MD5 Collision Vulnerability: Generating Identical Hash Values From Two Different Computer Programs

Colby J. Leclerc

Abstract

A common theme for cryptographic hashing algorithms is for them to be created, used, and trusted until a flaw is found, rendering the algorithm no longer cryptographically secure [1]. When such instances occur, many times the deprecated algorithm tends to still be in use, posing a significant security flaw in such systems. Specifically, the Message Digest 5 algorithm has been rendered cryptographically insecure for collision resistance, and 1st preimage resistance. Although there still exists evidence for the continued use of MD5, it's current vulnerabilities mean we should move to a more secure alternative.

1 Introduction

In 1991, Professor Ronald Rivest of MIT created the fifth version of the Message Digest algorithm, denoted as MD5. The fifth version was created as a response to the security issues found in Message Digest 4 after extensive cryptanalysis was performed. However, in 1996 a security flaw in the algorithm was found by Hans Dobbertin, leading security experts to recommend the deprecation of the MD5 algorithm [2]. Despite such recommendations, MD5 has still been used to verify the integrity of program files, which posses a risk due to the nature of the vulnerability.

2 Preliminaries

Definition 2.1 *Message Digest 5 (MD5) is a version of the Message Digest algorithm, designed in 1991 by Ron Rivest with the goal of addressing the weaknesses found in Message Digest 4 (MD4). The hash algorithm uses four variables of 32 bits each in a round-robin fashion to create a value that is compressed to generate the hash. [1]*

Definition 2.2 *A cryptographic hash function h takes as input a message of arbitrary length and produces as output a message digest m of fixed length [3]*

Definition 2.3 *A digest is the output of a hash function. For example, MD5 has a digest length of 128 bits.*

Definition 2.4 *Let h be a cryptographic hash function, and m be a message of arbitrary length. There are two categories of collision resistance, weakly collision free, and strongly collision free. Weakly collision free means that given x, it is computationally infeasible to find $x' \neq x$ with $H(x') = H(x)$. Strongly collision free means it is computationally infeasible to find messages m_1 and m_2 with $h(m_1) = h(m_2)$. [3]*

Definition 2.5 *Let h be a cryptographic hash function, and m be a message of arbitrary length. A hash function is said to be cryptographically secure if it fulfills the following properties [3]:*

1. *Given a message m, the message digest $h(m)$ can be calculated very quickly.*
2. *Deterministic, such that the same input results in the same digest*
3. *A small change to the input results in a large change to the output*
4. *Given a y, it is computationally infeasible to find an m' with $h(m') = y$*
5. *It is computationally infeasible to find messages m_1 and m_2 with $h(m_1) = h(m_2)$*

2.1 Attacks on Hash Functions

There exist three categories of attacks on hash functions:

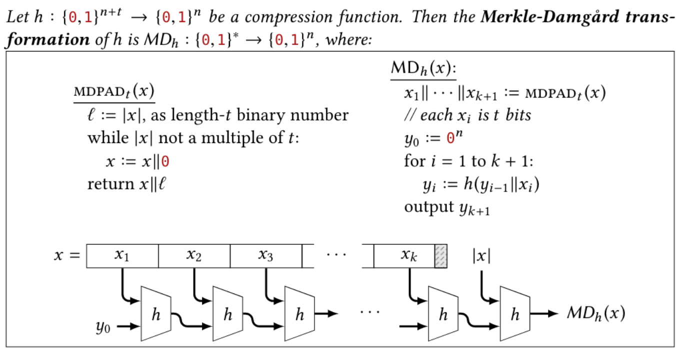
1. Collision: Attacker finds an input that resolves to a specific hash. The attacker must generate both files. "A pair of (x, x') $s.t.$ $H(x) = H(x'), x \neq x'$" [5]
2. 1st Preimage: Modify an input without changing the resulting hash. "For given y, x $s.t.$ $H(x) = y$ " [5]
3. 2nd Preimage: Find any two distinct inputs that have the same hash. An example would be an attacker who has the target file, and wants to modify the file without affecting the computed hash. "For given x, x' $s.t.$ $H(x) = H(x'), x \neq x'$ " [5]

Definition 2.6 *One use case for cryptographic hash functions is to ensure a program file has not been tampered with during transmission by performing integrity checks on the file. This is done by feeding the bytes of the file into a hashing algorithm, and recording the digest (presupposing the hashing algorithm is preimage resistant.) Then the file receiver would feed the bytes into the same hashing algorithm, and compare the digest with the sender's digest (over a separate, preferably secure channel).*

Definition 2.7 *MD5 uses the Merkle-Damgård construction, which is a design method for building hash functions. One feature of this design is the use of a compression function, whose job is to reduce the input parameter to a fixed output size.*

The Merkle-Damgård construction works by breaking the input parameter into equal sized blocks for processing. Then, each block is chained together using a compression function, which takes two parameters. For the first round of compression, the first input parameter is an initialization vector (IV), followed by the bytes of the first block. For subsequent rounds, the compression function uses the output of the previous compression round, along with the bytes of the next block to be processed.

