Analyzing Digital Signature Robustness with Message Digest Algorithms

Rubina B. Patel
Department of CSE
CTAE, MPUAT
Udaipur, India

Naveen Chaudhary
Department of CSE
CTAE, MPUAT
Udaipur, India

ABSTRACT
A hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, called the message digest, such that any change to the data will change the digest value. The message digest are being extensively used for digital signature for online transactions. In this paper we have analyzed the robustness of various message digest with respect to digital signature. The repetitive sequences in the digest will decrease the robustness of the message digest as it can lead to same message digest for differing message blocks. In this paper the robustness of message digest is analyzed with respect to sequence repetitions as digest with more repetitive sequence are more likely or have increased probability of generating the same digest for differing message blocks. We have analyzed the robustness of various popular message digest algorithms such as MD5, SHA1, RIEMD160, PANAMA, and TIGER.

Keywords
Message Digest, Hash function, Digital Signature, Secure communication, Information security

1. INTRODUCTION
A hash code does not use a key but is a function only of the input message. All hash functions operate using the following general principles. The input is viewed as a sequence of n-bit blocks, processed one block at a time in an iterative fashion to produce an n-bit digest. A message digest algorithm must be able to withstand all the known types of cryptanalytic attack. As a minimum, it must have the following properties. [1]

Preimage resistance - Given a hash *h* it should be difficult to find any message m such that *h=hash (m)*. In other words we can say it is a one-way function i.e. a function that is easy and quick to compute the digest of a message but if given the digest it should be impossible to derive the message.

Second preimage resistance - Given an input *m1* it should be difficult to find another input *m2* — where $m1 \neq m2$ — such that *hash (m1) = hash (m2).* That means two different messages must not result in same digest.

Collision resistance - It should be difficult to find two different messages *m1* and *m2* such that *hash (m1) =hash (m2).* Such a pair is called a cryptographic hash collision. The birthday "paradox" places an upper bound on collision resistance: if a hash function produces *N* bits of output, an attacker who computes only $2^{N/2}$ hash operations on random input is likely to find two matching outputs. If there is an easier method than this brute force attack, it is typically considered a flaw in the hash function [1].

In cryptography 'hard' means almost certainly beyond the reach of any adversary who must be prevented from breaking the system for as long as the security of the system is deemed important. In other words 'hard' is impossible to break in one's life even using all computing facility in the world.

These properties make hash functions useful in cryptography and other applications as they allow the representation of objects in a known fixed size.

The hash result provides a unique imprint of a message, and that the protection of a short imprint is easier than the protection of the message itself.

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 [2].

There are various message digest algorithms, although many have been found to be vulnerable and should not be used. Even if a hash function has never been broken, a successful attack against a weakened variant may challenge the expert's confidence and lead to its abandonment.

As of today, the two most popularly used cryptographic hash functions are MD5 and SHA-1. However, to ensure the long-term robustness of applications that use hash functions, there is a need to extensively test the robustness of the chosen hash function.

2. MESSAGE DIGEST HASH FUNCTION
In modern society information has become a valuable commodity. It is important to protect the authenticity of information. This has two aspects: it should be possible to check who the author is and information has not been modified by anyone. Message digest algorithms takes inputs of arbitrary length and produce as output a short string of bits. Their most important use is for the protection of data authenticity, but they are a versatile building block and are also used in conjunction with digital signature schemes, in addition to other commonly used applications such as password protection and pseudo-random string generation.

2.1 Applications
2.1.1 Digital signatures
Signatures are used to meet the needs of document recipients to verify that document is authentic, unforgeable and non-repudiable.

The process of digitally signing, starts by taking a mathematical summary (called a hash code) of the message. This hash code is a uniquely-identifying digital fingerprint of the message. If even a single bit of the message changes, the

hash code will dramatically change. The next step in creating a digital signature is to sign the hash code with your private key. This signed hash code is then appended to the message [2].

Well, the recipient of your message can verify the hash code sent by you, using your public key. At the same time, a new hash code can be created from the received message and compared with the original signed hash code. If the hash codes match, then the recipient has verified that the message has not been altered. The recipient also knows that only you could have sent the check because only you have the private key that signed the original hash code [2].

2.1.2 Integrity verification

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.

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 [3].

2.1.3 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.

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. If one assuming the two parties can safely create a secret key, this key can be used to generate message authentication codes [3].

