in their manufacturing process [112], [2]. Depending on the usage, this challenge response behavior may be provided in an invertible or non-invertible manner [113]. An individual PUF device, however, cannot be practically cloned or copied, even with access to the exact manufacturing process that produced it in the first place. This intrinsic randomness reduces computational costs, thus making PUFs a candidate for the security of resource-constrained devices such as embedded systems [114], and IoT [113].

There are two notable PUF types: Weak PUFs and Strong PUFs; the former accepts one or a few challenges and is employed as a secret key for device specific encryption, while the latter accepts, possibly, an exponential number of challenges and is considered as a physical hash function [115]. SRAM PUFs and their variants are the most popular implementation of Weak PUFs and Arbiter PUFs, and their variants are the most popular implementation of electrically Strong PUFs. Weak PUFs suffer from cloning and invasive attacks (e.g., Helfmeier et al. [116] created a physical clone of a SRAM PUF using Focused Ion). Cloning and invasive attacks are hardly applicable on Strong PUFs. The most common attacks on Strong PUFs are modeling attacks [117], side channel attacks [118], and the combination of both [100], [119].

To conclude, PUFs benfit from the following advantages:

I. Instead of storing a hash value or a sectret key on the device that includes both security consideration and additional device memory cost, the PUF response is derived when needed [115].
II. Most types of PUFs are tamper-resistant [115], but there are some side channel attacks enhanced by machine learning [100].

and suffer from the following disadvantages:

I. PUFs are prone to error and need to employ an error correction mechanism. Depending on PUF type, error correction may be executed on a PUF holding device or on a communication server [115].
II. In contrast to non-physical approaches, PUFs are prone to aging [115].

In essence, PUFs are maps between fixed length inputs and fixed length outputs, while arbitrary length input is desired. Therefore, PUFs are widely used for authentication and rarely used for integrity checks (a common application of hash functions).

Finally, PUFs based on nanotechnology are the recently reported trend of PUF design [120].

E. Quantum Hash Functions

There are two sub-categories of quantum hash functions, i.e. hard problems which belong to postquantum cryptography and hash functions based on quantum state. The former was described in Section 4.3, and the latter is discussed in the current section. In addition to the mentioned subcategories, there are quantum hash functions which operate on classical inputs and produce classical outputs [121].

Ziiatdinov [122] and Yang et al. [121] attributed the first state-based quantum hash function to Buhrman et al. [123], who introduced the notion of quantum fingerprinting. Ablayev and Vasiliev [124], [125] introduced quantum hash functions that map input data to quantum states so that the functions have pre-image resistance (sampling property), second pre-image resistance, and collision resistance properties. Ablayev et al. [126] discussed the reverse relation between the pre-

image resistance and collision resistance properties of quantum hash functions and introduced a construction to build balanced quantum hash functions.

F. Memory-Hard Functions

There are cases such as cryptocurrency mining and password hashing in which a hash function without an efficient input-to-output mapping property (*Property I* in Section 2) is desired. In contrast to the design goals of distributed electronic payment systems such as Bitcoin, multicore CPUs, GPUs, and dedicated ASIC modules are used to accelerate cryptocurrency mining at a low cost. This consolidates the computing power of the network. Some ASIC miners are roughly 200,000 times faster and 40,000 times more energy efficient than a modern multi-core CPU [127]. Dictionary attacks on hashed password databases are further examples of such parallel computation.

The ASIC resistance property (*Property X* in Section II) aims to reduce attackers' massively parallel advantage. To this end, MHFs [128] and BHFs [127] were introduced to increase the hardware capital cost and energy consumption, respectively. Percival [128] put forward the MHFs idea that with an increase in the size of a hash derivation circuit, the number of possible circuits on a given area of silicon will decrease. Furthermore, he introduced the *scrypt* hash function [128], [129] as the first instance of MHF.

Input-independent memory addressing, input-dependent memory addressing, and number of passes over the memory are major considerations in designing an MHF. For example, *Argon2* hash function [22] includes the following tree variants:

I. *Argon2d*: It uses data-dependent memory access and targets the design of cryptocurrency Proof-of-Work (PoW).
II. *Argon2i*: It uses data-independent memory access to resist side channel attacks and includes more passes over the memory in comparison with Argon2d. Argon2i aims to secure password hashing.
III. *Argon2id*: It is not a part of Argon2 hash function proposal [22] and use a sequential composition of data-depending and data-independent memory accesses. First half pass uses data-independent memory access and the second half uses data-dependent memory access.

As a last example of MHFs, Zamanov et al. [34] evaluated the memory demand of Equihash and Ethash algorithms. The former increases PoW memory usage based on the birthday problem, while the latter fills a huge amount of memory and searches within it.

Although MHFs incur additional capital costs, ASICs require far less energy than CPUs. To this end, BHFs define a large number of planned memory accesses to avoid the energy saving of ASIC hash engines [127].

G. Optical Hash Functions

Because of physical properties of light such as velocity and its parallel nature, light-based computing is promising and has been shown to outperform electronic computing in some cases [130]. Optical hash functions are photoelectric systems which encode blocks into images known as the "information plane" [131] and replace computations of a compression function with "confusion" and "diffusion" of modulated light [132]. Amplitude-only spatial light modulator, phase-only spatial light modulator, charge coupled devices along with lenses [131], half mirrors [131], and/or scattering media [132] are the basic constituents of such systems. As an example, Wen-Qi et al. [132] proposed an optical hash function which is based on scattering media and provides the avalanche effect and collision resistance. As another example, He and Peng [131] proposed two optical hash functions based on phase-truncated Fourier transform and interference phenomena (i.e. two beam interference). Last but not least on our list of examples, as noise inherent in free space setup can affect the security and performance of beam interference and phase truncation-based hash functions, Kumar et al. [133] proposed an optical hash function based on superposition.

