International Journal of Advanced Research in Engineering Technology & Science

ISSN: 2349-2819

Email: editor@ijarets.org Volume-9, Issue-3 March – 2022 www.ijarets.org

# EVALUATION OF THE PROPOSED HASH FUNCTIONS' PERFORMANCE

**Sudheer Kumar,**

Research Scholar, Dept of Computer Application,

Maharaja Agrasen Himalayan Garhwal University

**Dr Inderpal Singh,**

Professor, Dept of Computer Application,

Maharaja Agrasen Himalayan Garhwal University

## ABSTTRACT

The uses of hash functions are expanding from technical to mundane tasks, thus we must demand a higher level of security and efficiency. In daily life, dedicated hash functions—in particular, lightweight hash functions—are utilized for RFIDs, smart cards, and cards with embedded microchips. These are only a few examples of how hash functions have been used, although there are many more. Examples include malware detection, virus protection, and many more. Properties like the avalanche effect, preimage, second-preimage, and collision resistance are satisfied by an ideal hash function. A few of the aforementioned characteristics may become problematic in some applications, such as monitoring updated data or spotting malicious files. In a perfect hash function, a virus-infested file would produce a completely new hash digest, and we would never be able to determine which portion of the file is infected. We require a hash function extension for situations like this. It is known as hash variations, and we will talk about it in the following section. For instance, fuzzy hashing is a notion that is employed in the situation discussed to get around the avalanche effect in hash functions. Hash functions are used to generate sequences of pseudo random numbers by pseudo random number generators (PRNGs). These are effective at producing lengthy sequences of pseudo-random numbers. A seed, which is a completely random number, is used as the first input to a hash function in this procedure. The result is a hash function that produces a fixed-length pseudo-random sequence. It does pass the majority of statistical tests or random sequence features. This makes it possible to manufacture a lengthy string of fictitious random integers.

***KEY WORDS:*** *Random Numbers, Sequences, Digital Signatures, Message Authentication.*

## INTRODUCTION

Numerous protocols and applications, including digital signatures, message authentication, data integrity, password security, SSL, and the production of pseudo-random numbers, use cryptographic hash functions. The effectiveness of the cryptographic hash algorithms utilized determines how secure certain applications and protocols are. The simplicity of computing digital fingerprints, the difficulty of deriving data from a given digital fingerprint, and the difficulty of locating data with the same hash value are fundamental characteristics of hash functions. These characteristics, however, are insufficient to allow the use of cryptographic hash functions in cryptographic applications. In addition to these, strong cryptographic hash algorithms should also have statistical randomness and collision resistance characteristics. Finding any pair of messages with the identical hash value should be impossible, according to the collision resistance property. The avalanche criterion, which assesses the impact of hash input on hash value bits, is used to evaluate a hash function's capacity to distribute hash values at random. The avalanche property states that there is a 50% chance that every output bit will change when one input bit changes. The distribution of hash values becomes more random the closer this avalanche condition is met. To assess the performance of the hash functions in relation to the aforementioned criteria and those indicated below, we carried out a number of statistical benchmark tests that were recommended in.

## RESEARCH METHODOLOGY

### Randomness Test

For this test, we used a 512-bit input message M and computed the matching hash value. The new 512 modified messages M have been generated for 1 i 512 by modifying the ith bit of M. After creating hash values for each of these fresh messages, we calculated Hamming distances between the hash values of the original and modified messages. For DSHA-1 and MDA-192, the ideal values are 80 and 96, respectively. For the above-generated messages, we discovered that the Hamming distances for DSHA-1 and MDA-192 are respectively between 68 and 97 and 71 and 122. Similarly, we discovered that the Hamming distances for MNF-256 range from 106 to 153 (ideally, they should be 128) for the messages mentioned above.

