A Comprehensive Review on Broken Hashing Algorithms

Alireza Sadeghi-Nasab, Vahid Rafe
Computer Engineering Group, Faculty of Engineering, Arak University, Arak, Iran
sadeghinasab.alireza@gmail.com, v-rafe@araku.ac.ir

Abstract

Hash functions, which were originally designed for use in a few cryptographic schemes with specific security needs, have since become regular fare for many developers and protocol designers, who regard them as black boxes with magical characteristics. Message digesting, password verification, data structures, compiler operation and linking file name and path together are contemporary examples of hash functions applications. Since 2004, we've observed an exponential increase in the number and power of attacks against standard hash algorithms. In this paper, we investigated well known broken hashing algorithms. A hash function is said to be broken when an attack is found, which, by exploiting special details of how the hash function operates, finds a preimage, a second preimage or a collision faster than the corresponding generic attack. We collected information on all broken hashing algorithms, including their features, applications and attacks. To increase background knowledge, we also provide a summary of the types of attacks in this area.

Keywords: Broken hashing algorithms; cryptography; collision attack; preimage attack.

1. Introduction

Hashing is the process of changing a sequence of characters (string) known as the key into a usually shorter fixed-length sequence of characters known as the hash value in computer science. The input is used as the key in a hash function, and the result is the hash value, or hash for short. Simply, the major benefit and usage of hashing is to save time. Instead of searching a database for the complete original string, for example, only the hash value must be examined, reducing the number of resources required for a search query [1]. The most fundamental concept to grasp when it comes to hashing is that the hash function should, in theory, map each key to a unique hash result. In other words, each string can only match one hash, therefore if you use the same input to the hash function numerous times, the result will always be the same. Furthermore, a key should not have the hash of another key as a result [2]. Because there are so many different hash functions for different purposes with varied inputs and outputs, every method must have the same goal as the requirement mentioned above. As a result, while a given hashing method may perform well for database operations, it is unlikely to work well for error checking or cryptography [3]. Applications of hash functions can be listed as follows:

- File verification: The verification of message integrity is an essential use of secure hashes. Message digests (hash digests over the message) calculated before and after transmission can be used to determine whether the message or file has been modified. Malicious alterations to the file are detected using a cryptographic hash and a chain of trust. Non-cryptographic error-detecting codes like cyclic redundancy checks only protect the file from non-malicious changes, as a purposeful spoof may easily be created with the conflicting code value [4].
- Digital signature: Almost all digital signature systems necessitate the calculation of a cryptographic hash over the message. This enables the signature calculation to be done on the hash digest's relatively short, statically sized size. If the signature verification succeeds over the message with the signature and recalculated hash digest, the message is regarded legitimate. As a result, the cryptographic hash's message integrity attribute is used to develop secure and efficient digital signature techniques [5].
- Password verification: Cryptographic hashes are often used to verify passwords. If the password file is compromised, storing all user credentials in cleartext can result in a catastrophic security breach. Only storing the hash digest of each password is one method to mitigate this risk. To authenticate a user, the user's password is hashed and compared to a previously saved hash. When password hashing is implemented, a password reset procedure is required; original passwords cannot be regenerated using the stored hash value. Due to the fact that standard cryptographic hash functions are meant to be computed quickly, it is easy to try guessed passwords at a high pace. Every second, common graphics processing units may try billions of different passwords. A big random, non-secret salt value that can be kept with the password hash is required for a password hash. The salt randomizes the password hash output, making it impossible for an adversary to keep tables of passwords and precomputed hash values with which to compare the password hash digest. A cryptographic key can be created from the result of a password hash function. As a result, password hashes are also referred to as password-based key derivation functions (PBKDFs) [6].
- Proof of work: A proof-of-work system (or protocol, or function) is an economic mechanism that requires some labor from the service requester, usually in the form of computer processing time, to dissuade denial-of-service attacks and other service abuses such as spam on a

network. The asymmetry of these systems is a critical feature: the requester's effort must be relatively difficult (but possible), while the service provider's labor must be simple to verify. Partially hashed inversions are employed in one common method – utilized in Bitcoin mining and Hashcash – to prove that work was done, to unlock a mining reward in Bitcoin, and as a good-will token to send an e-mail in Hashcash. The sender must locate a message with a hash value that starts with a number of zero bits. The average amount of effort required by the sender to find a valid message is proportional to the number of zero bits in the hash value, whereas the recipient can check the message's correctness by executing a single hash function. In Hashcash, for example, a sender is required to create a header with a 160-bit SHA-1 hash value with the first 20 bits set to zero. The sender will have to try 2^{19} times on average to find a proper header [7].

- File or data identifier: Several source code management systems, such as Git, Mercurial, and Monotone, use the sha1sum of various sorts of content (file content, directory trees, ancestry information, and so on) to uniquely identify them. On peer-to-peer filesharing networks, hashes are used to identify files. An MD4-variant hash is paired with the file size in an ed2k link, for example, providing enough information for discovering file sources, downloading the file, and confirming its contents. Another example is magnet links. These file hashes are frequently the top hash of a hash list or tree, allowing for further benefits. One of the most common uses of a hash function is to look up data in a hash table quickly. Cryptographic hash functions are a special type of hash function that lends itself nicely to this application. Cryptographic hash functions, on the other hand, are far more computationally expensive than ordinary hash functions. As a result, they're most commonly utilized in situations where consumers need to protect themselves from forging (the production of data with the same digest as the expected data) by possibly hostile actors [8].

Although there is a huge list of cryptographic hash functions, many of them have been shown to be susceptible and should be avoided. Even if a hash function has never been cracked, a successful attack on a weakened variation may cause experts to lose faith [9].

The rest of this paper is organized as follows. In section 2, we review hashing algorithms construction methods. In section 3, well known types of attacks are reviewed. In section 4 which is the main part of our paper, all known hash algorithms that were attacked were investigated. Finally, the paper is concluded in section 6.

2. Construction of Hashing Algorithms

Cryptographic hash functions convert arbitrary (or very long) input strings into short, fixed-length output strings. Diffie and Hellman highlighted the necessity for a one-way hash function as a building element of a digital signature system in their landmark paper on public-key cryptography from 1976 [10]. The late 1970s work of [11], [12], and [13] provided the first definitions, analyses, and constructions for cryptographic hash functions.

The requirement for a fast and secure hash function was well acknowledged in the 1980s. A considerable number of designs were made in the late 1980s and early 1990s; about 50 concepts were known in 1993, but at least two-thirds of them were broken. Only a few of those early schemes are still secure after fifteen years of cryptanalysis [14].

The status of the three main groups of hash functions is then discussed: block cipher hash functions, modular arithmetic hash functions, and dedicated hash functions.

2.1. *Hash Functions Based on Block Ciphers*

The initial implementations of hash functions were all based on block ciphers, specifically DES [15]. This approach has numerous advantages: a block cipher's design and evaluation effort can be reused, and very compact implementations can be obtained. However, it's possible that a block cipher includes key scheduling flaws that have just a minor influence on encryption but are undesirable when employed in a hash function. The weak keys in DES [16] and the key schedule flaws in AES-192 [17] and AES-256 [18] are two examples.

Because most block ciphers have a block length of 64 or 128 bits, which is clearly insufficient for collision resistance, the challenging problem is how to design hash functions with a result that is greater than the block length. This subject has proven to be quite tough; significant progress has been done in terms of cryptanalysis and design [19] [20]. It is fair to say that our grasp of how to develop hash functions from simple building blocks is improving; yet, it is unclear if the most efficient hash functions can be designed by starting with a block cipher [21].