Mobile Service Requirements

Mobile devices can consume some services and also provide some other services, but they have several constraints on their resources which may jeopardize the Quality of Service (QoS). On the other hand, as mobile devices roam between environments, they are exposed to more attacks than stationary computers. Hence, lightweight but not less secure cryptographic hash functions which secure interactions of resource-constrained devices are urgently needed. Mobile service requirements are as follows:

I. Roaming may cause inaccessibility of some resources and accessibility to some others. To aid service continuity, hash functions are used to identify identical alternative resources and mutual authentication of the mobile device and remote servers [134].
II. Most mobile devices have low processing power in comparison with desktop computers.
III. Most mobile devices have small memory size in comparison with desktop computers.
IV. Limited battery capacity makes energy consumption an important consideration for mobile devices. Not only does WS-Security hash computation required by service invocation consume energy, but also the battery usage of hash computation is important to avoid power analysis side channel attacks [135].
V. Mobile device bandwidth is limited by the network interfaces of that device and by the network being

used. This limit mediated mobile WS-Security solutions usage [136], [137].

VI. From time to time, mobile devices undergo connection intermittence caused not only by roaming, but also by things such as other wireless devices, microwave ovens, and other devices with poorly shielded cabling.

VII. Some mobile devices have multiple network interfaces such as Wi-Fi, Bluetooth, NFC, and GPRS (in addition to LoRaWAN and ZigBee for IoT). To benefit from multi-homed architectures, authentication and integrity achieved by hash functions are urgent needs for mobile service communications [138].

Hence, low processor usage, thrifty memory usage, and limited battery usage are urgent needs of application-specific hash functions for mobile services. In addition, due to connection intermittence and bandwidth limitation, mobile security-related computations such as hash computation can hardly be delegated to servers that are available through wireless connections. For simplicity, the application-specific hash function for mobile services will be referred to hereafter as mobile hash functions.

Such mobile hash functions need to cope with the mentioned limitations, and it is desirable that they benefit from multi-homed architectures. Table 3 shows the appropriateness of each hash function category for satisfying mobile service requirements. As Table 3 outlines, optical hash functions and state-based quantum hash functions are not applicable for mobile devices. Algebraic-based functions benefit from provable security but have high computational costs. Bitwise equivalent of algebraic structures that belongs to post-quantum cryptography seemed like a good idea, but we could not find such algebraic-based hash functions in practice. PUFs have very low computational costs and communicate just challenge-responses. In addition, PUFs are available for IoT nodes [139]. Hence, we suggest PUFs with polynomial-time error correction for mobile service hashing.

Application Scenarios

All applications do not have the same requirements for security and performance. There are a number of application scenarios for cryptographic hash functions. Four scenarios and their corresponding analysis to select appropriate cryptographic hash functions are presented in Table 4. The first scenario benefits from the parallel processing capability of hash functions such as MD-6. The second scenario uses the intrinsic randomness of PUFs to lighten hash computation load for resource constrained sensor nodes. The third shows the usage of hash chains for process authentication. Finally, the last scenario shows the need for output transformation in the lack of encryption.

Conclusion

Massive usage, significant competitions such as the SHA-3 competition, the Password Hashing competition and the NIST lightweight competition, and nationwide hash standards [20], [21], [23]-[25] have led to the introduction of new hash functions and new hash function constructions. To the best of our knowledge, recent research and competitions make the following futuristic trends possible: Resource constrained devices are used in IoT solutions such as smart farming and smart cities. Security plays a crucial role in the success such systems so that employing hash functions need to be both resource efficient and side-channel resistant [141]. Hence, lightweight hash functions received great attention in recent years so that IoT specific hash functions emerged and NIST lightweight competition is ongoing since 2018 [142]-[144]. In contrast to the lightweight design of these hash functions, it is important that a hash function cannot be computed too fast on massively parallel computers and quantum computers. Hence, evaluation of hash functions on quantum computers is a recent measure to avoid brute force attacks [145].

Table 3: Appropriateness of each Hash Function category for satisfying mobile service requirements

Row	Hash Function Category		Mobile Service Hash Consideration				
			Processing	Memory	Battery	Security	Applicability
1	Hash functions based on a block cipher		High				
2	Hash functions based on algebraic structures		High			Proven	
3	Custom-designed hash functions		Low				
4	Physical unclonable functions		Very low	None or very low (depending on PUF type)			
5	Quantum hash functions	Quantum states	No reported work (have not found yet)				Not applicable
		Post-quantum cryptography	High				
6	Memory-hard functions			High			
7	Optical hash functions		No reported work (have not found yet)				Not applicable