**TABLE 1: RANGE OF DISTANCES FOR SHA-1 AND DSHA-1**

| | SHA-1 | | DSHA-1 | |
|---|---|---|---|---|
| Distances | Hash pairs | Percentage (%) | Hash pairs | Percentage (%) |
| 80±5 | 306 | 59.76 | 328 | 64.06 |
| 80±10 | 467 | 91.29 | 471 | 91.99 |
| 80±15 | 499 | 97.46 | 507 | 99.02 |

**TABLE 2: RANGE OF DISTANCES FOR MDA-192.**

| Distances | Hash pairs | Percentage (%) |
|---|---|---|
| 96±5 | 260 | 50.78 |
| 96±10 | 412 | 80.46 |
| 96±15 | 484 | 94.53 |

**TABLE 3: RANGE OF DISTANCES FOR NEWFORK-256 AND MNF-256**

| | NewFORK-256 | | MNF-256 | |
|---|---|---|---|---|
| Distances | Hash pairs | Percentage (%) | Hash pairs | Percentage (%) |
| 128±5 | 231 | 45.11 | 263 | 51.36 |
| 128±10 | 383 | 74.81 | 428 | 83.59 |
| 128±15 | 476 | 92.96 | 484 | 94.53 |

**Bit Variance Test**

The bit variance test involves altering the input message bits and assessing the effect on the digest bits. The message digests (for each altered input) are calculated after bits of an input message are changed. The probability Pi for each digest bit to have a value of 1 or 0 is then calculated from all the digests generated. In terms of the bit variance test, the hash function under examination performs optimally if Pi (1) Pi (0) 1/ 2 for all digest bits i 1 i n, where n is the digest length. As a result, the bit variance test essentially assesses the consistency of each digest bit. We have assessed the outcomes for as many as 513 messages because it is computationally challenging to take into account all input message bit changes.

**TABLE 4: RESULTS FOR BIT VARIANCE ANALYSIS**

| Hash function | Number of Digests | Mean frequency of 1s (Expected) | Mean frequency of 1s (Calculated) |
|---|---|---|---|
| SHA-1 | 513 | 256.50 | 249.11 |
| DSHA-1 | 513 | 256.50 | 256.67 |
| MDA-192 | 513 | 256.50 | 257.03 |

| | | | |
|---|---|---|---|
| NewFORK-256 | 513 | 256.50 | 251.12 |
| MNF-256 | 513 | 256.50 | 257.15 |

The investigation mentioned above demonstrates that the hash function exhibits a respectable avalanche effect. As a result, it can be applied to cryptographic purposes.

**Statistical Analysis of Diffusion**

The fundamental design criterion for hash functions and encryption methods is diffusion. To conceal the statistical nature of the plaintext, diffusion refers to spreading the influence of a single plaintext bit over as much of the cipher text as possible. Each bit for the hash value in binary format can only be either 1 or 0. Therefore, the ideal diffusion effect should result in a 50% chance that each output bit will change as a result of even the smallest changes in the input. The following diffusion test was run by us: A message is selected at random, and a hash value is produced. A fresh hash value is then created after a random bit in the message is selected and toggled.

**RESULTS AND DISCUSSION**

We compare the two hash values and count the number of altered bits as Bi before concluding.

**TABLE 5: STATISTICS OF NUMBER OF CHANGED BITS FOR SHA-1**

| *N* | *B* | *P*% | □*B* | □*P*% |
|---|---|---|---|---|
| 256 | 79.4519 | 49.4289 | 6.6232 | 3.5634 |
| 512 | 79.6023 | 49.3874 | 6.4522 | 3.2356 |
| 1024 | 80.0313 | 49.1093 | 6.6743 | 3.8431 |
| 2048 | 79.3926 | 49.3945 | 6.3472 | 3.6452 |
| Mean | 79.6195 | 49.3301 | 6.5242 | 3.5718 |

**TABLE 6: STATISTICS OF NUMBER OF CHANGED BITS FOR DSHA-1**

| *N* | *B* | *P*% | □*B* | □*P*% |
|---|---|---|---|---|
| 256 | 80.4609 | 50.2881 | 6.3387 | 3.9617 |
| 512 | 80.6035 | 50.3772 | 6.2059 | 3.8787 |

| | | | | |
|---|---|---|---|---|
| 1024 | 80.5215 | 50.3259 | 6.1935 | 3.8709 |
| 2048 | 80.4171 | 50.2606 | 6.1707 | 3.8567 |
| Mean | 80.50075 | 50.31295 | 6.2272 | 3.8920 |

**TABLE 7: STATISTICS OF NUMBER OF CHANGED BITS FOR MDA-192.**

| *N* | *B* | *P*% | □*B* | □*P*% |
|---|---|---|---|---|
| 256 | 96.1132 | 50.0590 | 9.4420 | 4.9177 |
| 512 | 96.3242 | 50.1688 | 8.6162 | 4.4876 |
| 1024 | 96.0019 | 50.0010 | 8.4931 | 4.4234 |
| 2048 | 96.1928 | 50.1004 | 8.6174 | 4.4882 |
| Mean | 96.1580 | 50.0823 | 8.6174 | 4.5792 |