2.2. *Hash Functions Based on Arithmetic Primitives*

Hash function structures have also drawn inspiration from public key cryptography, particularly modular arithmetic. As a result, hash functions with a security proof based on number theoretic assumptions like factoring and discrete logarithm have been developed. The performance of schemes based on additive or multiplicative knapsacks is appealing. Despite the theoretical basis, however, real constructs have not fared well up to this point [22].

2.3. *Dedicated Hash Functions*

The constraints of hash functions based on block ciphers necessitated a number of new designs. These hash functions were among the first to be designed for use in software rather than hardware implementations on microprocessors [23].

Dedicated hash functions are built from the ground up with the goal of hashing plain text with optimal efficiency while avoiding the use of existing system components like block ciphers and modular arithmetic. Hard issues like factorization and discrete logarithms aren't used to create these hash functions. A serial sequential repetition of a small step function

is the most typical approach of building compression functions for dedicated hash functions [24].

3. Attacks on Hashing Algorithms

Since hash algorithms are often used in critical capabilities, they have always been attacked. In this section, we will discuss the types of these attacks.

3.1. *Collision Attack*

A Hash Collision Attack aims to locate two hash function input strings that generate the same hash result. Because hash functions have an indefinite input length and a predefined output length, it's unavoidable that two different inputs will create the same output hash. A collision occurs when two different inputs give the same hash output. Any program that compares two hashes together, such as password hashes, file integrity checks, and so on, can take advantage of this collision. Of course, the chances of a collision are slim, especially for functions with huge output values. The ability to brute force hash collisions becomes more and more realistic as available processing power grows [25].

In practice, there are various ways to take advantage of a hash collision. If the attacker was giving a file download and displaying the hash to guarantee the file's integrity, he might substitute a new file with the same hash and the person downloading it would have no way of knowing the difference. Because it contains the same hash as the supposed actual file, the file appears to be valid [26].

3.2. *Preimage Attack*

A preimage attack on cryptographic hash functions aims to find a message with a certain hash value in cryptography. A cryptographic hash function's preimage should be resistant to attacks (set of possible inputs).

There are two types of preimage resistance in the context of an attack:

- Preimage resistance: it is computationally infeasible to discover any input that hashes to any pre-specified output; that is, given y, it is difficult to find an x such that h(x) = y.
- Second-preimage resistance: finding another input that produces the same output for a given input is computationally infeasible; i.e., given x, finding a second input x' x such that h(x) = h(x') is tough [27].

These can be compared to collision resistance, which states that finding any two separate inputs x, x' that hash to the same output (i.e., h(x) = h(x') is computationally impossible. Collision resistance entails second-preimage resistance, but it does not imply preimage resistance. A collision attack, on the other hand, is implied by a second-preimage attack (due to the fact that, in addition to x', x is already known) [28].

An ideal hash function is one in which a brute-force attack is the fastest way to compute the first or second preimage. This attack has a temporal complexity of 2^n for an n-bit hash, which is too high for a typical output size of n = 128 bits. If an opponent can only attain this level of complexity, the hash function is deemed preimage-resistant. However, quantum computers undertake a structured preimage attack in $\sqrt{2^n} = 2^{\frac{n}{2}}$, implying a second preimage and consequently a collision attack [29]. By cryptanalyzing particular hash functions, faster preimage attacks can be discovered, and they are unique to that function. Although several substantial preimage attacks have been developed, they are still not feasible. Many Internet protocols would be severely harmed if a practical preimage attack was uncovered. In this situation, "realistic" means that an attacker with a fair number of resources could carry it out. A preimaging attack that costs trillions of dollars and takes decades to preimage one desired hash value or message, for example, is not realistic; one that costs a few thousand dollars and takes a few weeks could be [30].

3.3. *Birthday Attack*

A birthday attack is a form of cryptographic attack that takes advantage of the probability theory mathematics underpinning the birthday problem. This attack can be used to manipulate two or more parties' communication. The attack is based on the increased frequency of collisions discovered between random attack attempts and a fixed number of permutations (pigeonholes). It is possible to find a collision of a hash function in $\sqrt{2^n}$ with a birthday attack, with 2^n being the standard preimage resistant security. In $\sqrt[3]{2^n}$, quantum computers can undertake birthday attacks, thus bypassing collision resistance, according to a general result [31].

A birthday attack on digital signatures is possible. Typically, a message m is signed by computing f(m), where f is a cryptographic hash function, and then signing using a secret key (m). Let's say Mallory wants to dupe Bob into signing a phony contract. Mallory drafts a legitimate contract m and a fictitious contract m'. She then looks for places where m can be altered without changing the meaning, such as commas, blank lines, one vs two spaces after a sentence, synonym replacement, and so on. She may build a tremendous number of permutations on m by mixing these adjustments, all of which are fair contracts. Mallory builds a large number of variations on the fraudulent contract m' in a similar fashion. She then applies the hash function to all of these permutations until she discovers a fair contract version and a fraudulent contract version with the same hash value, f(m) = f(m'). She hands Bob the fair version for him to sign. Mallory grabs Bob's signature and attaches it to the forged contract after he signs. This signature "proves" Bob's involvement in the phony contract [32].

3.4. *Boomerang Attack*

The boomerang attack is a cryptanalysis approach for block ciphers based on differential cryptanalysis in cryptography. David Wagner released the attack in 1999, and it was used to break the COCONUT98 encryption. The boomerang attack has opened up new attack pathways for many ciphers previously thought to be immune to differential cryptanalysis. Differential cryptanalysis is used in the boomerang attack. In differential cryptanalysis, an attacker takes advantage of how differences in the plaintext input to a cipher might alter the difference at the output (the ciphertext). All, or almost all, of the cipher must be covered by a high-probability "differential" (that is, an input difference that will yield a likely output difference). The

boomerang attack allows differentials that only cover a portion of the cipher to be used [33].

3.5. *Rebound Attack*

The rebound attack is a cryptographic hash function cryptanalysis technique. Florian Mendel, Christian Rechberger, Martin Schläffer, and Sren Thomsen initially reported the attack in 2009. The Rebound Attack is a statistical attack on hash functions that uses rotational and differential cryptanalysis techniques to identify collisions and other interesting characteristics. The attack's main concept is to look for a specific differential characteristic in a block cipher (or a portion of one), a permutation, or another sort of primitive. the rebound attack consists of 2 phases:

1. The inbound (or match-in-the-middle) phase encompasses the portion of the differential characteristic that is difficult to satisfy probabilistically. The goal is to identify a large number of solutions with a low average complexity for this component of the feature. To do so, the set of equations that describes the characteristic in this phase should be underdetermined. When looking for a solution, there are many degrees of freedom available, resulting in a wide range of options. The inbound phase can be performed numerous times in order to get a large enough number of beginning locations for the outbound phase to succeed.
2. Each solution from the inbound phase is propagated outwards in both directions in the outbound phase, with the characteristic being checked in both directions. The characteristic's probability in the outbound phase should be as high as possible.

The capacity to efficiently calculate the problematic elements of the differential characteristic in the inbound phase is a benefit of using an inbound and two outbound phases. It also ensures a high possibility of success in the outbound phase. As a result, the total likelihood of discovering a differential feature is increased when compared to typical differential approaches.