Table 4: Application scenarios – selecting appropriate Hash Function

Row	Scenario Name	Scenario	Analysis
1	A file server on a multiprocessor host	A multiprocessor file server stores some large multimedia files. This server needs to provide the hash value of each file as a checksum. Users can download files along with corresponding checksums. To ensure a file has not been tampered with after the checksum was created, user computes the hash of the downloaded file and compares it with the checksum.	Computing hash for large files connotes the need for fast computation. It may be obtained by using a fast hash function such as BLAKE [101], [140] (BLAKE 2 or 3) or a multiprocessing support hash function such as MD-6 [36]. The multiprocessor server indicates the latter function as choice.
2	Message authentication in a sensor network	A sensor network sends monitored data to a server. A hash function is used for message authentication. Each sensor node has limited memory and limited processing speed. More importantly, each sensor node operates with limited battery energy and will die as its energy is consumed.	Resource constraints of sensor nodes and the reverse relationship between energy consumption and node lifetime suggest the use of intrinsic properties of sensors instead of running a hash algorithm on these nodes. Hence, PUFs [139] are appropriate for this scenario.
3	One-time passwords	In a geographically distributed organization, it is required that two processes hosted on different servers authenticate and communicate with each other. There is no deployed authentication (or encryption) facilities such as Primary Key Infrastructure (KPI).	This scenario may benefit from one-time passwords that are a hash chain made by consecutive computation of hash values and using the hash values in descending order (using last value first). Any hash function that supports the one-way property is appropriate for this scenario, so that an eavesdropper cannot use an observed password to compute the next valid password.
4	Authentication and integrity without encryption	A key is shared between sender and receiver. To send a message, the sender hashes that message prepended by the shared key. Then the message along with the hash value is transmitted to the receiver. Having the shared key, the receiver will hash the received message prepended by the shared key and compares it with the received hash value.	This scenario is prone to length extension attack (Section 4.2.1). It allows the attacker to forge messages with the same prefix. Hence, both authentication and integrity will be lost. Section 4.2.1 pointed out that exposure of the internal state of the hash function causes this vulnerability. Hence, hash functions benefitting from output transformation such as SHA-3 (Keccak) and MASH (section 5.2) are appropriate for this scenario.

As mentioned, PUFs based on nanotechnology are the recently reported trend of PUF design [120]. Last but not least, optical computing has a long history to trace back and was introduced 60-year ago [146], but optical hash functions were introduced in recent years are among the futuristic trend of hash functions. In addition, application-specific properties have been defined for applications such as cryptocurrency and video hashing. In this article, we discussed 11 properties of hash functions (Section 2), overviewed the concepts of compression function and domain extension, and outlined four iterative and three noniterative hash function constructions and combiners (Section 3). The current research also investigated those hash functions and proposed a seven-category classification (Section 5). To the best of our knowledge, four out of seven categories have not been sufficiently addressed in most existing surveys [18], [19], [26]-[29]. In addition, this article discussed some attacks affecting each category (Table 2) and summarized what effective attacks entail (Section 4).

Furthermore, considering the prevalence of mobile devices, this paper discussed mobile service requirements on hash functions (Section 6), outlined how each hash function category fits these requirements (Table 3), and suggested (strong) PUFs with polynomial-time error correction for mobile service hashing. In addition, the bitwise equivalent of algebraic structures that belong to post-quantum cryptography seemed like a good idea, but we could not find such algebraic-based hash functions in practice. Finally, to clarify the usage, four application scenarios and their corresponding analysis to select appropriate cryptographic hash functions were presented (Table 4). The authors aim to extend this work by extracting patterns which fulfill the 11 properties discussed in second section. This extension, along with the other mentioned benefits, can assist design, choice, and analysis of hash functions.

Author Contributions

Second and third authors supervised this research by sketching roadmap, and evaluating the results at each

step. First author searched in authentic journals and research repositories to gather all relevant papers, and read the selected papers in details. In addition, he made a comparison of investigated hash functions.

All authors discussed and analyzed the results and cooperatively summed up the work.

Acknowledgment

The authors gratefully thank the anonymous reviewers and the editor of JECEI.

Conflict of Interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Abbreviations

MHF	Memory-hard Functions
BHF	Bandwidth-hard Functions
PUF	Physical Unclonable Function
SHA	Secure Hash Algorithm
UOWHF	Universal One-Way Hash Functions
WS-Security	Web Services Security

References

[1] J. Hoffstein, J. Pipher, J. H. Silverman, An introduction to mathematical cryptography. New York: Springer, 2008.

[2] H. C. A. van Tilborg, S. Jajodia, Eds., Encyclopedia of cryptography and security. Springer Science+Business Media, 2011.

[3] J. Keller and S. Wendzel, "Reversible and plausibly deniable covert channels in one-time passwords based on hash chains," Appl. Sci., 11(2): 731, 2021.

[4] W. Stallings, Cryptography and network security: principles and practice, sixth ed. Pearson Education, 2014.

[5] C. Wang, S. J. Li, D. Wang, Q. H. Wang, "P-28: A method of holographic encryption based on hash function," Dig. Tech. Pap., 47(1): 1228–1230, 2016.

[6] L. C. Washington, Elliptic curves: number theory and cryptography, second ed. Boca Raton, Fl: Chapman & Hall/Crc, 2008.

[7] R. C. Merkle, "A certified digital signature," in Proc. Conf. on the Theory and Application of Cryptology: 218-238, 1989.

[8] J. Rosenberg, D. L. Remy, Securing web services with WS-security: demystifying WS-security, WS-policy, SAML, XML signature, and XML encryption. Indianapolis, Ind.: Sams, 2004.

[9] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker, Eds., Web services security: SOAP message security 1.1. OASIS, 2006. Last accessed: Jan. 7, 2023.

[10] L. Demir, A. Kumar, M. Cunche, C. Lauradoux, "The pitfalls of hashing for privacy," IEEE Commun. Surv. Tutor., 20(1): 551-565, 2018.

[11] M. Wang, M. Duan, J. Zhu, "Research on the security criteria of hash functions in the blockchain," in Proc. the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts: 47-55, 2018.