**TABLE 8: STATISTICS OF NUMBER OF CHANGED BITS FOR NEWFORK-256.**

| *N* | *B* | *P*% | □*B* | □*P*% |
|---|---|---|---|---|
| 256 | 127.4395 | 49.7811 | 9.6344 | 3.7634 |
| 512 | 127.9187 | 49.9682 | 8.9791 | 3.5074 |
| 1024 | 128.1426 | 50.0556 | 8.5386 | 3.3353 |
| 2048 | 127.7227 | 49.8916 | 9.0768 | 3.5456 |
| Mean | 127.8059 | 49.9241 | 9.0572 | 3.5379 |

**TABLE 9: STATISTICS OF NUMBER OF CHANGED BITS FOR MNF-256**

| *N* | *B* | *P*% | □*B* | □*P*% |
|---|---|---|---|---|
| 256 | 128.3398 | 50.1327 | 8.4072 | 3.2841 |
| 512 | 128.2559 | 50.0999 | 8.4417 | 3.2975 |
| 1024 | 128.4414 | 50.1724 | 8.3763 | 3.2721 |
| 2048 | 129.1615 | 50.4537 | 8.9771 | 3.5066 |

| | | | | |
|---|---|---|---|---|
| Mean | 128.5497 | 50.2146 | 8.5505 | 3.3401 |

## ANALYSIS OF COLLISION RESISTANCE

Collision denotes that the hash values produced by several random inputs are identical. We have run two collision tests to look at the suggested hash function's capacity to withstand collisions. The first experiment involves selecting a random message and generating a hash value for it. After that, the hash value is saved in ASCII format. A fresh hash value is then created and saved in the same format after a bit in the message is randomly chosen and toggled.

**TABLE 10: RESULTS FOR ABSOLUTE DIFFERENCE**

| *AD* | Maximum | Minimum | Mean | Mean/character |
|---|---|---|---|---|
| SHA-1 | 2332 | 695 | 1642.63 | 82.13 |
| DSHA-1 | 2834 | 993 | 1724.40 | 86.22 |
| MDA-192 | 3155 | 927 | 2068.11 | 86.17 |
| NewFORK-256 | 3178 | 1686 | 2658.31 | 83.07 |
| MNF-256 | 3650 | 1833 | 2752.74 | 86.02 |

In the second experiment, a randomly selected message's hash value is generated and saved in ASCII format. Every two hash results should be compared since the subject of this experiment is the potential for collisions between every two hash results. There are 2048 iterations of the simulation. For SHA-1, DSHA-1, and MDA-192, the distribution of the number of ASCII characters having the same value at the same position is plotted.

## ROBUSTNESS AGAINST DIFFERENTIAL CRYPTANALYSIS

We investigated the proposed hash function's resistance to differential cryptanalysis. This attack examines the pairs of hashes and their matching pairings of plaintext. The message digest pair difference d for the corresponding two message digests can be determined, for instance, if the difference between two messages is two bits, or, let's say, d=2. The standard deviation () is computed from the distribution of d corresponding to various message pairs. The hash function is resistant to differential cryptanalysis if 10%. A 10-byte input message was taken into consideration for the experiment. For each conceivable d, 1, 2, 4, 8, 16, and 32-bit difference for an input message, experiments are conducted. The findings in Table demonstrate the security of the suggested hash functions against the differential attack.

**TABLE 11: RESULTS FOR DIFFERENTIAL CRYPTANALYSIS.**

| *d* | SHA-1 ($\sigma$) | DSHA-1 ($\sigma$) | MDA-192 ($\sigma$) | NewFORK-256 ($\sigma$) | MNF-256 ($\sigma$) |
|---|---|---|---|---|---|
| 1 | 7.56 | 6.18 | 8.67 | 8.81 | 8.39 |
| 2 | 8.34 | 6.11 | 8.39 | 8.86 | 8.57 |
| 4 | 10.02 | 6.07 | 7.64 | 8.92 | 8.45 |
| 8 | 8.87 | 6.14 | 8.13 | 9.26 | 8.27 |
| 16 | 10.14 | 6.23 | 8.49 | 8.43 | 8.95 |
| 32 | 7.19 | 6.09 | 8.84 | 9.89 | 8.76 |

