Interested in learning
more about security?

SANS Institute
InfoSec Reading Room

An Overview of Cryptographic Hash Functions and Their Uses

This paper provides a discussion of how the two related fields of encryption and hash functions are complementary, not replacement technologies for one another.

John Edward Silva
January 15, 2003
GIAC Security Essentials Practical
Version 1.4b Option 1

An Overview of Cryptographic Hash Functions and Their Uses

1.0 Abstract

A hash function is a function that takes a relatively arbitrary amount of input and produces an output of fixed size. The properties of some hash functions can be used to greatly increase the security of a system administrator's network; when implemented correctly they can verify the integrity and source of a file, network packet, or any arbitrary data.

To understand the viability of using hash functions to verify integrity and source of information, one must first examine the properties and origin of the basic hash function. The standard hash function serves as a basis for the discussion of Cryptographic Hash Functions. There are several hash functions currently in use today, including MD5 and SHA1. By examining the history and security available in each function, the user can determine which algorithm is best suited for their application.

Data integrity is a crucial part of any secure system. By using the message digests generated by a cryptographic hash function a system administrator can detect unauthorized changes in files. This is especially important when safeguarding critical system binaries and sensitive databases. After learning the theory behind data integrity verification, the system administrator is given a brief introduction into several freely available tools that can be used immediately for data verification. The tools mentioned are all based on cryptographic hash functions and include Tripwire, md5sum and sha1sum. When used by a knowledgeable system administrator, these tools are invaluable in verifying that a malicious user did not tamper with important system files.

Hash functions can also be combined with other standard cryptographic methods to verify the source of data. When hashing algorithms are combined with encryption, they produce special message digests that identify the source of the data; these special digests are called Message Authentication Codes. The standard algorithm currently used today is called HMAC. The HMAC algorithm provides verification of the source of data, and also prevents against attacks such as the replay attack. Network programmers can use the HMAC algorithm in their applications today; it is currently available in the latest version of Java.

Lastly there is a discussion of how the two related fields of encryption and hash functions are complementary, not replacement technologies for one another.

1

After examining all of the information presented, one will observe that hash functions, when properly implemented, can greatly increase the integrity and security in a system administrator's network.

2.0 Anatomy of a Hash Function

Hash functions are mathematical computations that take in a relatively arbitrary amount of data as input and produce an output of fixed size. The output is always the same when given the same input. The inputs to a hash function are typically called messages, and the outputs are often referred to as message digests (RSA Laboratories). Nearly any piece of data can be defined as a message, including character strings, binary files and TCP packets. An example of a simple hash function would be the following:

Hash function H accepts messages of any length, and outputs a fixed length digest of one-bit. H returns 0 as the message digest if the input has an even number of characters, and returns 1 if the output has an odd number of characters.

All hash functions have the property that it is impossible to determine the input knowing only the output. In our example function, knowing that the output is 1 does not reveal any information about the input other than it has an odd number of digits. For example, if an attacker was given the fact that a message has a digest of "1", the original message could have been "102", "xqpr3", or any input of odd length. The attacker has no way of determining what the original message was by being given the digest. This property makes this hash functions a one-way function, meaning that it is difficult, if not impossible to deduce the input for a given output.

There are some hash functions which are much more powerful than the example given above; they are known as Cryptographic hash functions. Cryptographic hash functions have another property that most hash functions do not; the property that it is very difficult to find two different messages that produce the same message digest. Two distinct messages that result in the same digest are called collisions. In our example function, it is simple to create collisions. Our example above could not be considered a cryptographic hash function because it would be trivial to construct two inputs to this hash function that would create the same output, for example, both the inputs "101" and "32821" would have an output of 1, because they both have a length which is odd. In modern hash functions, it is so difficult to create collisions that there are no known efficient methods to produce them (RSA Laboratories).

Since different messages almost always produce different digests, one can conclude that if a message digest of a file changes, then the file itself has changed. This property can be used to provide data integrity and data authentication to a system administrator, as one will soon see.

2

2.1 Popular Hash Functions

There are two primarily cryptographic hash functions in use today, MD5 and SHA1.