[12] S. Abed, R. Jaffal, B. J. Mohd, M. Al-Shayeji, "An analysis and evaluation of lightweight hash functions for blockchain-based IoT devices," Cluster Comput., 24(4): 3065-3084, 2021.

[13] A. Kuznetsov, I. Oleshko, V. Tymchenko, K. Lisitsky, M. Rodinko, A. Kolhatin, "Performance analysis of cryptographic hash functions suitable for use in blockchain," Int. j. comput. netw. inf. secur., 13(2): 1-15, 2021.

[14] A. M. Antonopoulos, Mastering bitcoin: Programming the open blockchain, 2nd ed. O'Reilly Media, 2017.

[15] J. Garay, A. Kiayias, N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in Proc. Annu. Int. Conf. on the Theory and Applications of Cryptographic Techniques: 281-310, 2015.

[16] G. Wu, J. Han, Y. Guo, L. Liu, G. Ding, Q. Ni, L. Shao, "Unsupervised deep video hashing via balanced code for large-scale video retrieval," IEEE Trans. Image Process., 28(4): 1993-2007, 2019.

[17] M. S. Jan, M. Afzal, "Hash chain based strong password authentication scheme," in Proc. 13th Int. Bhurban Conf. on Applied Sciences and Technology (IBCAST): 355-360, 2016.

[18] A. A. Alkandari, I. F. Al-Shaikhli, M. A. Alahmad, "Cryptographic hash function: A high level view," in Proc. 2013 Int. Conf. on Informatics and Creative Multimedia: 128-134, 2013.

[19] B. Preneel, "The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition," in Proc. Cryptographers' track at the RSA Conf.: 1-14, 2010.

[20] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "Keccak," in Proc. 32nd Annu. Int. Conf. on the Theory and Applications of Cryptographic Techniques: 313-314, 2013.

[21] W. Stallings, "Inside SHA-3," IEEE Potentials, 32(6): 26-31, 2013.

[22] A. Biryukov, D. Dinu, D. Khovratovich, "Argon2: new generation of memory-hard functions for password hashing and other applications," in Proc. 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbruecken, Germany: 292-302, 2016.

[23] S. Shen, X. Lee, R. Tse, W. Wong, Y. Yang, "The SM3 cryptographic hash function," draft-sca-cfrg-sm3-02, 2018.

[24] V. Dolmatov, A. Degtyarev, "GOST R 34.11-2012: hash function," RFC 6986, 2013.

[25] R. Oliynykov, I. Gorbenko, O. Kazymyrov, V. Ruzhentsev, O. Kuznetsov, Y. Gorbenko, A. Boiko, O. Dyrda, V. Dolgov, A. Pushkaryov, "A new standard of Ukraine: The Kupyna hash function," Cryptology ePrint Archive, DSTU 7564: 2014, 2015.

[26] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, "Cryptographic hash functions: A survey," Department of Computer Science, University of Wollongong, Technical Report 95-09, Jul. 1995.

[27] J. Delvaux, R. Peeters, D. Gu, I. Verbauwhede, "A survey on lightweight entity authentication with strong PUFs," ACM Comput. Surv., 48(2): 1-42, 2015.

[28] I. Mironov, "Hash functions: Theory, attacks, and applications," Microsoft Research, Silicon Valley Campus, 1-22, Nov. 2005.

[29] R. Purohit, U. Mishra, A. Bansal, "A survey on recent cryptographic hash function designs," Int. J. Emerging Trends & Technology in Computer Science (IJETTCS), 2(1): 2278-6856, 2013.

[30] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, S. Goldfeder, Bitcoin and cryptocurrency technologies: A comprehensive introduction. Princeton, NJ: Princeton University Press, 2016.

[31] M. Rjaško, "On chosen target forced prefix preimage resistance," Tatra Mt. Math. Publ., 47(1): 115-135, 2010.

[32] B. Mennink, "Increasing the flexibility of the herding attack," Inf. Process. Lett., 112(3): 98-105, 2012.

[33] E. Andreeva, B. Mennink, "Provable chosen-target-forced-midfix preimage resistance," in Int. Workshop on Selected Areas in Cryptography: 37-54, 2011.

[34] A. R. Zamanov, V. A. Erokhin, P. S. Fedotov, "ASIC-resistant hash functions," in Proc. 2018 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus): 394-396, 2018.

[35] H. Chen, Y. Wo, G. Han, "Multi-granularity geometrically robust video hashing for tampering detection," Multimed. Tools Appl., 77(5): 5303-5321, 2017.

[36] R. L. Rivest, B. Agre, D. V. Bailey, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, Y. L. Yin, "The MD6 hash function–a proposal to NIST for SHA-3," Submission to NIST, 2(3), 2008.

[37] R. Reischuk, M. Hinkelmann, "One-way functions - mind the trap - escape only for the initiated," in Proc. Algorithms Unplugged, Berlin, Heidelberg: Springer Berlin Heidelberg, 131-139, 2011.

[38] O. Goldreich, S. Goldwasser, S. Halevi, "Collision-free hashing from lattice problems," in Proc. Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation, Berlin, Heidelberg: Springer Berlin Heidelberg, 30-39, 2011.

[39] W. Hu, N. Xie, L. Li, X. Zeng, S. Maybank, "A survey on visual content-based video indexing and retrieval," IEEE Trans. Syst. Man Cybern. C Appl. Rev., 41(6): 797-819, 2011.

[40] A. Appleby, Murmurhash 3.0, 2016. Last accessed: Aug. 25, 2022.