The inbound phase will often start with a small number of active state bytes (bytes with non-zero differences), then propagate to a large number of active bytes in the middle of the round, before returning to a low number of active bytes at the end. The concept is to have a large number of active bytes at an S-input boxes and output in the middle of the phase. Characteristics may therefore be determined quickly by selecting values for the differences at the beginning and end of the inbound phase, propagating them to the middle, and looking for matching in the S-input boxes and output. This can usually be done row- or column-wise for AES-like ciphers, making the method rather efficient. In the incoming phase, differing starting and ending values result in a variety of differential features.

The purpose of the outbound phase is to assess whether the intended characteristics are followed by propagating the characteristics discovered in the inbound phase backwards and forwards. Truncated differentials are commonly employed in this case since they provide higher probabilities and the particular values of the differences are unimportant for the purpose of identifying a collision. The number of active bytes in the characteristic and how they are ordered in the characteristic determine the likelihood of the characteristic following the desired pattern of the outgoing phase. It is not sufficient for the differentials in the outbound phase to be of a specified type to achieve a collision. Any active bytes at the beginning and end of the characteristic must have a value that cancels any feed-forward operation. As a result, any number of active bytes at the start and conclusion of the outgoing phase should be in the same location while creating the characteristic. The probability of these bytes canceling contributes to the outbound characteristic's probability [34].

3.6. *Length Extension Attack*

A length extension attack is a form of attack in cryptography and computer security in which an attacker can calculate Hash($message_1$ ‖ $message_2$) for an attacker-controlled $message_2$ using Hash($message_1$) and the length of $message_1$, without knowing the content of $message_1$. The majority of algorithms based on the Merkle–Damgrd structure are vulnerable to this type of attack. A length extension attack allows anyone to include extra information at the end of the message and produce a valid hash without knowing the secret when a Merkle–Damgrd based hash is used as a message authentication code with construction H(secret ‖ message) and message and the length of secret is known. HMAC hashes are not vulnerable to length extension attacks because they do not use this architecture [35].

The vulnerable hashing functions work by transforming an internal state using the input message. The hash digest is formed by outputting the internal state of the function after all of the input has been processed. The internal state can be reconstructed from the hash digest, which can then be utilized to process fresh data. In this approach, the message can be extended and the hash that is a valid signature for the new message can be computed [36].

4. Broken Hashing Algorithms

In this section, we discuss all broken hash algorithms. We first give a summary of their features and functions and then explain how they are broken. It should be noted that here, the algorithms are listed in order of their publish date.

4.1. *MD2*

The MD2 Message-Digest Algorithm was created by Ronald Rivest in 1989 as a cryptographic hash function [37]. The technique is designed for computers with an 8-bit processor. RFC 1319 of the Internet Engineering Task Force specifies MD2 [38]. Despite the fact that MD2 has not yet been totally penetrated, the IETF designated it as "historic" in 2011, noting "signs of weakness." In favor of SHA-256 and other robust hashing algorithms, it has been deprecated. Nonetheless, as of 2014, it was still being used in public key infrastructures as part of MD2 and RSA certificates [39].

Collisions of MD2's compression function were disclosed by Rogier and Chauvaud (1997), albeit they were unable to extend the attack to the entire MD2 [40]. MD2 was discovered to be vulnerable to a preimage attack with a time complexity comparable to 2^{104} compression function operations in 2004.

"MD2 can no longer be called a secure one-way hash algorithm," writes the author [37]. MD2 improved on a preimage attack in 2008, with a time complexity of 2^{73} compression function evaluations and 2^{73} message blocks in memory. MD2 was discovered to be vulnerable to a collision attack in 2009, requiring $2^{63.5}$ compression function executions and 2^{52} hash values in memory. The birthday attack, which is estimated to require $2^{62.5}$ compression function evaluations, is marginally better [41]. OpenSSL, GnuTLS, and Network Security Services all received security patches in 2009 that disabled MD2 [42].

4.2. *Snefru*

Ralph Merkle devised Snefru, a cryptographic hash function, in 1990. The function can output data in 128-bit and 256-bit formats. It was named after Egyptian Pharaoh Sneferu, carrying on the Khufu and Khafre block ciphers' legacy [43].

Eli Biham and Adi Shamir were able to employ differential cryptanalysis to uncover hash collisions, proving that Snefru's initial architecture was insecure. After then, the design was tweaked by increasing the number of iterations in the algorithm's main pass from two to eight. Although differential cryptanalysis can crack the improved version with less complexity than brute force search (a certification flaw), the attack requires $2^{88.5}$ operations, making it impractical in reality at the moment [44].

4.3. *MD4*

Ronald Rivest created the MD4 Message-Digest Algorithm in 1990. A 128-bit digest is used. Later designs, like the MD5, SHA-1, and RIPEMD algorithms, were influenced by the method [45]. On Microsoft Windows NT, XP, Vista, 7, 8, and 10, MD4 is used to create NTLM password-derived key digests [46].

In a paper published in 1991, Den Boer and Bosselaers revealed MD4's flaws [47]. Hans Dobbertin discovered the first full-round MD4 collision attack in 1995, which took only seconds to execute at the time [48]. Wang et al. discovered an extremely effective collision attack, as well as attacks on later hash function designs in the MD4/MD5/SHA-1/RIPEMD family, in August 2004. Later work by Sasaki et al. enhanced this result, and today producing a collision is as cheap as validating it (a few microseconds) [49]. Gatan Leurent also cracked MD4's preimage resistance in 2008, using a 2^{102} attack [50]. Guo et al. disclosed a $2^{99.7}$ attack in 2010 [51]. RFC 6150 stated in 2011 that RFC 1320 (MD4) is historic (obsolete). Figure 1 shows a collision example of MD4:

```
k1 = 839c7a4d7a92cb5678a5d5b9eea5a757
3c8a74deb366c3dc20a083b69f5d2a3bb3719
dc69891e9f95e809fd7e8b23ba6318edd45e5
1fe39708bf9427e9c3e8b9

k2 = 839c7a4d7a92cbd678a5d529eea5a757
3c8a74deb366c3dc20a083b69f5d2a3bb3719
dc69891e9f95e809fd7e8b23ba6318edc45e5
1fe39708bf9427e9c3e8b9

k1 ≠ k2, but MD4(k1) = MD4(k2) = 4d7e6a1defa93d2dde05b45d864c429b
```

Figure 1. MD4 collision example

4.4. *MD5*

Ronald Rivest devised and implemented MD5, a cryptographic hash function, in 1992 with the goal of upgrading MD4 after the algorithm was severely hacked [52]. MD5 and MD4 are part of a sequence of message digest algorithms in which the successors were conceived and developed to replace the predecessors. The algorithm's output specification is a 32-digit hexadecimal number that is a 128-bit (16-byte) hash value. The technique is mostly based on 32-bit integers and includes addition and bitwise operations including XOR, OR, AND, bitwise rotation, and Add (mod 232). MD5 was once one of the most widely used hash functions; however, due to a main flaw, the algorithm is no longer suitable for use in applications such as cryptographic ones [53].

In 1993, B. den Boer and A. Bosselaers discovered a "pseudo-collision" consisting of the identical message with two different sets of beginning values a year after MD5 was created, revealing a flaw in the algorithm [54]. Despite this, no hard evidence of collisions was discovered until 2004. Xiaoyun Wang and Hongbo Yu devised a specially constructed attack known as modular differential, which got MD5 collisions in 15 minutes to an hour of computing time [55]. This attack isn't limited to MD5, but it can also be used to break other functions like HAVAL-128, SHA-0, and RIPEMD. The attack is complicated in general because it necessitates a thorough understanding of the inner mechanics of MD5's algorithm. Regardless, the results showed that MD5 has two pairs of collisions, confirming its vulnerability to collisions, which turned out to be very plausible. Many MD5 collisions have been discovered since then, like the one depicted in Figure 2, in which the two message blocks hash to the same value of *79054025255fb1a26e4bc422aef54eb4*.