MD5 stands for "Message Digest 5" because it is the fifth revision of a message digest algorithm devised by R.L. Rivest of RSA Laboratories (RSA Laboratories). The early revisions of this algorithm were published prior to 1989, and the most recent revision of the algorithm was published in 1991. It has an arbitrary input length and produces a 128-bit digest (Rivest). Although weaknesses have been found in the algorithm, there has never been a published collision.

SHA1 stands for "Secure Hash Algorithm 1", it is the first revision of a hash algorithm developed by the National Security Agency. The algorithm was first published in 1995 (Wikipedia). SHA1 supports messages of any length less than 2^{64} bits as input, and produces a 160-bit digest. In the unlikely event that one wishes to compute the digest of a message larger than 2^{64} bits in length (over 2 billion GB of information), the simplest solution would be to divide the large messages into smaller messages. There are no known weaknesses in SHA1, and it is generally considered the more secure of the two algorithms. There are also variations of SHA1 which produce longer digests, SHA-256, SHA-512. They produce digests of 256 bits and 512 bits, respectively (Eastlake).

The SHA1 and MD5 algorithms are considered secure because there are no known techniques to find collisions, except via brute force. In a brute force attack random inputs are tried, storing the results until a collision is found. If we do not limit ourselves to finding a collision with a specific message, one can expect to find a collision within $2^{n/2}$ computations, where n is the number of bits in the digest. (This is commonly known as the birthday attack, please see reference Krawczyk for more details). This means that an attacker would need to compute the digests of approximately 2^{64} messages to find a collision in the MD5 function, and approximately 2^{80} computations to find a collision in SHA1. Note that SHA1 may be more secure than MD5, but it is more costly to compute a message digest using SHA1 than MD5. If one is expressing security concerns SHA1 would be the function of choice, however, if speed is an issue it is likely that MD5 would result in faster performance, and would likely still be secure enough for most applications. In August 2001, a complex computing grid theorized by IBM was believed to be able to achieve 13.6 trillion calculations per second, which would make it one of the most powerful computers known (IBM Press Release). Even at this rate, assuming one computation of a digest per super computer calculation, it would take over 2800 years to find a collision in SHA1. In the unlikely event that a collision was ever found, security minded individuals could just use one of the SHA algorithms that produce larger outputs; these algorithms would require an even greater amount of time to find collisions in.

3

3.0 Data Integrity

Since two distinct messages are extremely unlikely to generate identical message digests, one can use this property of cryptographic hash functions to detect when a message has been altered. If one takes a binary file and computes a digest of the file, one can record this baseline digest. In the future, the digest can be recomputed on the file. If the new digest differs from the original baseline digest, then one can be assured that the file has been altered in some way (Sptizner). The only way that one could compute the digest of an altered file and have the digests match would be if one found a collision. Since collisions are extremely unlikely to occur, if the new digest matches the original digest, it is extremely likely that the file has not been altered. Therefore, we see that the properties of cryptographic hash functions can be used to verify that files have not been altered; one can quickly determine file integrity. Notice though that one cannot determine specifically what contents of the message have changed, only that something in the message has changed. For example, if an attacker were to alter bank account records, one could detect the change by seeing a changed digest, although one would not be able to determine which records were altered.

Note that using message digests to verify data integrity is not possible if an attacker is able to modify the place at which the digests are stored. An attacker could simply make an unauthorized change, compute the new digest for the file, and modify the digest database to include the new digest. A system administrator would not know the difference (unless a digest of the database itself was stored in an independent location unavailable to the attacker). One should always at a minimum password protect their digest database, or risk having their digests corrupted by a malicious user.

3.1 Tools to compute digests

Many tools exist and are readily available to system administrators that can be used to quickly compute the digests of files. Two simple tools that are included in most Linux distributions are md5sum and sha1sum. (A Windows port of md5sum is available at http://etree.org/md5com.html) Both programs are executed by typing md5sum <filename> or sha1sum <filename> at the command prompt and hitting return. The resulting message digest is displayed. In the exercise below, one computes the digest of a file, alters it, then recomputes the digest. One can then verify that the digest changes as well.