To ensure the block are of equal sized, a padding of 0*s are appended to the input if needed. The construction also has a final block added to the original input blocks, which encodes the length of the input parameter. This process is illustrated below [12]:*



Let $h : \{0,1\}^{n+t} \to \{0,1\}^n$ be a compression function. Then the ***Merkle-Damgård transformation*** *of h is $MD_h : \{0,1\}^* \to \{0,1\}^n$, where:*

```
MDPAD_t(x)
  $\ell := |x|$, as length-$t$ binary number
  while $|x|$ not a multiple of $t$:
    $x := x \| 0$
  return $x \| \ell$
```

```
MD_h(x):
  $x_1 \| \cdots \| x_{k+1} := \text{MDPAD}_t(x)$
  // each $x_i$ is $t$ bits
  $y_0 := 0^n$
  for $i = 1$ to $k+1$:
    $y_i := h(y_{i-1} \| x_i)$
  output $y_{k+1}$
```

3 Applications

Once a hashing algorithm is deemed cryptographically secured, a wide array of information security applications are available.

3.1 Verify File Integrity

How can a user verify the integrity of a file after transmitting it over an insecure channel? The user can use a cryptographic hash function to hash the bytes of the file before and after transmission, then compare the hashes to verify no bytes were altered. Typically the user would transmit the hash codes over a separate channel to verify file integrity.

Although the MD5 algorithm is vulnerable to collision attacks, it is still preimage resistant, meaning its applicability to verify file integrity is still relevant. However, due to MD5's collision vulnerability, one must ensure the hash was generated by a trusted source.

3.2 Password Verification

Definition 3.1 *A salt is a random string of characters that is appended, or prepended, to the plain text of a password before the string is sent to a hashing algorithm. Salting passwords is an information security measure that makes certain attacks (such as rainbow table attacks) on the passwords harder to perform, or computationally infeasible, depending on what information the attacker has obtained.*

Definition 3.2 *A rainbow table is a large list of precomputed hashes for a set of common passwords. Such lists tend to have multiple digests for each password, covering the more common hashing algorithms. Although these tables can be relatively large and expensive to transmit, such an attack method uses more computer space as a trade off to computer processing power (since each password hash does not need to be regenerated)*

There exist multiple security levels regarding safe password storage. Such levels are described below, from least secure to most secure:

1. Storing passwords in plain text
2. Storing passwords using an insecure hashing algorithm
3. Storing passwords using a secure hashing algorithm
4. Storing passwords using a salt in conjunction with a secure hashing algorithm
5. Storing passwords using a unique, long salt per user, with the salt stored separately from the hashed passwords. Passwords are hashed using a secure hashing algorithm

The above list is not exhaustive, however it shows that there exists a wide array of security levels regarding password storage.

Why is password storage important? Firstly is user privacy. If a software system stores passwords in plaintext, then anyone with access to the storage medium will be able to view the passwords of all users. Furthermore, we cannot assume users generate a unique password for each service, thus this poses a major privacy issue.

Secondly is to protect the company. If user passwords are compromised via a data breach, securely storing your user's passwords can mean the difference between 10 million passwords being leaked, vs 10 million hashes being leaked. This difference is not arbitrary, as shown below:

Example 3.3 Let us suppose a company has stored their user's password hashes in one storage location, and the unique salts for each user in a separate location. Furthermore, let us assume the company has a strong infrastructure policy in which internal networks and servers are properly sectioned off.

One day, it is discovered an attacker breached into the password database, and stole all user's passwords. However, because of the infrastructure policy in place, the attacker did not have access to the salts used for each user. Thus, the attacker must now brute force each password, since the attacker knows not the length of the salt, nor the character set used for the salt.

Now let us assume a similar company obeys similar practices mentioned above, but without using the salting method when storing passwords. Once again, an attacker steals the password hashes, but since no salt exists, the attacker can use a rainbow table to discover the plain text for a large portion of the passwords.

In the former case, the company would only have to ask it's users to change their passwords on their software system. In the later case, the company would have to also disclose to its users that their passwords have been compromised, and to change the password of any service that share the same password. Furthermore, according to the newly enacted GDPR legislature, the later company would be liable for the data leaked. [7]

The National Institute for Standards and Technology does not recommend the use of the MD5 hashing algorithm for password storage. [8]

3.3 File Identification

Using cryptographic hash functions, we can assign a document a unique identifier that matches to the specific document, and version of the document. If the document's contents change, then the hash is changed. Using this property, we can identify large volumes of documents by their hash digest.

One such application is for the tracking of documents in professions of law. In many cases, legal teams have thousands of documents that must be collected, tracked, read, and cited to build successful cases. Using the digest allows the legal team to track individual documents, assure no changes were made to the documents, and to quickly find the document's source if the digest is known.

4 MD5 Security

4.1 First Vulnerability Discovered

(Discuss MD5 collision found in 1996) MD5 is based on the Merkle–Damgård construction for building cryptographic hash functions. The goal of this compression function is to take an arbitrary length of input bytes, to then output a fixed-length digest. The MD5 function is designed as follows:

1. Hash input is padded
2. Input is broken into equal sized blocks
3. Using an initialization vector (IV), the first block is fed into the compression function
4. Then, the output of the compression function, along with the next block in the chain, are fed the compression function again, repeating until each block has been compressed
5. The result is a 128-bit length digest of the input

The attack works by finding collisions within the chaining process. In the CryptoBytes article, the attack is described as such [2] :

Let rounds 1/2 be chaining steps 12-26, and rounds 3/4 be chaining steps 36-51.