[41] C. Estébanez, Y. Saez, G. Recio, P. Isasi, "Performance of the most common non-cryptographic hash functions," Softw. Pract. Exp., 44(6): 681-698, 2014.

[42] S. Chen and C. Jin, "A second preimage attack on zipper hash," Secur. Commun. Netw.,. 8(16): 2860–2866, 2015.

[43] M. Liskov, "Constructing an ideal hash function from weak ideal compression functions," in Proc. 13th Int. Workshop on Selected Areas in Cryptography: 358-375, 2006.

[44] B. Denton, R. Adhami, "Modern hash function construction," in Proc. the Int. Conf. on Security and Management (SAM): 479-483, 2011.

[45] Z. Bao, I. Dinur, J. Guo, G. Leurent, L. Wang, "Generic attacks on hash combiners," J. Cryptology, 33(3): 742-823, 2019.

[46] M. Fischlin, A. Lehmann, D. Wagner, "Hash function combiners in TLS and SSL," in Proc. Cryptographers' Track at the RSA Conf.: 268–283, 2010.

[47] D. X. Charles, K. E. Lauter, E. Z. Goren, "Cryptographic hash functions from expander graphs," J. Cryptology, 22(1): 93-113, 2009.

[48] C. Petit, J. J. Quisquater, "Cryptographic hash functions and expander graphs: The end of the story?," in Proc. The New Codebreakers, Berlin, Heidelberg: Springer Berlin Heidelberg: 304–311, 2016.

[49] B. A. Forouzan, Cryptography & network security. Maidenhead, England: McGraw Hill Higher Education, 2007.

[50] R. C. Merkle, "One way hash functions and DES," in Proc. Conf. on the Theory and Application of Cryptology: 428-446, 1989.

[51] I. B. Damgård, "A design principle for hash functions," in Conf. on the Theory and Application of Cryptology: 416-427, 1989.

[52] E. Andreeva, G. Neven, B. Preneel, T. Shrimpton, "Seven-property-preserving iterated hashing: ROX," in Proc. 13th Int. Conf. on the Theory and Application of Cryptology and Information Security: 130-146, 2007.

[53] V. Shoup, "A composition theorem for universal one-way hash functions," in Int. Conf. on the Theory and Applications of Cryptographic Techniques: 445-452, 2000.

[54] I. Mironov, "Hash functions: From merkle-damgård to shoup," in Proc. Int. Conf. on the Theory and Applications of Cryptographic Techniques: 166-181, 2001.

[55] E. Andreeva, C. Bouillaguet, O. Dunkelman, J. Kelsey, "Herding, second preimage and trojan message attacks beyond Merkle-Damgård," in Proc. 16th Int. Workshop on Selected Areas in Cryptography: 393-414, 2009.

[56] J. Katz, Y. Lindell, Introduction to modern cryptography, 2nd ed. Philadelphia, PA: Chapman & Hall/CRC, 2014.

[57] K. Malvoni, J. Knezovic, "Are your passwords safe: Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware," in Proc. 8th USENIX Workshop on Offensive Technologies (WOOT 14): 1-7, 2014.

[58] H. J. Mun, S. Hong, J. Shin, "A novel secure and efficient hash function with extra padding against rainbow table attacks," Cluster Computing, 21(1): 1161-1173, 2017.

[59] E. Conrad, S. Misenar, J. Feldman, Cissp Study Guide, 2nd ed. Waltham, MA, USA: Syngress Publishing, 2012.

[60] J. Guo, S. Ling, C. Rechberger, H. Wang, "Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2," in 16th Int. Conf. on the Theory and Application of Cryptology and Information Security: 56-75, 2010.

[61] K. Aoki, Y. Sasaki, "Preimage attacks on one-block MD4, 63-step MD5 and more," in Proc. 15th Annu. Int. workshop on selected areas in cryptography: 103-119, 2008.

[62] S. Knellwolf, D. Khovratovich, "New preimage attacks against reduced SHA-1," in Proc. 32nd Annu. Cryptology Conf.: 367-383, 2012.

[63] K. Aoki, Y. Sasaki, "Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1," in Proc. 29th Annu. Int. Cryptology Conf.: 70-89, 2009.

[64] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, L. Wang, "Preimages for step-reduced SHA-2," in Proc. 15th Int. Conf. on the Theory and Application of Cryptology and Information Security: 578-597, 2009.

[65] J. Kelsey, T. Kohno, "Herding hash functions and the Nostradamus attack," in Proc. Annu. Int. Conf. on the Theory and Applications of Cryptographic Techniques: 183-200, 2006.

[66] M. Stamp, R. M. Low, Applied cryptanalysis: breaking ciphers in the real world. Hoboken, N.J.: Wiley-Interscience, 2007.

[67] W. Stallings, Network security essentials: Applications and standards, 4th ed. Prentice Hall, 2010.

[68] G. V. Bard, Algebraic Cryptanalysis. Springer, 2009.

[69] D. Jovanović, P. Janičić, "Logical analysis of hash functions," in 5th Int. Workshop on Frontiers of Combining Systems: 200-215, 2005.

[70] F. Massacci, L. Marraro, "Logical cryptanalysis as a SAT problem," J. Automated Reasoning, 24(1): 165-203, 2000.

[71] F. Legendre, G. Dequen, M. Krajecki, "Encoding hash functions as a sat problem," in Proc. 2012 IEEE 24th Int. Conf. on Tools with Artificial Intelligence, 1: 916-921, 2012.