First, a new file is created. In this example, the file myfile.txt is created with the message "moo" within (Figure 1).
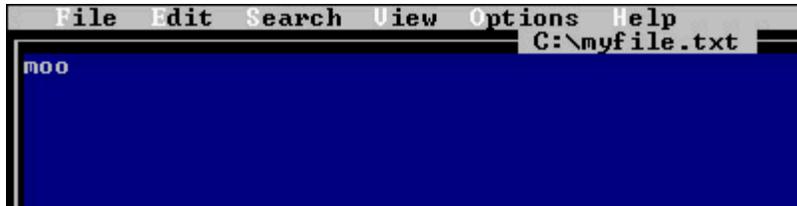
4

Figure 1

Now the digest of the file is computed using md5sum (Figure 2).


```
C:\>md5sum myfile.txt
77be7983a68edff00330f289d87d3a2c *myfile.txt
```
Figure 2

Now the file content is altered, "moo" is changed to "foo" (Figure 3). Note that a file is considered change with any addition or deletion of any character, including whitespace and case changes.

Figure 3

Now rerun the hash function programs on the same input file to get new digests. Notice that when the new digest is compared to the original, it is different (Figure 4).


```
C:\>md5sum myfile.txt
2145971cf82058b108229a3a2e3bff35 *myfile.txt
```
Figure 4

Remember this only shows you that the file was changed, not how it was changed. Digests are certainly not a substitute for backups.

There are many products that will take periodic digests of the files you specify and compare them to the previous digests. If they change, they have the ability to notify the system administrator of a problem. This is especially valuable for verifying the integrity of commonly used, but rarely changed files, such as ls or pwd. Such files are common targets of hackers and root kits (Prosise). One popular tool that automates the file integrity checking process is called Tripwire (developed by Tripwire, Inc.). Tripwire is available as both a commercial product and free open-source Linux project. Unfortunately, the configuration and usage of Tripwire exceeds the scope of this paper, but there is very informative article on the topic available at
http://www.linuxsecurity.com/feature_stories/feature_story-81.html (Lynch).

5

3.0 Data Authentication

Another application of cryptographic hash functions is data authentication. Data authentication is the process of being able to verify the source of data. With data authentication, one can distinguish messages originating from the intended sender and an attacker. Hash functions alone, unfortunately, cannot provide data authentication. Since the hashing functions are freely available, it is trivial to anyone, including an attacker, to create a digest for an arbitrary message. If one is given both a message and a digest, one can verify the integrity of the message. However, it does not necessarily mean that it was the message sent by the original sender. For example, if an email is sent with a message digest attached, the recipient could use the digest to verify the integrity of the message. However, it is possible that an attacker modified both the message and the digest. This change would be undetectable to the recipient. The point is illustrated in the example below:

Suppose Customer A sends a message to their bank, asking them to transfer 5 dollars from their checking to their savings account. Attacker A then blocks the transmission of Customer A's message, and creates one of their own stating to transfer 500 dollars from Customer A's checking account into Attacker A's account. Attacker A then computes the appropriate md5 checksum (something similar to b7ab99c9fc23453f77fb6bfef131bc07) for the fraudulent message and sends it to the bank. The bank could then verify that the data was not modified in transit, because the digest matches the message sent. However, the message did not originate from Customer A, the only one who is authorized to make transactions from their checking account.

This is a very common attack called forgery. If the bank simply verified the message digest matches the message, it can never be assured that the sender was actually Customer A. One would like a method by which the authenticity of the source of data can be verified. Fortunately, using cryptographic hash functions and secret key cryptography, this can be achieved.

3.1 Message Authentication Codes

Any time one sends a message masquerading as another user this is forgery, and as one can see from the above example, this is a very big problem. In order to prevent this type of attack, Message Authentication Codes were developed.

Message authentication codes are similar in usage to a message digest. By taking the message and performing some computations, one can verify the integrity of the data. Additionally, message authentication codes are also able to verify the source of data. Message authentication codes are specially created message digests that can be created only by the original sender.