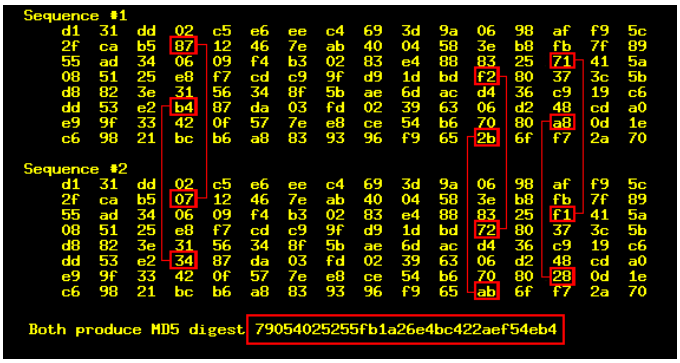

Figure 2. MD5 collision example

4.5. *RIPEMD*

The RIPEMD (RIPE Message Digest) family of cryptographic hash functions was created in 1992 (the first RIPEMD) and 1996 (the second RIPEMD) (other variants). RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, and RIPEMD-320 are the five functions in the family, with RIPEMD-160 being the most frequent. The original RIPEMD, as well as RIPEMD-128, are not regarded secure due to the 128-bit result being too small, as well as design flaws (for the original RIPEMD). The RIPEMD 256-bit and 320-bit variants give the same level of security as RIPEMD-128 and RIPEMD-160, respectively; they are meant for applications where the security

level is appropriate but a longer hash result is required. While RIPEMD functions are less well-known than SHA-1 and SHA-2, they are utilized in Bitcoin and other Bitcoin-based coins [56].

In August 2004, a collision was reported for the original RIPEMD [57]. This does not apply to RIPEMD-160 [58].

4.6. *HAVAL*

HAVAL is a hash function for cryptography. HAVAL, unlike MD5, can generate hashes of various lengths: 128 bits, 160 bits, 192 bits, 224 bits, and 256 bits. The number of rounds (3, 4, or 5) utilized to compute the hash can also be specified with HAVAL. Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry invented this algorithm in 1992. HAVAL hashes (also known as fingerprints) are usually 32-, 40-, 48-, 56-, or 64-digit hexadecimal values. The following shows a 43-byte ASCII input and the HAVAL hash that corresponds (256 bits, 5 passes) [59].

The usage of HAVAL (at least the variation with 128 bits and three passes with 2^6 operations) is now called into doubt due to flaws discovered during research. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu reported collisions for HAVAL (128 bits, 3 passes) on August 17, 2004 [57].

4.7. *SHA-0*

The Secure Hash Algorithms are a set of cryptographic hash functions published as a U.S. Federal Information Processing Standard (FIPS) by the National Institute of Standards and Technology (NIST). The original version of the 160-bit hash function, known as "SHA," was released in 1993 and was given the moniker "SHA-0." It was pulled soon after publishing due to a disclosed "significant fault" and replaced by the slightly altered SHA-1 version [23].

Florent Chabaud and Antoine Joux, two French researchers, presented an attack on SHA-0 at CRYPTO 98: collisions may be identified with a complexity of 2^{61}, which is lower than the 2^{80} for an ideal hash function of the same size [60]. Biham and Chen discovered SHA-0 near-collisions in 2004. two messages that hash to virtually the same result. in this case, 142 out of 160 bits are equal. They also discovered that entire SHA-0 collisions were decreased to 62 out of 80 cycles [61]. Joux, Carribault, Lemuet, and Jalby then announced a collision for the complete SHA-0 algorithm on August 12, 2004. A generalization of the Chabaud and Joux attack was used to accomplish this. On a supercomputer with 256 Itanium 2 processors, finding the collision had a complexity of 2^{51} and required roughly 80,000 processor hours (equivalent to 13 days of full-time use of the computer) [62]. Wang, Feng, Lai, and Yu presented preliminary results of an attack against MD5, SHA-0, and other hash algorithms at the CRYPTO 2004 Rump Session on August 17, 2004. Their SHA-0 attack has a complexity of 2^{40}, which is much better than Joux et al [57]. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu announced an attack in February 2005 that could identify collisions in SHA-0 in 2^{39} operations [63]. Another attempt using the boomerang attack in 2008 reduced the difficulty of discovering collisions to $2^{33.6}$, which was expected to take 1 hour on an ordinary PC in 2008 [64].

4.8. *GOST*

The GOST hash function is a 256-bit cryptographic hash function established in the standards *GOST R 34.11-94* and *GOST 34.311-95*. It was first defined in *GOST R 34.11-94* Information Technology – Cryptographic Information Security – Hash Function, a Russian national standard. GOST converts a variable-length message into a 256-bit fixed-length output. The input message is divided into 256-bit blocks (eight 32-bit little endian integers) and padded with as many zeros as are required to get the message length up to 256 bits. The remaining bits are filled with a 256-bit integer reflecting the length of the original message in bits, followed by a 256-bit integer representing the arithmetic total of all previously hashed blocks [65].

The full-round GOST hash algorithm was broken by an attack disclosed in 2008. A collision attack in 2^{105} time, as well as first and second preimage attacks in 2^{192} time, are presented in this study (2^n time refers to the approximate number of times the algorithm was calculated in the attack) [66].

4.9. *SHA-1*

The National Security Agency (NSA) developed secure hash algorithm 1 (SHA-1) in 1995, based on SHA-0, and the National Institute of Standards and Technology (NIST) released it as a Federal Processing Standard in 1996. SHA-1 is a member of the SHA family (SHA-0, SHA-1, SHA-2, SHA-3) developed by the National Security Agency (NSA) and adopted by several government platforms following its release. SHA-1 produces a 160-bit (20-byte) hash value that accepts communications with less than 2^{64} bits as input. The hash value is usually represented as a 40-digit hexadecimal number. The design ideas of MD4 and MD5 were used to create SHA-1. They are the same as MD5 in terms of operations employed in the function [67].

Unlike MD5, SHA-1 has no known clashes with message blocks or passwords as of now. NIST, on the other hand, stopped using SHA-1 in 2005 because cryptanalysts discovered flaws in the function's architecture, indicating that collisions might be found with fewer computations than a brute force attack. As a result, NIST mandated that SHA-1 be replaced by SHA-2 by 2010. Google and Mozilla recently declared that encrypted SSL certificates with expiration dates after December 31^{st}, 2016 will no longer be trusted by their respective browsers [68] [69]. In compared to MD5, however, the replacement of SHA-1 has not progressed significantly, and its replacement from mainstream applications is still a big work in progress.