## COMPUTATION TIMES OF HASH VALUES

The following approach was used to conduct the speed test on an Intel Pentium 4 CPU running at 1.47 GHz: we chose a message size S (in bytes) and produced 1000 random messages of that size. Each of these 1000 messages is subjected to the hash function, and the time taken to compute each of them is recorded. The average of more than 1000 samples is then calculated. All five of the algorithms go through this process. Table lists the average CPU computation times (in seconds) for the SHA-1, DSHA-1, MDA-192, MNF-256, and NewFORK-256 algorithms. It has been discovered that proposed DSHA-1 and MDA-192 require more time to compute the hash value than SHA-1. This is because DSHA-1 adds additional 2560-bit inputs to the compression function and MDA-192 adds additional 32-bit chaining variables and complex step operations for processing. However, the performance of DSHA-1 is essentially identical to that of SHA-1 when a tiny dither sequence is included. On the basis of randomness, collision resistance, diffusion quality, and speed, we have evaluated the performance of suggested hash functions and their parent hash algorithms. We discovered that the proposed hash functions had good randomness and non linear behavior from the results of the randomness test. Our suggestions achieved their maximum performance, according to a bit variance test, and they demonstrate a good avalanche effect. According to a statistical investigation of the diffusion effect, all three of the hash functions that have been proposed have steady diffusion capabilities. Additionally, proposals have proven to be well-resistant to collision assaults.

## APPLICATIONS OF HASH FUNCTIONS

When it comes to network security, hash functions are a crucial tool because message integrity is our first priority. Hash functions don't just have uses for message integrity; they also have a wide range of uses in related fields.

**Password Hashing**

Since the early days of UNIX Operating Systems, we have used the concept of password hashing in computers. Users' passwords for Unix operating systems are kept in a related file or database as hashed values. If the operating system keeps passwords in their current state (that is, in plain text), an adversary would find it simple to obtain any user's password if he obtained access to the password file on the system. As a result, the hash value of the password is retained rather than the actual password itself, rendering password theft useless until the adversary is also aware of the hashing algorithm. When a user inputs a password, the hash value of that password is compared to a previously stored hash value; if the two values match, access is given to the user; otherwise, it is denied. With the proposed hash function—R-U Hash—we have created a password hashing tool. The software prompts you for a password, which it then checks before returning.

**Digital Signatures**

Like a handwritten signature on some digital data on a computer network, the digital signature is a technique to authenticate the sender (or signatory) and the contents of message being signed. As a result, we can employ a digital signature to confirm a document. It fixes the non-repudiation issue. Non-repudiation is a circumstance in which the sender of a message occasionally claims that he did not transmit it after it has been received, and occasionally, another party delivers the message to the recipient using the name of another sender. A digital signature may be useful in these situations to settle disagreements.

**Virus Checking**

Computer data is at risk from viruses because they replicate themselves. Every time a virus impacts a file or message, it modifies the contents of the file, even slightly. while a file is created, that is, while the file is in its initial form, the hash value of the file may be stored. Later, it is possible to compare the digests of the original, unaffected file with the potential affected file. If both digests are equal, we may say that the file is virus-free; if not, it has been. Software vendors and developers also employ this method for distributing software to the client.

**Pseudo-Radom Number Generation**

Statistically random values or components are known as pseudo random numbers since they are obtained from a known starting point. Pseudorandom numbers are helpful in providing the necessary values for processes that require randomization, such as those for synchronizing sending and receiving devices in a spread spectrum transmission or for synthesizing test signals. These numbers are referred to as "pseudo" random because the algorithm can repeat the sequence and is therefore not completely random.

**Trusted Digital Time Stamping**

It may be necessary to provide evidence that a specific digital document existed at a particular time in various scenarios when we need to settle a disagreement or a non-repudiation case. In these situations, digital time stamping is utilized as the fix. Digital time stamps are created using both hash functions and digital signatures. The requesting party computes the hash of the desired file, communication, or document and transmits it to a Time Stamp Authority (TSA) as the initial step in producing digital time stamping.