[72] D. De, A. Kumarasubramanian, R. Venkatesan, "Inversion attacks on secure hash functions using SAT solvers," in 10th Int. Conf. on Theory and Applications of Satisfiability Testing: 377-382, 2007.

[73] P. Morawiecki, M. Srebrny, "A SAT-based preimage analysis of reduced Keccak hash functions," Inf. Process. Lett., 113(10-11): 392-397, 2013.

[74] E. Homsirikamol, P. Morawiecki, M. Rogawski, M. Srebrny, "Security margin evaluation of SHA-3 contest finalists through SAT-based attacks," in Proc. 11th IFIP Int. Conf. on Computer Information Systems and Industrial Management: 56-67, 2012.

[75] X. Wang, H. Yu, "How to break MD5 and other hash functions," in Proc. 24th Annu. Int. Conf. on the Theory and Applications of Cryptographic Techniques: 19-35, 2005.

[76] X. Wang, Y. L. Yin, H. Yu, "Finding collisions in the full SHA-1," in Proc. 25th Annu. Int. Cryptology Conf.: 17-36, 2005.

[77] W. Vinci, T. Albash, A. Mishra, P. A. Warburton, D. A. Lidar, "Distinguishing classical and quantum models for the d-wave device," Cornell University Library, 2014.

[78] E. Knill, "Quantum computing," Nature, 463(7280): 441-443, 2010.

[79] "D-Wave announces first order for 2000Q quantum computer," ID Quantique, 24-Feb-2017. Last accessed: Jan. 10, 2023. Available at: D-Wave Announces D-Wave 2000Q Quantum Computer and First System Order — D-Wave Government (dwavefederal.com).

[80] "With new Microsoft breakthroughs, general purpose quantum computing moves closer to reality," Stories, Sep. 25, 2017. Last accessed: Jan. 10, 2023.

[81] "Quantum simulator," Google Quantum AI. Last accessed: Jan. 10, 2023.

[82] "qiskit.org," Qiskit.org. Last accessed: Jan. 10, 2023.

[83] "D-wave ocean software documentation — ocean documentation 5.3.0 documentation," Dwavesys.com. Last accessed: Jan. 10, 2023.

[84] A. J. Abhari et al., "Scaffold: Quantum programming language," Princeton univ NJ dept of computer science, Rep. TR-934-12, 2012. Last accessed: Jan. 10, 2023.

[85] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, "Quipper: a scalable quantum programming language," in Proc. the 34th ACM SIGPLAN Conf. on Programming language design and implementation: 333-342, 2013.

[86] "Language-Integrated Quantum Operations: LIQUi|>," Microsoft Research. Last accessed: Jan. 10, 2023.

[87] S. Patil, A. JavadiAbhari, C. F. Chiang, J. Heckey, M. Martonosi, F. T. Chong, "Characterizing the performance effect of trials and rotations in applications that use Quantum Phase Estimation," in Proc. 2014 IEEE Int. Symposium on Workload Characterization (IISWC): 181-190, 2014.

[88] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10Th Anniversary Edition. Cambridge, England: Cambridge University Press, 2010.

[89] S. Imre, "Quantum computing and communications – Introduction and challenges," Comput. Electr. Eng., 40(1): 134-141, 2014.

[90] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in Proc. 35th Annu. Symposium on Foundations of Computer Science: 124-134, 1994.

[91] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proc. the twenty-eighth Annu. ACM symposium on Theory of computing: 212-219, 1996.

[92] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," Phys. Rev. Lett., 79(2): 325-328, 1997.

[93] D. J. Bernstein, "Grover vs. mceliece," in Third Int. Workshop on Post-Quantum Cryptography: 73-80, 2010.

[94] P. Wang, S. Tian, Z. Sun, N. Xie, "Quantum algorithms for hash preimage attacks," Quantum Eng., 2(2): 2020.

[95] X. Dong, S. Sun, D. Shi, F. Gao, X. Wang, L. Hu, "Quantum collision attacks on AES-like hashing with low quantum random access memories," in Proc. 26th Int. Conf. on the Theory and Application of Cryptology and Information Security: 727-757, 2020.

[96] D. J. Bernstein, "Introduction to post-quantum cryptography," in Post-Quantum Cryptography, Berlin, Heidelberg: Springer Berlin Heidelberg, 1-14, 2009.

[97] J. Watrous, "Zero-Knowledge against Quantum Attacks," SIAM j. comput., 39(1): 25-58, 2009.

[98] E. Kashefi, I. Kerenidis, "Statistical Zero Knowledge and quantum one-way functions," Theor. Comput. Sci., 378(1): 101-116, 2007.

[99] S. J. Chang et al., "Third-round report of the SHA-3 cryptographic hash algorithm competition," National Institute of Standards and Technology, Gaithersburg, MD, Rep. 7896, Nov. 2012.

[100] U. Rührmair et al., "Efficient Power and Timing Side Channels for Physical Unclonable Functions," in Proc. 16th Int. Workshop on Cryptographic Hardware and Embedded Systems, 476-492, 2014.

[101] J. P. Aumasson, W. Meier, R. C. W. Phan, L. Henzen, The hash function BLAKE. Springer-Verlag Berlin Heidelberg, 2014.

[102] A. Bauer, E. Jaulmes, E. Prouff, J.-R. Reinhard, J. Wild, "Horizontal collision correlation attack on elliptic curves: - Extended Version - ," Cryptogr. Commun., 7(1): 91-119, 2015.

[103] S. Y. Yan, Cryptanalytic attacks on RSA. Springer, 2008.