1. Find an inner collision for rounds 1/2
2. Find an inner collision for rounds 3/4
3. Connect the two inner collisions

Such an attack can be reduced to the following equation:

$$\phi\big(a_1, b_1, z\big) + k = \phi\big(a_2, b_2, z + \delta z\big) \tag{1}$$

"where z is the unknown, ϕ is a Boolean function coming from step operation, and the words a_1, b_1, a_2, b_2, k and δz are given" [2]

4.2 Collision Vulnerability

When designing a secure hash function, typically an n-bit hash function is meant to have a security level of 2^n hash invocations, protecting the hashed data from 1st and 2nd preimage attacks. For collision attacks, the security level is $2^{n/2}$ hash invocations.

MD5 is a 128-bit hash function, thus it's intended security level is 2^{128} against preimage attacks, and 2^{64} against collisions. One method of attack is to discover a vulnerability that reduces the security level from computationally infeasible, to computationally feasible. MD5 suffers from a collision vulnerability, reducing it's collision resistance from requiring 2^{64} hash invocations, to now only 2^{18}. [4]

Attackers can take advantage of this vulnerability by writing two separate programs, and having both program files hash to the same digest. In doing so, an attacker can show the 'good' source code to their target, have the target hash the file, then send over the 'evil' file which would also hash to the same value.

4.3 1st Preimage Vulnerability

Researchers discovered a theoretical preimage attack against MD5 in 2009, that reduces the security level of the algorithm from 2^{128} to $2^{123.4}$ [5]

4.4 2nd Preimage Resistant

The MD5 algorithm is still 2nd preimage resistant, thus MD5 is still safe to use for file integrity checks, as long as the agent generating the files and hash can be trusted. However, it is still recommended to use a stronger hash algorithm to safeguard one's self in the future, in case a 2nd preimage attack becomes computationally feasible.

5 Alternatives

5.1 SHA-2 and SHA-3 Family

The Secure Hashing Algorithm version 2 (SHA-2) family of crypographic hash functions include SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. The proceeding number indicates the bit length of the digest.

NIST "...encourages application and protocol designers to implement SHA-256 at a minimum for any applications of hash functions requiring interoperability" [11]

The SHA family of cryptographic hashing algorithms are designed by members of the National Institute of Standards and Technology, with their most recent iteration the SHA-3 family released in 2015.

In their 2015 press release, NIST states SHA-3 ”...doesn't replace SHA-2, which has not shown any problem, but offers a backup. It takes years to develop a new standard, and we wanted to be prepared in case problems do occur.” [9] [10]

6 Further Research

Although MD5 is still 2nd preimage resistant, further research can be performed to look into weakening MD5's 2nd preimage resistance, as Sasaki and Aoki have [5]. Finding a 2nd preimage vulnerability would potentially lead to the total abandonment of MD5, as suggested by the NIST [11]

References

[1] Ciampa, Mark (2009). *CompTIA Security+ 2008 in depth.* Australia ; United States: Course Technology/Cengage Learning. p. 290.

[2] Hans Dobbertin (Summer 1996). ”The Status of MD5 After a Recent Attack” (PDF). *CryptoBytes* .

[3] 8.1 Hash Functions. *Introduction to Cryptography: with Coding Theory* , by Wade Trappe and Lawrence C Washington, Pearson Education, 2006, pp. 218-219.

[4] T. Xie, F. Liu, and D. Feng. *Fast collision attack on MD5* . IACR Cryptology ePrint Archive, 2013.

[5] Sasaki, Y., Aoki, K.: *Finding Preimages in Full MD5 Faster Than Exhaustive Search* . In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg

[6] Ronald L. Rivest. *Request for Comments 1321: The MD5 Message Digest Algorithm.* The Internet Engineering Task Force, 1992. (http://www.ietf.org/rfc/rfc1321.txt).

[7] Presidency of the Council: *Compromise text. Several partial general approaches have been instrumental in converging views in Council on the proposal for a General Data Protection Regulation in its entirety. The text on the Regulation which the Presidency submits for approval as a General Approach appears in annex,* 201 pages, 11 June 2015, PDF.

[8] Locke, Gary, et al. *Recommendation for Password-Based Key Derivation* . NIST Special Publication 800-132, U.S. Department of Commerce and National Institute of Standards and Technology, Dec. 2010, nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf.

[9] Boutin, Chad. *NIST Releases SHA-3 Cryptographic Hash Standard.* NIST, 8 Jan. 2018, www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard.

[10] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* . FIPS PUB 202 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION , NIST Information Technology Laboratory, nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[11] *NIST Policy on Hash Functions.* NIST Computer Security Resource Center, csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions.

[12] Rosulek, Mike. “The Joy of Cryptography.” Oregon State University Open Textbook Initiative, 22 Jan. 2018.