**Password Protection**

Password protection makes extensive use of hash functions. Nowadays, practically all computers, mobile devices, and other electronic devices have built-in password protection. It operates under the tenet that every time you log into a password-protected system, the hash of the password you input is compared to the hash value saved in the pass-file. Instead of saving your password directly, it stores the password's hash. Here, we make use of the hash function's ability to avoid collisions because, without it, we could find the same hash digest for two different passwords. In this case, the system would let an intruder access, which is undesirable in a real-world situation. Electronic Signature An electronic equivalent of a physical signature is a digital signature. The difference between the two is that a digital signature also considers the message when producing it, whereas a physical signature uses the same stamp for all messages. It confirms the message's authenticity and prevents the sender from retracting. On public key cryptography, it is based. Everybody in a public key cryptography has two keys: a private signing key (sk) that is only known to the signer, and a public verification key (pk) that is known to everybody. With the aid of a message (to be signed) M and the sender's signing/private key sk, the signature is created using a cryptographic hash function h. Utilizing h, the message's hash digest is calculated. By utilizing the secret key of the signer to encrypt h(M), a signature Signsk(h(M)) is produced. A message M and the sender's signature Signsk(h(M)) are transmitted to the recipient. Signsk(h(M)) is the sender's signature. When the pair (M, Signsk(h(M)) is received, the receiver computes the message's hash. The signature is then applied with the public

key. If both digests match, the signature has been authenticated and is not a forgery. Additionally, the message's reliability is checked. This operation is facilitated easily and economically using a cryptographic hash function.

**Fuzzy Hashing**

Fuzzy hashing is a concept that gained notoriety for its outstanding performance in the detection of malware. The primary goal of fuzzy hashing is to assess how similar the two unique objects are. Additionally, it provides the percentage level of resemblance. This method of expressing file similarities and the degree to which viruses and malware have affected them is incredibly helpful. It provides a number or a percentage for the degree of similarity. It differs from hashing since hashing indicates whether or not two files are identical. The degree of resemblance between the two identical input files would not be shown by it. Context-triggered piecewise hashing (CTPH), a combination of a piecewise hash and a rolling hash, is the standard method for fuzzy hashing. CTPH can be carried out with the use of a cryptographic hash. The message is broken up into blocks of a set length, and the hash digest of each block is calculated. The differences between each of these hash digests are next examined. The detection of steganographic images, or finding hidden messages in the photographs, can also be done using fuzzy hashing. This is a fascinating and difficult task that can be completed via fuzzy hashing. Additionally, spam detection uses it. By using fuzzy hashing, all spam and unwanted emails are recognized. Different scans of a biometric feature correspond to the same key even in biometrics. Here, the idea of fuzzy hashing is once more used.

**Fuzzy Plagiarism**

One of the most frequently used words when discussing original or innovative work and legitimate research is "plagiarism." All research papers and PhD theses submitted to universities and other organizations are subjected to plagiarism checks to ensure their validity. Plagiarism basically verifies that a word, phrase, sentence, or paragraph was not lifted verbatim from another source from one of the many online databases that are accessible. In order to match the strings, it simply compares the lines of testing material (text) with all online text databases that are now available. There are numerous different plagiarism detection programs on the market, and they all operate under the same guiding idea of comparing the phrases to find matches.

**CONCLUSION**

Digital signatures sign documents using the sender's private key. Because a digital signature requires the use of the sender's private key and only the sender can sign on his behalf, the sender cannot deny sending a message that has that signature. Since a digital signature uses a hash function to confirm the integrity of the message before

applying the signature algorithm, it helps to prevent message tampering after the sender has signed it. Additionally, if the receiver's end could not verify the signature, he would discard the bogus communication. Hash functions therefore ensure the message's originality in the digital signature system.

A simple hash function that can be utilized in this application is Neeva. It is fairly effective for its size and can be utilized for this application. The authenticity of a research paper or thesis can be determined using fuzzy hashing in an incremental manner. A small number of programs that are specifically designed for this application may actually use an effective hash function like Neeva for integrity checking.