Following the disclosure of SHA-1's flaws in 2005, Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu presented research showing that finding a collision in SHA-1 required only 2^{69} operations. The number of processes required would be 2^{80} in contrast to a brute force search [70]. In 2015, a group of people led by Marc Stevens, Pierre Karpman, and Thomas Peyrin demonstrated a SHA-1 collision attack devised by Marc Stevens. With $2^{57.5}$ operations in 2010, Marc Steven claims to have discovered near-working collisions against SHA-1. Using a 16-node cluster and 64 Graphical Processing Units (GPUs), a free-start collision was discovered. An actual collision might be identified for $75,000 to $120,000 US, according to the authors, which is within a criminal organization's budget and the NSA's [71]. The SHAttered attack was announced on February 23,

2017, by the CWI (Centrum Wiskunde & Informatica) and Google, in which they created two separate PDF files with the identical SHA-1 hash in around $2^{63.1}$ SHA-1 evaluations. This approach is 100,000 times faster than using a birthday attack to brute force a SHA-1 collision, which is predicted to take 2^{80} SHA-1 evaluations. The attack requires 6,500 years of single-CPU computations and 110 years of single-GPU computations in order to be successful [72]. A paper presented at Eurocrypt 2019 on April 24th by Gatan Leurent and Thomas Peyrin described an improvement to the previously greatest chosen-prefix attack in Merkle–Damgrd–like digest functions based on Davies–Meyer block ciphers. This approach can now discover chosen-prefix collisions in about 2^{68} SHA-1 evaluations thanks to these enhancements. This is roughly 1 billion times faster (and now usable for many targeted attacks, thanks to the ability to choose a prefix, for example malicious code or faked identities in signed certificates) than the previous attack's $2^{77.1}$ evaluations (but without chosen prefix, which was impractical for most targeted attacks because the found collisions were almost random) and is fast enough to be practical for resourceful attackers, requiring around $100,000 of coding [73]. The authors published an enhanced attack on January 5, 2020. They show a chosen-prefix collision attack with a complexity of $2^{63.4}$ in this work, which would cost 45k USD per created collision at the time of publishing [74].

4.10. *Tiger*

Tiger is a cryptographic hash function created in 1995 by Ross Anderson and Eli Biham for 64-bit platforms. Tiger hash values are 192 bits in length. For compliance with protocols that assume a specific hash size, truncated versions (known as Tiger/128 and Tiger/160) can be utilized. Unlike the SHA-2 family, there are no distinct initialization values; instead, they are just prefixes to the complete Tiger/192 hash value. Tiger is widely used as part of a Merkle hash tree, which is referred to as TTH (Tiger Tree Hash). Many clients on the Direct Connect and Gnutella file sharing networks use TTH, and it can be put in the BitTorrent metafile to improve content availability. Tiger was considered for inclusion in the OpenPGP standard, but RIPEMD-160 was chosen instead [75].

Except for pseudo-near collision, there are no known viable attacks on the complete 24-round Tiger, unlike MD5 or SHA-0/1 [76]. While MD5 processes its state with 64 simple 32-bit operations per 512-bit block and SHA-1 with 80, Tiger processes its state with a total of 144 such operations per 512-bit block, including huge S-box look-ups for added security. John Kelsey and Stefan Lucks discovered a collision-finding attack on 16-round Tiger with a time complexity of roughly 2^{44} compression function invocations, as well as another attack on 20-round Tiger that detects pseudo-near collisions with work less than 2^{48} compression function invocations [77]. Florian Mendel et al. have improved on these attacks by outlining a collision attack that spans 19 Tiger rounds and a pseudo-near-collision attack that spans 22 rounds. These attacks demand work equivalent to around 2^{62} and 2^{44} Tiger compression function evaluations, respectively [78].

4.11. *PANAMA*

PANAMA is a cryptographic primitive that can be used as a hash function or a stream cipher, however its hash function mode has been broken and is no longer appropriate for cryptographic application. Joan Daemen and Craig Clapp designed it and presented it at the Fast Software Encryption (FSE) conference in 1998. The cipher has impacted a number of other schemes, including MUGI and SHA-3 [79].

Collisions were demonstrated as a hash function by Vincent Rijmen et al. in their article. The attack has a computational complexity of 2^{82} and uses very little memory [80]. Joan Daemen and Gilles Van Assche demonstrated at FSE 2007 a practical attack on the Panama hash function that results in a collision after 2^{6} evaluations of the state updating function [81].

4.12. *Whirlpool*

Whirlpool is a hash function for cryptographic data. It was first defined in 2000 by Vincent Rijmen (co-creator of the Advanced Encryption Standard) and Paulo S. L. M. Barreto. Whirlpool is a hash that was inspired by the square block cipher and belongs to the same family of block cipher functions. It's a Miyaguchi-Preneel design based on a significantly altered Advanced Encryption Standard (AES). Whirlpool returns a 512-bit message digest from any message that is less than 2256 bits long [82].

Full collisions against 4.5 rounds in 2^{120} operations, semi-free-start collisions against 5.5 rounds in 2^{120} time, and semi-free-start near-collisions against 7.5 rounds in 2^{128} time were announced in 2009 [83].

4.13. *RadioGatún*

Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche designed RadioGatún, a cryptographic hash primitive. The NIST Second Cryptographic Hash Workshop, held in Santa Barbara, California on August 24–25, 2006, was the first time it was publicly presented as part of the NIST hash function competition. The same team that created RadioGatún went on to improve this cryptographic primitive significantly, resulting in the Keccak SHA-3 algorithm [84].

RadioGatún is a collection of 64 hash functions that are differentiated by a single parameter, the word width in bits (w), which can be set between 1 and 64. The 32-bit and 64-bit RadioGatún variants are the only word sizes having certified test vectors. The algorithm stores its internal state in 58 words, each with w bits, therefore the 32-bit version requires 232 bytes (since each word requires 32 bits or four bytes, and 58 multiplied by four is 232), and the 64-bit version requires 464 bytes (each word using eight bytes) [85]. Although RadioGatún is a variant of PANAMA, when employed as a hash function, it does not share PANAMA's flaws. RadioGatún continues to be a safe hash function. The version of RadioGatún with a word size of two bits is the one that is broken the most. The 32-bit version of RadioGatún has a security strength of 304 bits, whereas the 64-bit version has a security strength of 608 bits. This assertion has not been debunked by the most well-known cryptanalysis: The 32-bit version requires 352 bits of effort, whereas the 64-bit version requires 704 bits of work. RadioGatún can be used as a hash function or a stream cipher, and it can generate an infinite

stream of pseudo-random integers; this type of hash is now known as Extendable-Output Function (XOF) [86].

Dmitry Khovratovich offers two attacks, one with a complexity of 2^{18w} and the other with a complexity of $2^{23.1w}$, that do not break the designers' security claims [87]. Khovratovich also wrote a paper entitled "Cryptanalysis of hash functions with structures," in which he describes a 2^{18w} attack [88]. With the 1-bit version of the technique, Charles Bouillaguet and Pierre-Alain Fouque offer an attack that requires $2^{24.5}$ operations to generate collisions. Because all of the possible trails we knew for the 1-bit version turned out to be impossible to extend to n-bit versions, the attack can't be extended to larger versions. This attack is less effective than the others and does not compromise RadioGatn's security [89]. The most effective attack against the algorithm, devised by Thomas Fuhr and Thomas Peyrin, has a complexity of 2^{11w}. They break the 2-bit (word size of two) version of RadioGatn in the paper. Despite being more effective than the other attacks, this one fails to violate the security claim [90]. The developers of RadioGatún have claimed that their "own experiments did not inspire confidence in RadioGatún" [91].