6

In many instances, when two parties communicate they create a shared secret key known only to themselves. This shared key is used to encrypt data during the session. There are several techniques used to create this shared key without exposing it to an attacker, such as the Diffie-Hellman key exchange protocol. Unfortunately, the mechanics of such key exchange algorithms are outside the scope of this document (for more information, please consult Palmgren). If one assuming the two parties can safely create a secret key, this key can be used to generate message authentication codes. Using the simple algorithm below, one can see how when hash functions and secret keys are combined, data authentication is achieved.

One simple method would be to append the secret key to the message prior to performing the digest. This digest becomes the message authentication code, and it is sent to the recipient. In order to verify the source, the recipient would append the secret key to the received message and perform the digest. If the digest is the same as the sent authentication code, then both the integrity and the source of the data has been verified; because only the sender and recipient know the secret key, it is not possible for an attacker to generate a successful message authentication code (RSA Laboratories).

3.2 The HMAC scheme

A popular implementation of message authentication codes is the HMAC (Hash Message Authentication Code) scheme (Krawczyk). Although the algorithm described in the above section seems secure, it is actually susceptible to several attacks, such as the replay attack. The standard protocol for creating and verifying message authentication codes generated via hash functions has many methods for dealing with these attacks. This protocol in use today has come to be known as the HMAC algorithm. The HMAC (which stands for Hashing Message Authentication Codes) algorithm is defined in RFC 2085 and was developed by NIST researchers in 1997 (Oehler). The use of HMAC is very common in any system where messages require authenticity of source. Many secure Internet protocols use HMAC to provide authenticity of data, including some variations of IPSec (Frankel).

3.3 Replay Attacks thwarted by HMAC

One has already seen that message authentication codes such as HMAC prevent data forgery; that is it detects when messages are sent by anyone other than the original sender. There is another type of attack that is particularly worrisome, the replay attack (Oehler).

An attacker may not be able to successfully create a message authentication code for a new message. However, an attacker has likely viewed previously valid message authentication codes in transit. Imagine this scenario:

7

Attacker A is an Internet merchant selling books on cryptography. Whenever a purchase is made, he watches the messages that are sent to the bank to authorize the bank to transfer money from the customer's account into his own. The attacker has now seen a valid message (transfer money from his account to my account) and the associated authentication code. The attacker can then send this message, along with its valid authentication code repeatedly, eventually transferring the customer's entire account into his own.

HMAC prevents this type of attack by appending a form of timestamp to each message (Oehler). The recipient can then verify that the message has not been previously received. If it is truly the case where multiple messages of the same type are sent, then the new timestamp will differentiate the messages. Note that the mathematics behind the HMAC algorithm are extremely complex and not as straightforward as presented above. They are presented above in simpler form for the sake of simplicity. If one is interested in the full details of the algorithm, one should consult the RFC (Oehler).

3.4 HMAC in Java

If one wishes to use the HMAC system in a programming project, there is a reference implementation included in the Java programming language v1.4 (Sun Microsystems). The HMAC algorithm is a vital component in the Java Secure Sockets Extension libraries; whenever Java secure sockets are used in an application, the HMAC scheme is providing authentication of data while it traverses the network. HMAC is available with either MD5 or SHA1 as the underlying hash algorithm.

4.0 Encryption vs. Integrity and Authentication

Many believe the related field of encryption can be used to provide the same benefits as hash functions, such as file integrity, because if someone were able to modify the data it will be obvious to the person after the file is unencrypted. Unfortunately, in many cases it is difficult, if not impossible to see these corruptions in the file. Suppose the file contained a random bit string; any change would not be visible to the user. Digests afford another luxury that encryption does not, which is that the verification method can be made publicly available. If one uses encryption to perform file integrity checks, only one who knows the key to decrypting the file can determine its integrity. Therefore, if one wishes the integrity of a file to be publicly verifiable, they must divulge their decryption key, a large breach of security to say the least. However, with message digests, the digest can be publicly distributed, and anyone able to compute a message digest of the same type can verify the integrity of the file. This verification can come independently of the file being encrypted or not.

Hash functions also have another property that encryption algorithms do not; this property is known as "transient" effect (Krawczyk). What this means is that past

8

integrity and authentication of data is always valid. If in the future, a hash function is proven flawed, then all data that was verified prior to this discovery of the flaw still maintains its integrity. However, if in the future an encryption algorithm is found to be flawed, then all messages encrypted using that algorithm can be decrypted. The primarily goal of encryption, data secrecy, is compromised. Hash functions, on the other hand, maintain their past integrity.