[104] S. Contini, A. K. Lenstra, R. Steinfeld, "VSH, an efficient and provable collision-resistant hash function," in Proc. 25th Int. Conf. on the Theory and Applications of Cryptographic Techniques: 165-182, 2006.

[105] M. J. O. Saarinen, "Security of VSH in the real world," in Proc. 7th Int. Conf. on Cryptology in India: 95–103, 2006.

[106] ISO/IEC 10118-4:1998 Information technology — Security techniques — Hash-functions — Part 4: Hash-functions using modular arithmetic, ISO/IEC 10118-4:1998, Dec. 1998.

[107] S. Kölbl, E. Tischhauser, P. Derbez, A. Bogdanov, "Troika: a ternary cryptographic hash function," Des. Codes Cryptogr., 88(1): 91-117, 2020.

[108] N. Mishra, S. H. Islam, S. Zeadally, "A comprehensive review on collision-resistant hash functions on lattices," J. Inf. Secur. Appl., 58: 102782, 2021.

[109] V. Manuceau, "About a fast cryptographic hash function using cellular automata ruled by far-off neighbours," Int. j. eng. trends technol., 69(2): 39-41, 2021.

[110] J. S. Teh, K. Tan, M. Alawida, "A chaos-based keyed hash function based on fixed point representation," Cluster Comput., 22(2): 649-660, 2018.

[111] M. Zohner, M. Kasper, M. Stöttinger, S. A. Huss, "Side channel analysis of the SHA-3 finalists," in Proc. 2012 Design, Automation & Test in Europe Conf. & Exhibition (DATE): 1012-1017, 2012.

[112] C. Herder, M. D. Yu, F. Koushanfar, S. Devadas, "Physical unclonable functions and applications: A tutorial," Proc. IEEE. Electr. Electron. Eng., 102(8): 1126-1141, 2014.

[113] T. F. Lee, W. Y. Chen, "Lightweight fog computing-based authentication protocols using physically unclonable functions for internet of medical things," J. Inf. Secur. Appl., 59: 102817, 2021.

[114] A. P. Fournaris, N. Sklavos, "Secure embedded system hardware design – A flexible security and trust enhanced approach," Comput. Electr. Eng., 40(1): 121-133, 2014.

[115] U. Rührmair, D. E. Holcomb, "PUFs at a glance," in 2014 Design, Automation & Test in Europe Conf. & Exhibition (DATE): 1-6, 2014.

[116] C. Helfmeier, C. Boit, D. Nedospasov, J. P. Seifert, "Cloning physically unclonable functions," in 2013 IEEE Int. Symposium on Hardware-Oriented Security and Trust (HOST): 1-6, 2013.

[117] U. Rührmair, J. Sölter, "PUF modeling attacks: An Introduction and overview," in Proc. 2014 Design, Automation & Test in Europe Conf. & Exhibition (DATE): 1-6, 2014.

[118] J. Delvaux, I. Verbauwhede, "Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise," in Proc. 2013 IEEE Int. Symposium on Hardware-Oriented Security and Trust (HOST): 137-142, 2013.

[119] A. Mahmoud, U. Rührmair, M. Majzoobi, F. Koushanfar, "Combined modeling and side channel attacks on strong PUFs," Cryptology ePrint Archive, 2013.

[120] Y. Gao, S. F. Al-Sarawi, D. Abbott, "Physical unclonable functions," Nat. Electron., 3(2): 81-91, 2020.

[121] Y. G. Yang, J. R. Dong, Y. L. Yang, Y. H. Zhou, W. M. Shi, "Usefulness of decoherence in quantum-walk-based hash function," Int. J. Theor. Phys., 60(3): 1025-1037, 2021.

[122] M. Ziiatdinov, "Quantum Hashing. Group approach," Lobachevskii J. Math., 37(2): 222-226, 2016.

[123] H. Buhrman, R. Cleve, J. Watrous, R. de Wolf, "Quantum fingerprinting," Phys. Rev. Lett., 87(16), 2001.

[124] F. Ablayev, A. Vasiliev, "Quantum hashing," Cornell University Library, arXiv:1310.4922 [quant-ph], 2013.

[125] F. Ablayev, M. Ablayev, "Quantum hashing via ε-Universal hashing constructions and freivalds' fingerprinting schemas," in Proc. 16th Int. Workshop on Descriptional Complexity of Formal Systems: 42-52, 2014.

[126] F. Ablayev, M. Ablayev, A. Vasiliev, "On the balanced quantum hashing," J. Phys. Conf. Ser., 681(1): 012019, 2016.

[127] L. Ren, S. Devadas, "Bandwidth hard functions for asic resistance," in Proc. Theory of Cryptography Conf.: 466-492, 2017.

[128] C. Percival, Stronger key derivation via sequential memory-hard functions. 1-16, 2009. Last accessed: Jan. 10, 2023.

[129] C. Percival, S. Josefsson, The scrypt password-based key derivation function. Internet Engineering Task Force (IETF), No. rfc7914, Aug. 2016.

[130] X. Li, Z. Shao, M. Zhu, J. Yang, Fundamentals of optical computing technology: forward the next generation supercomputer. Singapore: Springer, 2018.

[131] W. He, X. Peng, "Optical one-way hash function," In: Advanced Secure Optical Image Processing for Communications, Al Falou, A. ed. IOP Publishing. 2018.

[132] W. Q. He, J. Y. Chen, L. B. Zhang, D. J. Lu, M.-H. Liao, X. Peng, "Optical Hash function based on multiple scattering media," Acta Physica Sinica, 70(5): 054203, 2021.