4.14. *Streebog*

The Russian national standard GOST R 34.11-2012 Information Technology – Cryptographic Information Security – Hash Function defines Streebog as a cryptographic hash function. It was developed to replace an outmoded GOST hash function established in GOST R 34.11-94 and as an asymmetric response to the US National Institute of Standards and Technology's SHA-3 competition. Streebog works with 512-bit input blocks, and uses the Merkle–Damgrd architecture to support inputs of any size. The new hash function's high-level structure is similar to that of GOST R 34.11-94, however the compression function has been drastically altered. The compression mechanism uses a 12-round AES-like encryption with a 512-bit block and 512-bit key in Miyaguchi–Preneel mode. Streebog-256 differs from Streebog-512 in that it uses a different initial state and truncates the output hash, but otherwise is similar [92].

Ma et al. describe a preimage attack that finds a single preimage of GOST-512 reduced to 6 rounds in 2^{496} time and 2^{64} memory or 2^{504} time and 2^{11} memory. In the same publication, they describe a collision attack with a time complexity of 2^{181} and a memory demand of 2^{64} [93]. If the message comprises more than 2^{259} blocks, Guo et al. describe a second preimage attack on full Streebog-512 with a total time complexity corresponding to 2^{266} compression function evaluations [94]. An attack on a modified version of Streebog with different round constants was published by AlTawy and Youssef. While this attack may not have had a direct influence on the original Streebog hash function's security, it did raise questions regarding the provenance of the function's utilized parameters. These are pseudorandom constants generated with a Streebog-like hash function, presented with 12 different natural language input messages, according to the inventors [95]. AlTawy et al discovered a 5-round free-start collision and a 7.75 free-start near collision for the internal cipher with complexities of 2^8 and 2^{40} , respectively, as well as attacks on the compression function with 7.75 round semi free-start collisions with time complexity 2^{184} and memory complexity 2^8, 8.75 and 9.75 round semi free-start near collisions with time complexities of 2^{120} and 2^{196}, respectively [96]. Wang et al. describe a 9.5-round collision attack on the compression function with a time complexity of 2^{176} and a memory complexity of 2^{128} [97]. Biryukov, Perrin, and Udovenko reverse engineered the unpublished S-box generation structure (which had previously been reported to be created randomly) in 2015 and discovered that the underlying components are cryptographically weak [98].

4.15. *Blake2s, Blake2b*

BLAKE is a cryptographic hash function based on Daniel J. Bernstein's ChaCha stream cipher, but before each ChaCha round, a permuted copy of the input block is added, XORed with round constants. There are two variations, similar to SHA-2, that differ in word size. ChaCha is based on a 44-word array. BLAKE truncates the ChaCha result to get the next hash value by combining an 8-word hash value with 16 message words. BLAKE-256 and BLAKE-224 employ 32-bit words to produce 256-bit and 224-bit digests, respectively, whereas BLAKE-512 and BLAKE-384 use 64-bit words to produce 512-bit and 384-bit digests, respectively. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan submitted BLAKE to the NIST hash function competition. There were 51 submissions in 2008. BLAKE advanced to the final round of five candidates in 2012, but lost to Keccak, which was chosen for the SHA-3 algorithm. BLAKE, like SHA-2, is available in two versions: one that uses 32-bit words for hashes up to 256 bits long, and another that employs 64-bit words for hashes up to 512 bits long. Only 8 words (256 or 512 bits) are retained across blocks in the core block transformation, which combines 16 words of input with 16 working variables. It employs a table of 16 constant words (the leading 512 or 1024 bits of the fractional part of π) and a table of 10 16-element permutations [99].

Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein produced BLAKE2, a cryptographic hash function based on BLAKE. The purpose of the invention was to replace the widely used, but faulty, MD5 and SHA-1 algorithms in software applications that required high performance. On December 21, 2012, BLAKE2 was announced. On 64-bit x86-64 and ARM architectures, BLAKE2b outperforms MD5, SHA-1, SHA-2, and SHA-3. BLAKE2 is more secure than SHA-2 and similar to SHA-3 in terms of length extension resistance, indistinguishability from a random oracle, and so on [100]. BLAKE2 removes the addition of constants to message words from the BLAKE round function, changes two rotation constants, simplifies padding, adds an XOR'ed parameter block with initialization vectors, and reduces the number of rounds from 16 to 12 for BLAKE2b (successor of BLAKE-512) and 14 to 10 for BLAKE2s (successor of BLAKE-256). Keying, salting, personalization, and hash tree modes are all supported by BLAKE2, which can output digests ranging from 1 to 64 bytes for BLAKE2b and 32 bytes for BLAKE2s. BLAKE2bp (4-way parallel) and BLAKE2sp (2-way parallel) are parallel variants developed for improved performance on multi-core systems (8-way parallel) [101].

collisions against 2.5 rounds in 2^{112} operations for BLAKE2s and 2.5 rounds in 2^{224} operations for BLAKE2b and near-collisions against 2.5 rounds in 2^{241} operations for

BLAKE2s and 2.5 rounds in 2^{481} operations for BLAKE2b were announced in 2009 [102].

4.16. *Kupyna*

The Ukrainian national standard *DSTU 7564:2014* defines Kupyna as a cryptographic hash function. It was designed to replace an antiquated GOST hash function described in the old standard *GOST 34.11-95*, which was similar to the Russian Streebog hash function. The Davies–Meyer compression function, which is based on the Even–Mansour cipher, is used by the Kupyna hash function. The compression function is made up of four operations: AddRoundConstant, SubBytes, ShiftBytes, and MixColumns, which are borrowed from the Kalyna block cipher. Four separate S-boxes are used in the round function. The function can return a digest of any length between 8 and 512 bits; the Kupyna-n function returns an n-bit digest. 256, 384, and 512 bits are the suggested digest lengths. After four rounds of compression, the designers say that differential and rebound attacks are ineffectual [103].

Based on rebound attacks on Grøstl, Christoph Dobraunig, Maria Eichlseder, and Florian Mendel present a collision attack on Kupyna-256 reduced to 4 rounds with time complexity $2^{67}$and Kupyna-256 reduced to 5 rounds with time complexity 2^{120} [104]. Jian Zou and Le Dong also describe a 5-round Kupyna-256 collision attack with a time complexity of 2120, as well as a pseudo-preimage attack on 6-round Kupyna-256 with time and memory complications of 2^{250} and on 8-round Kupyna-512 with time and memory complexities of 2^{498}. They point out that these attacks pose no harm to Kupyna's security claims [105]. When Kupyna is used for MAC schemes, Onur Duman published differential fault analysis. According to the research, retrieving one byte of the state requires 2.21–2.42 failures [106].

5. Conclusion

In This paper, we studied all of the broken hash algorithms. We began by discussing hash functions and their applications. Then, in order to increase the background knowledge, we explained several well-known attacks in this field, and finally, we listed all of the broken algorithms in order of publication year and described the attacks that performed on them. According to the authors of this paper, choosing a standard and secure hash algorithm is critical, because hashing algorithms are mostly employed in essential and sensitive applications. Reading this article for this decision can be very helpful.

References

[1] W. Penard and T. van Werkhoven, "On the secure hash algorithm family," *Cryptography in context*, pp. 1-18, 2008.

[2] L. Chi and X. Zhu, "Hashing techniques: A survey and taxonomy," *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1-36, 2017.

[3] J. Wang, T. Zhang, N. Sebe, H. T. Shen and others, "A survey on learning to hash," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 769-790, 2017.

[4] J. E. Silva, "An overview of cryptographic hash functions and their uses," *GIAC*, vol. 6, 2003.