5.0 Conclusion

Clearly, the properties of cryptographic hash functions have many applications in the realm of computer security, and programs built on top of cryptographic hash functions have the ability to help a system administrator detect changes of valuable data on his or her network. They also are able to prove the originator of messages in a system. These concepts are particularly relevant in the growing online world, where every message sent across the wire can be worth money, and every file on a server is a valuable resource. Without safeguards such as those afforded by hash functions, data would be extremely vulnerable to attack. Now that the system administrator is aware of the issues that exist, they can make an informed decision when using and purchasing technologies to protect data. Every application must be scrutinized with respect to the integrity and authentication checks it performs, and it must use the latest hash functions to guarantee security. The system administrator now understands that simply encrypting data is not enough, and other precautions must be taken. Customers and employees demand these safeguards in our unsure digital world where our data is constantly coming under attack from hackers and malicious insiders.

9

6.0 Sources and References

Eastlake, Motorola, Jones. "RFC 3174 - US Secure Hash Algorithm 1 (SHA1)", September 2001. URL: http://www.faqs.org/rfcs/rfc3174.html

Frankel, S. "Internet Draft - The HMAC-SHA-256-128 Algorithm and Its Use With Ipsec", June 2002. URL: http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ciph-sha-256-01.txt

IBM Press Release. "IBM Selected to Build World's Most Powerful Computing Grid", August 2001. URL: http://www-916.ibm.com/press/prnews.nsf/jan/7613B7AF8EA527D385256AA3006EC06B

Krawczyk, Bellare, Canetti. "RFC 2104- HMAC: Keyed-Hashing for Message Authentication", February 1997. URL: http://www.ietf.org/rfc/rfc2104.txt

Lynch, William. "Getting Started with Tripwire (Open Source Linux Edition)", March 2001. URL: http://www.linuxsecurity.com/feature_stories/feature_story-81.html.

Oehler, Glenn. "RFC 2085 - HMAC-MD5 IP Authentication with Replay Prevention", February 1997. URL: http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2085.html

Palmgren, Keith. "Diffie-Hellman Key Exchange - A Non-Mathematician's Explanation", October 2000. URL: http://networking.earthweb.com/netsecur/article.php/624441

Prosise, Chris; Shahm Saumil. "Anatomy of a Hack", January 2001. URL: http://dotphoto.cnet.com/webbuilding/0-7532-8-4561014-2.html

Rivest, R. "RFC 1321 - The MD5 Message-Digest Algorithm", April 1992, URL: http://www.cis.ohio-state.edu/rfc/rfc1321.txt

RSA Laboratories. "What are MD2, MD4, and MD5?", Date Unknown. URL: http://www.rsasecurity.com/rsalabs/faq/3-6-6.html.

RSA Laboratories. "What is a hash function?", Date Unknown. URL: http://www.rsasecurity.com/rsalabs/faq/2-1-6.html

RSA Laboratories. "What are Message Authentication Codes?" Date Unknown. URL: http://www.rsasecurity.com/rsalabs/faq/2-1-7.html

Sptizner, Lance. "What is MD5, and why do I care?", Date Unknown. URL: http://www.spitzner.net/md5.html

10

Sun Microsystems. "Java^{TM} Secure Socket Extension (JSSE) Reference Guide",
2001. URL:
http://java.sun.com/j2se/1.4.1/docs/guide/security/jsse/JSSERefGuide.html