[133] A. Kumar, A. Fatima, N. K. Nishchal, "An optical Hash function construction based on equal modulus decomposition for authentication verification," Opt. Commun., 428: 7-14, 2018.

[134] L. D. Tsobdjou, S. Pierre, A. Quintero, "A new mutual authentication and key agreement protocol for mobile client—server environment," IEEE trans. netw. serv. manag., 18(2): 1275-1286, 2021.

[135] P. Kocher, J. Jaffe, B. Jun, P. Rohatgi, "Introduction to differential power analysis," J. Cryptogr. Eng., 1(1): 5-27, 2011.

[136] S. N. Srirama, M. Jarke, W. Prinz, "Mobile web services mediation framework," in Proc. the 2nd workshop on Middleware for service oriented computing: held at the ACM/IFIP/USENIX Int. Middleware Conf.: 6-11, 2007.

[137] M. Asif, S. Majumdar, R. Dragnea, "Partitioning the WS execution environment for hosting mobile web services," in Proc. 2008 IEEE Int. Conf. on Services Computing, vol. 2: 315-322, 2008.

[138] J. Li, W. Zhang, V. Dabra, K. K. R. Choo, S. Kumari, D. Hogrefe, "AEP-PPA: An anonymous, efficient and provably-secure privacy-preserving authentication protocol for mobile services in smart cities," J. Netw. Comput. Appl., 134: 52-61, 2019.

[139] J. Kong, F. Koushanfar, "Processor-based strong physical unclonable functions with aging-based response tuning," IEEE Trans. Emerg. Top. Comput., 2(1): 16-29, 2014.

[140] J. P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," in Proc. 11th Int. Conf. on Applied Cryptography and Network Security: 119-135, 2013.

[141] R. Chakraborty, A. Ghosh, V. E. Balas, A. A Elngar, Blockchain: Principles and Applications in IoT. Boca Raton: Chapman and Hall/CRC, 2022.

[142] C. Dobraunig, M. Eichlseder, F. Mendel, M. Schläffer, "Ascon v1.2: Lightweight Authenticated Encryption and Hashing," J. Cryptol, 34(3), 2021.

[143] P. Podimatas, K. Limniotis, "Evaluating the Performance of Lightweight Ciphers in Constrained Environments-The Case of Saturnin," Signals, 3(1): 86-94, 2022.

[144] S. Blanc, A. Lahmadi, K. Le Gouguec, M. Minier, L. Sleem, "Benchmarking of lightweight cryptographic algorithms for wireless IoT networks," Wirel. netw, 28(8): 3453-3476, 2022.

[145] W. K. Lee, K. Jang, G. Song, H. Kim, S. O. Hwang, H. Seo, "Efficient Implementation of lightweight hash functions on GPU and quantum computers for IoT applications," IEEE Access, 10: 59661-59674, 2022.

[146] J. E. Midwinter, Photonics in Switching, 1st ed. Academic Press, 1993.

Biographies

Behrouz Sefid-Dashti received his B.S. and M.S. degrees in computer engineering from the Iran Information Technology Development and Islamic Azad University (Tehran North Branch), respectively. He has over 18+ years of experience in the field of software development, and is currently a Ph.D. candidate of software engineering at the University of Kashan. His career and experience include software analysis, design and architecture, and blockchain development. His research interests include blockchain, software architecture, and cryptography.

- Email: b.sefiddashti@grad.kashanu.ac.ir
- ORCID: 0000-0002-3767-4303
- Web of Science Researcher ID: HJP-5663-2023
- Scopus Author ID: 56123693700
- Homepage: NA

Javad Salimi Sartakhti is an assistant professor of artificial intelligence in the department of computer engineering at the University of Kashan, Iran. He obtained his B.Sc. degree in computer engineering from the University of Kashan and his M.Sc. degree in software engineering from the Tarbiat Modares University, Tehran, Iran, in 2008 and 2013, respectively. In January 2017, he obtained his Ph.D. degree in artificial intelligence at the Isfahan University of Technology. He ranked first among students of computer engineering in all three degrees. His main research interests are mechanism design and game theory, blockchain, machine learning, and Deep learning.

- Email: salimi@kashanu.ac.ir
- ORCID: 0000-0003-1183-1232
- Web of Science Researcher ID: HJY-2812-2023
- Scopus Author ID: 51864592100
- Homepage: https://faculty.kashanu.ac.ir/salimi/en

Hassan Daghigh is an associate professor of mathematics at the University of Kashan, Iran. He received his M.Sc. in mathematics (Commutative Algebra) at the University of Tarbiat Modarres, Iran. He got his Ph.D. degree in mathematics (elliptic curves) at McGill university, Canada in 1998, under the direction of Henri Darmon. His research activities are now focused on number theory (elliptic curves, algebraic number theory) and applications in cryptography in particular lattice and isogeny based cryptography.

- Email: hassan@kashanu.ac.ir
- ORCID: 0000-0002-4242-769X
- Web of Science Researcher ID: HJY-1325-2023
- Scopus Author ID: NA
- Homepage: https://faculty.kashanu.ac.ir/daghigh/en

How to cite this paper:
B. Sefid-Dashti, H. Daghigh, J. S. Sartakhti, "Brand new categories of cryptographic hash functions: A survey," J. Electr. Comput. Eng. Innovations, 11(2): 335-354, 2023.

DOI: 10.22061/jecei.2023.9271.598

URL: https://jecei.sru.ac.ir/article_1840.html