[5] C. Dods, N. P. Smart and M. Stam, "Hash based digital signature schemes," in *IMA international conference on cryptography and coding*, Springer, 2005, pp. 96-115.

[6] J. Camenisch, A. Lehmann and G. Neven, "Optimal distributed password verification," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 182-194.

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

[8] J. Loeliger and M. McCullough, Version Control with Git: Powerful tools and techniques for collaborative software development, O'Reilly Media, Inc., 2012.

[9] A. Regenscheid, S. Zhang, J. Kelsey, M. Nandi, S. Paul, R. Perlner and A. Regenscheid, Status report on the first round of the SHA-3 cryptographic hash algorithm competition, Citeseer, 2009.

[10] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Secure communications and asymmetric cryptosystems*, Routledge, 2019, pp. 143-180.

[11] R. C. Merkle, Secrecy, authentication, and public key systems, Stanford university, 1979.

[12] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Inst of Tech Cambridge Lab for Computer Science, 1979.

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

[14] B. Preneel, "Analysis and design of cryptographic hash functions," Citeseer, 1993.

[15] D. E. Standard and others, "Data encryption standard," *Federal Information Processing Standards Publication*, vol. 112, 1992.

[16] J. H. Moore and G. J. Simmons, "Cycle structure of the DES for keys having palindromic (or antipalindromic) sequences of round keys," *IEEE Transactions on Software Engineering*, no. 2, pp. 262-273, 1987.

[17] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich and A. Shamir, "Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2010, pp. 299-319.

[18] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full AES-192 and AES-256," in *International conference on the theory and application of cryptology and information security*, 2008.

[19] L. R. Knudsen, X. Lai and B. Preneel, "Attacks on fast double block length hash functions," *Journal of Cryptology*, vol. 11, no. 1, pp. 59-72, 1998.

[20] J. P. Steinberger, "The collision intractability of MDC-2 in the ideal-cipher model," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2007.

[21] P. Rogaway and J. Steinberger, "Constructing cryptographic hash functions from fixed-key blockciphers," in *Annual International Cryptology Conference*, 2008.

[22] D. X. Charles, K. E. Lauter and E. Z. Goren, "Cryptographic hash functions from expander graphs," *Journal of CRYPTOLOGY*, vol. 22, no. 1, pp. 93-113, 2009.

[23] B. Preneel, "The first 30 years of cryptographic hash functions and the NIST SHA-3 competition," in *Cryptographers' track at the RSA conference*, 2010.

[24] M. Bellare and T. Ristenpart, "Hash functions in the dedicated-key setting: Design choices and MPP transforms," in *International Colloquium on Automata, Languages, and Programming*, 2007.

[25] M. Daum, "Hash collisions (The poisoned message attack) The story of Alice and her boss," *Presented at the rump session of Eurocrypt'05*, 2005.

[26] M. Gebhardt, G. Illies and W. Schindler, "A Note on the Practical Value of Single Hash Collisions for Special File Formats," in *Sicherheit*, vol. 77, Citeseer, 2006, pp. 333-344.

[27] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International workshop on fast software encryption*, 2004.

[28] A. Maetouq, S. Daud, N. Ahmad, N. Maarop, N. N. A. Sjarif and H. Abas, "Comparison of hash function algorithms against attacks: A review," *International Journal of Advanced Computer Science and Applications, br*, vol. 8, 2018.

[29] D. J. Bernstein, "Quantum attacks against Blue Midnight Wish, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Shabal, SHAvite-3, SIMD, and Skein," *Citeseer*, 2010.

[30] P. Hoffman and B. Schneier, "Attacks on cryptographic hashes in internet protocols," RFC 4270, November, 2005.

[31] D. J. Bernstein, "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete," *SHARCS*, vol. 9, p. 105, 2009.

[32] A. Petzoldt, M.-S. Chen, B.-Y. Yang, C. Tao and J. Ding, "Design principles for HFEv-based multivariate signature schemes," in *International conference on the theory and application of cryptology and information security*, 2015.

[33] C. Boura and A. Canteaut, "On the boomerang uniformity of cryptographic sboxes," *IACR Transactions on Symmetric Cryptology*, pp. 290-310, 2018.

[34] M. Lamberger, F. Mendel, M. Schlaffer, C. Rechberger and V. Rijmen, "The rebound attack and subspace distinguishers: Application to Whirlpool," *Journal of Cryptology*, vol. 28, no. 2, pp. 257-296, 2015.

[35] A. D. Myasnikov and A. Ushakov, "Length based attack and braid groups: cryptanalysis of Anshel-Anshel-Goldfeld key exchange protocol," in *International Workshop on Public Key Cryptography*, 2007.

[36] D. M. A. Cortez, A. M. Sison and R. P. Medina, "Cryptographic randomness test of the modified hashing function of SHA256 to address length extension attack," in *Proceedings of the 2020 8th International Conference on Communications and Broadband Networking*, 2020.

[37] F. Muller, "The MD2 hash function is not one-way," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2004.

[38] B. Kaliski, "RFC1319: The MD2 Message-Digest Algorithm," RFC Editor, 1992.

[39] C. Adams and S. Lloyd, Understanding public-key infrastructure: concepts, standards, and deployment considerations, Sams Publishing, 1999.

[40] N. Rogier and P. Chauvaud, "MD2 is not secure without the checksum byte," *Designs, Codes and Cryptography*, vol. 12, no. 3, pp. 245-251, 1997.

[41] S. S. Thomsen, "An improved preimage attack on md2," *Cryptology ePrint Archive*, 2008.

[42] L. R. Knudsen, J. E. Mathiassen, F. Muller and S. S. Thomsen, "Cryptanalysis of MD2," *Journal of cryptology*, vol. 23, no. 1, pp. 72-90, 2010.

[43] R. C. Merkle, "A fast software one-way hash function," *Journal of Cryptology*, vol. 3, no. 1, pp. 43-58, 1990.

[44] E. Biham, "New techniques for cryptanalysis of hash functions and improved attacks on Snefru," in *International Workshop on Fast Software Encryption*, 2008.

[45] R. L. Rivest, "The MD4 message digest algorithm," in *Conference on the Theory and Application of Cryptography*, 1990.

[46] B. Smith and B. Komar, Microsoft Windows security resource kit, Microsoft Press, 2020.

[47] B. d. Boer and A. Bosselaers, "An attack on the last two rounds of MD4," in *Annual International Cryptology Conference*, 1991.

[48] H. Dobbertin, "Cryptanalysis of MD4," in *International Workshop on Fast Software Encryption*, 1996.

[49] Y. Sasaki, L. Wang, K. Ohta and N. Kunihiro, "New message difference for MD4," in *International Workshop on Fast Software Encryption*, 2007.

[50] G. Leurent, "MD4 is not one-way," in *International Workshop on Fast Software Encryption*, 2008.

[51] J. Guo, S. Ling, C. Rechberger and H. Wang, "Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2010.

[52] R. Rivest and S. Dusse, *The MD5 message-digest algorithm*, MIT Laboratory for Computer Science Cambridge, 1992.

[53] S. Gupta, N. Goyal and K. Aggarwal, "A review of comparative study of md5 and ssh security algorithm," *International Journal of Computer Applications*, vol. 104, no. 14, 2014.

[54] B. d. Boer and A. Bosselaers, "Collisions for the compression function of MD5," in *Workshop on the Theory and Application of of Cryptographic Techniques*, 1993.

[55] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Annual international conference on the theory and applications of cryptographic techniques*, 2005.