## APPLICATION OF NEEVA HASH FUNCTION IN A WIRELESS SENSOR NETWORK

A wireless sensor network (WSN) is a network made up of numerous tiny sensors that are sensitive to changes in the physical environment, such as sound, pressure, light, and electricity. WSNs were first utilized in military applications such battlefield surveillance and the detection of chemical, biological, and nuclear attacks. These can keep an eye on a place where the operational circumstances are difficult or impossible because of the climate. It is very important in the biological and medicinal fields. This network operates using a large number of sensor nodes that are set up to carry out the monitoring task. Resource-limited devices are, in essence, what sensor nodes are. A sensor node is made up of a power management module, which supplies the energy needed for communication, a sensor, which detects changes in the physical environment, a microprocessor, which receives and processes the sensor's data, and a transceiver, which transmits the data and enables physical communication. The sensors react to a physical change in the environment and cause a response across the network and all of the connected devices.

### WSN challenges

(i) Resource Constraints: WSN operates in a setting with severe resource limitations. Because the processor and RAM in the sensory nodes are so memory constrained, there is an energy constraint. WSN ought to be capable of operating effectively and safely in memory-constrained conditions.

(ii) electricity Consumption: Wireless sensory networks require a constant flow of electricity to process data, sense physical changes, and communicate those changes throughout the network. The power management module in the WSN typically generates the energy required to carry out these tasks. This can be accomplished by providing direct power to a WSN or by using a passive system (ad hoc system) for power supply. One need to operate a microelectromechanical device is the minimal energy consumption.

Extreme environmental circumstances: The WSN should be able to endure extremely harsh and unusable environmental conditions. They must be capable of self-organization and self-configuration in response to their environment.

**WSN Security Requirements**

(i) Data Confidentiality: Only the intended recipient should be able to understand the data that is being transmitted by the nodes of the sensor network.

(ii) Data Integrity: Any message or piece of information sent over a WSN should be unaltered. It should not be changed when it moves from one network node to its destination node.

(iii) Data Freshness: Data must be reliable and recent enough to prevent an adversary from playing back older messages and representing them as the most recent. As an alternative, a nonce or a counter with the message can be utilized.

(iv) Secure Localization: Each node in a WSN needs to be able to be accurately located. It aids in identifying network issues.

(v) Authentication: A key component of wireless sensor networks is data authentication. It confirms that the node is the one it claimed to be and that the communication between nodes is authentic. MAC can be used to do this in communication.

Neeva is applicable to WSN sensor nodes. We've previously established through our analysis of Neeva that the application only needs a tiny amount of memory. Consequently, wireless sensor networks may use them.

**REFERENCES**

1. Wang X., Lai X., Feng D., Chen H., Yu X., “Cryptanalysis of the Hash Functions MD4 and RIPEMD”, Eurocrypt’05, LNCS, vol. 3494, pp. 1-18, 2005.
2. Naito Y., Sasaki Y., Kunihiro N., Ohta K., “Improved Collision Attack on MD4”, Cryptology ePrint Archive, Report 2005/151, May 2005. http://eprint.iacr.org/2005/151.pdf
3. Yu H., Wang G., Zhang G., Wang X., “The Second-Preimage Attack on MD4”, CANS’05, LNCS, vol. 3810, pages 1-12, 2005.

4. Yu H., Wang X., "Multi-collision Attack on the Compression Functions of MD4 and 3-Pass HAVAL", ICISC'07, LNCS, vol. 4817, pp. 206-226, 2007.

5. Boer B. den, Bosselaers A., "Collisions for the Compression Function of MD5", Eurocrypt'93, LNCS, vol.765, pp. 293–304, 1994.

6. Wang X., Feng D., Lai X., Yu H., "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", Cryptology ePrint Archive, Report 2004/199, 2004.

7. Hawkes P., Paddon M., Rose G., "Musings on the Wang et al. MD5 Collision", Cryptology ePrint Archive, Report 2004/264, 2004.

8. Black J., Cochran M., Highland T., "A Study of the MD5 Attacks: Insights and Improvements", Fast Software Encryption, pp. 262-277, 2006.

9. Wang X., Yu H., "How to Break MD5 and Other Hash Functions", Eurocrypt'05, LNCS, vol. 3494, pp.19-35, 2005.

10. Klima V., "Finding MD5 Collisions on a Notebook PC using Multi-message Modifications", Cryptology ePrint Archive, Report 2005/102, 2005. http://eprint.iacr.org/2005/102.

11. Sasaki Y., Naito Y., Kunihiro N., Ohta K., "Improved Collision Attack on MD5", Cryptology ePrint Archive, Report 2005/400, 2005. http://eprint.iacr.org/2005/400.