Wikipedia. "SHA-1", March 2002. URL: http://www.wikipedia.org/wiki/SHA-1

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Minneapolis 2018	**Minneapolis, MNUS**	**Jun 25, 2018 - Jun 30, 2018**	**Live Event**
SANS Paris June 2018	**Paris, FR**	**Jun 25, 2018 - Jun 30, 2018**	**Live Event**
SANS Vancouver 2018	**Vancouver, BCCA**	**Jun 25, 2018 - Jun 30, 2018**	**Live Event**
SANS London July 2018	**London, GB**	**Jul 02, 2018 - Jul 07, 2018**	**Live Event**
SANS Cyber Defence Singapore 2018	**Singapore, SG**	**Jul 09, 2018 - Jul 14, 2018**	**Live Event**
SANS Charlotte 2018	**Charlotte, NCUS**	**Jul 09, 2018 - Jul 14, 2018**	**Live Event**
SANSFIRE 2018	**Washington, DCUS**	**Jul 14, 2018 - Jul 21, 2018**	**Live Event**
SANS Cyber Defence Bangalore 2018	**Bangalore, IN**	**Jul 16, 2018 - Jul 28, 2018**	**Live Event**
SANS Pen Test Berlin 2018	**Berlin, DE**	**Jul 23, 2018 - Jul 28, 2018**	**Live Event**
SANS Riyadh July 2018	**Riyadh, SA**	**Jul 28, 2018 - Aug 02, 2018**	**Live Event**
SANS Pittsburgh 2018	**Pittsburgh, PAUS**	**Jul 30, 2018 - Aug 04, 2018**	**Live Event**
Security Operations Summit & Training 2018	**New Orleans, LAUS**	**Jul 30, 2018 - Aug 06, 2018**	**Live Event**
SANS August Sydney 2018	**Sydney, AU**	**Aug 06, 2018 - Aug 25, 2018**	**Live Event**
SANS Hyderabad 2018	**Hyderabad, IN**	**Aug 06, 2018 - Aug 11, 2018**	**Live Event**
SANS San Antonio 2018	**San Antonio, TXUS**	**Aug 06, 2018 - Aug 11, 2018**	**Live Event**
SANS Boston Summer 2018	**Boston, MAUS**	**Aug 06, 2018 - Aug 11, 2018**	**Live Event**
Security Awareness Summit & Training 2018	**Charleston, SCUS**	**Aug 06, 2018 - Aug 15, 2018**	**Live Event**
SANS New York City Summer 2018	**New York City, NYUS**	**Aug 13, 2018 - Aug 18, 2018**	**Live Event**
SANS Northern Virginia- Alexandria 2018	**Alexandria, VAUS**	**Aug 13, 2018 - Aug 18, 2018**	**Live Event**
Data Breach Summit & Training 2018	**New York City, NYUS**	**Aug 20, 2018 - Aug 27, 2018**	**Live Event**
SANS Virginia Beach 2018	**Virginia Beach, VAUS**	**Aug 20, 2018 - Aug 31, 2018**	**Live Event**
SANS Krakow 2018	**Krakow, PL**	**Aug 20, 2018 - Aug 25, 2018**	**Live Event**
SANS Prague 2018	**Prague, CZ**	**Aug 20, 2018 - Aug 25, 2018**	**Live Event**
SANS Chicago 2018	**Chicago, ILUS**	**Aug 20, 2018 - Aug 25, 2018**	**Live Event**
SANS San Francisco Summer 2018	**San Francisco, CAUS**	**Aug 26, 2018 - Aug 31, 2018**	**Live Event**
SANS Copenhagen August 2018	**Copenhagen, DK**	**Aug 27, 2018 - Sep 01, 2018**	**Live Event**
SANS SEC504 @ Bangalore 2018	**Bangalore, IN**	**Aug 27, 2018 - Sep 01, 2018**	**Live Event**
SANS Amsterdam September 2018	**Amsterdam, NL**	**Sep 03, 2018 - Sep 08, 2018**	**Live Event**
SANS Tokyo Autumn 2018	**Tokyo, JP**	**Sep 03, 2018 - Sep 15, 2018**	**Live Event**
SANS Wellington 2018	**Wellington, NZ**	**Sep 03, 2018 - Sep 08, 2018**	**Live Event**
SANS Tampa-Clearwater 2018	**Tampa, FLUS**	**Sep 04, 2018 - Sep 09, 2018**	**Live Event**
SANS MGT516 Beta One 2018	**Arlington, VAUS**	**Sep 04, 2018 - Sep 08, 2018**	**Live Event**
SANS Cyber Defence Canberra 2018	**OnlineAU**	**Jun 25, 2018 - Jul 07, 2018**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**