[56] H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," in *International Workshop on Fast Software Encryption*, 1996.

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

[58] F. Mendel, N. Pramstaller, C. Rechberger and V. Rijmen, "On the collision resistance of RIPEMD-160," in *International Conference on Information Security*, 2006.

[59] Y. Zheng, J. Pieprzyk and J. Seberry, "HAVAL—a one-way hashing algorithm with variable length of output," in *International workshop on the theory and application of cryptographic techniques*, 1992.

[60] F. Chabaud and A. Joux, "Differential collisions in SHA-0," in *Annual International Cryptology Conference*, 1998.

[61] E. Biham and R. Chen, "Near-collisions of SHA-0," in *Annual International Cryptology Conference*, 2004.

[62] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet and W. Jalby, "Collisions of SHA-0 and Reduced SHA-1," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005.

[63] X. Wang, H. Yu and Y. L. Yin, "Efficient collision search attacks on SHA-0," in *Annual International Cryptology Conference*, 2005.

[64] S. Manuel and T. Peyrin, "Collisions on SHA-0 in one hour," in *International Workshop on Fast Software Encryption*, 2008.

[65] T. Isobe, "A single-key attack on the full GOST block cipher," in *International Workshop on Fast Software Encryption*, 2011.

[66] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak and J. Szmidt, "Cryptanalysis of the GOST hash function," in *Annual International Cryptology Conference*, 2008.

[67] C. Rechberger, V. Rijmen and N. Sklavos, "The NIST cryptographic workshop on hash functions," *IEEE Security & Privacy*, vol. 4, no. 1, pp. 54-56, 2006.

[68] "Google will drop SHA-1 encryption from Chrome by January 1, 2017," 18 12 2015. [Online]. Available: https://venturebeat.com/2015/12/18/google-will-drop-sha-1-encryption-from-chrome-by-january-1-2017/. [Accessed 18 3 2022].

[69] "The end of SHA-1 on the Public Web," Mozilla Security Blog, 23 2 2017. [Online]. Available: https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/. [Accessed 18 3 2022].

[70] X. Wang, Y. L. Yin and H. Yu, "Finding collisions in the full SHA-1," in *Annual international cryptology conference*, 2005.

[71] P. Karpman, T. Peyrin and M. Stevens, "Practical free-start collision attacks on 76-step SHA-1," in *Annual Cryptology Conference*, 2015.

[72] "Announcing the first SHA1 collision," Google Security Blog, 23 2 2017. [Online]. Available: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html. [Accessed 18 3 2022].

[73] G. Leurent and T. Peyrin, "From collisions to chosen-prefix collisions application to full SHA-1," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2019.

[74] G. Leurent and T. Peyrin, "SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[75] R. Anderson and E. Biham, "Tiger: A fast new hash function," in *International Workshop on Fast Software Encryption*, 1996.

[76] J. Kelsey and S. Lucks, "Collisions and near-collisions for reduced-round tiger," in *International Workshop on Fast Software Encryption*, 2006.

[77] F. Mendel and V. Rijmen, "Cryptanalysis of the Tiger hash function," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2007.

[78] F. Mendel, B. Preneel, V. Rijmen, H. Yoshida and D. Watanabe, "Update on tiger," in *International Conference on Cryptology in India*, 2006.

[79] J. Daemen and C. Clapp, "The Panama cryptographic function," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 23, no. 12, pp. 42-46, 1998.

[80] V. Rijmen, B. V. Rompay, B. Preneel and J. Vandewalle, "Producing collisions for PANAMA," 2001.

[81] J. Daemen and G. V. Assche, "Producing collisions for PANAMA, instantaneously," in *International Workshop on Fast Software Encryption*, 2007.

[82] P. Barreto, V. Rijmen and others, "The Whirlpool hashing function," in *First open NESSIE Workshop, Leuven, Belgium*, 2000.

[83] F. Mendel, C. Rechberger, M. Schlaffer and S. S. Thomsen, "The rebound attack: Cryptanalysis of reduced Whirlpool and Grostl," in *International Workshop on Fast Software Encryption*, 2009.

[84] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "The road from Panama to Keccak via RadioGatun," in *Dagstuhl Seminar Proceedings*, 2009.

[85] N. Kishore and P. Raina, "Parallel cryptographic hashing: Developments in the last 25 years," *Cryptologia*, vol. 43, no. 6, pp. 504-535, 2019.

[86] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Radiogatun, a belt-and-mill hash function," *Cryptology ePrint Archive*, 2006.

[87] D. Khovratovich, "Two attacks on RadioGatun," in *International Conference on Cryptology in India*, 2008.

[88] D. Khovratovich, "Cryptanalysis of hash functions with structures," in *International Workshop on Selected Areas in Cryptography*, 2009.

[89] C. Bouillaguet and P.-A. Fouque, "Analysis of the Collision Resistance of RadioGatunUsing Algebraic Techniques," in *International Workshop on Selected Areas in Cryptography*, 2008.

[90] T. Fuhr and T. Peyrin, *Cryptanalysis of RadioGatun*, Cryptology ePrint Archive, Report 2008/515, 2008.

[91] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche and G. NIST, "Keccak and the SHA-3 Standardization," *NIST, Gaithersburg*, 2013.

[92] A. Biryukov, L. Perrin and A. Udovenko, "The secret structure of the S-box of Streebog, Kuznechik and Stribob," *Cryptology ePrint Archive*, 2015.

[93] B. Ma, B. Li, R. Hao and X. Li, "Improved cryptanalysis on reduced-round GOST and Whirlpool hash function (Full version)," *Cryptology ePrint Archive*, 2014.

[94] J. Guo, J. Jean, G. Leurent, T. Peyrin and L. Wang, "The usage of counter revisited: Second-preimage attack on new russian standardized hash function," in *International Conference on Selected Areas in Cryptography*, 2014.

[95] R. AlTawy and A. M. Youssef, "Watch your constants: malicious Streebog," *IET Information Security*, vol. 9, no. 6, pp. 328-333, 2015.

[96] R. AlTawy, A. Kircanski and A. M. Youssef, "Rebound attacks on Stribog," in *International Conference on Information Security and Cryptology*, 2013.

[97] Z. Wang, H. Yu and X. Wang, "Cryptanalysis of GOST R hash function," *Information Processing Letters*, vol. 114, no. 12, pp. 655-662, 2014.

[98] A. Biryukov, L. Perrin and A. Udovenko, "Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1 (Full Version)," in *Cryptology ePrint Archive*, 2016.

[99] J.-P. Aumasson, W. Meier, R. C.-W. Phan and L. Henzen, "The hash function BLAKE," *Springer*, 2014.

[100] J.-P. Aumasson, W. Meier, R. C.-W. Phan and L. Henzen, "Blake2," in *The Hash Function BLAKE*, 2014.

[101] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," in *International Conference on Applied Cryptography and Network Security*, 2013.

[102] J. Guo, P. Karpman, I. Nikolic, L. Wang and S. Wu, "Analysis of BLAKE2," in *Cryptographers' Track at the RSA Conference*, 2014.

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

[104] C. Dobraunig, M. Eichlseder and F. Mendel, "Analysis of the kupyna-256 hash function," 2016.

[105] J. Zou and L. Dong, "Cryptanalysis of the round-reduced Kupyna hash function," *Cryptology ePrint Archive*, 2015.

[106] O. Duman, "Application of Fault Analysis to Some Cryptographic Standards," Concordia University, 2016.