2.2 MD5

MD5 Message-Digest Algorithm designed by Ron Rivest produces a 128-bit (16-byte) hash value. An MD5 hash is expressed as a 32-digit hexadecimal number. The MD5 algorithm is designed to be quite fast on 32-bit machines [4].

The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit little endian integers); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64-bit little endian integer representing the length of the original message, in bits [4].

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *A, B, C* and *D* also called the chaining variables. These are initialized to fixed constants, i.e. A-01234567, B-89ABCDEF, C- FECDBA98, and D-76543210.

Also an array of constant K which contains 64 elements each of 32 bits is initialized. The main algorithm then operates on each 512-bit message block in turn, each block modifying the variables. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a non-linear function F, modular addition, one with sub-block of message and one with a 32 bit constant, and left rotation. The size of the digest i.e.128 bits is small enough to think of a birthday attack [4].

Following is an instance showing, a 33-byte ASCII input and the corresponding 32 digit MD5 digest:

Text - There is CHF1500 in the blue box.

Digest- 717478e39090cb4d19dc4c6743b0acc5

2.3 SHA-1

SHA-1 produces a 160-bit (20 byte) message digest. Although slower than MD5, this larger digest size makes it stronger against brute force attacks.

The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by US government standards agency NIST (National Institute of Standards and Technology). This version is now often referred to as SHA-0. It was superseded by the revised version, published in 1995 in FIPS PUB 180-1 and commonly referred to as SHA-1. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function; this was done, according to NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. [1]

SHA-1 consists of 4 rounds, each containing 20 iterations (i.e. 80 iterations all in all). The algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, D and E, i.e. 5 variables as compared to 4 variables in MD5. These are initialized to fixed constants, i.e. A-01234567, B-89ABCDEF, C- FECDBA98, D-76543210, and E-C3D2E1F0. An array of constant K[t] which contains 78 elements each of 32 bits is initialized which have only four constants, one used in each of four rounds.

Following is an instance showing, a 33-byte ASCII input and the corresponding 40 digits SHA-1 digest:

Text - There is CHF1500 in the blue box.

Digest- 3d4cb01cdbef5ed1b9f5b94b3a628655050f7484

2.4 RIPEMD160

RIPEMD-160 is a 160 bit digest algorithm whose digest are represented as 40-digit hexadecimal numbers. There also exist 128, 256 and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320. [5]

Its primitive operations are: Left-rotation of words; Bitwise Boolean operations (AND, NOT, OR, exclusive-OR); Two's complement modulo 232 addition of words [5].

RIPEMD-160 compresses an arbitrary size input string by dividing it into blocks of 512 bits each. Each block is divided into 16 strings of 4 bytes each, and each such 4-byte string is converted to a 32-bit word using the little-endian convention, which is a.o. used on the Intel 80x86 architecture; MD4, MD5

and RIPEMD use the same convention, while SHA-1 uses the big-endian convention [5].

In order to guarantee that the total input size is a multiple of 512 bits, the input is padded in the same way as for all the members of the MD4-family.The result of RIPEMD-160 is contained in five 32- bit words, which form the internal state of the algorithm. The final content of these five 32-bit words is converted to a 160-bit string, again using the little-endian convention [5].

This state is initialized with a fixed set of five 32-bit words, the initial value. The main part of the algorithm is known as the compression function: it computes the new state from the old state and the next 16-word block. The compression function consists of 5 parallel rounds, each containing 16 steps. The total number of steps is thus 5 ×16× 2 = 160. RIPEMD-160 is about 15% slower than SHA-1 and four times slower than MD4. On a big-endian RISC machine, the difference between SHA-1 and RIPEMD-160 will be slightly larger [5].

Following is an instance showing, a 33-byte ASCII input and the corresponding 40 digit RIPEMD-160 hash:

Text - There is CHF1500 in the blue box.

Digest- 6372e0dc709f9e70d9ac9ac4723be7c944420927

2.5 Panama

Panama is a cryptography primitive which can be used both as a hash function and a stream cipher.

Panama contains two main elements: A shift register, with 32 cells, each containing a vector with eight 32-bit words, and a re-circulating mixing function, resembling the f-function in a block cipher, which operates on a "state" consisting of seventeen 32-bit words.[6]

There are three fundamental operations that form part of Panama.[7]

- Panama is reset by setting both the 17-word state and the contents of the shift register to all zeroes.
- A vector of eight 32-bit words is fed to Panama through a Push operation. Operations unique to the Push function are shown by the light dotted lines in the diagram. In a Push operation, the incoming vector is used as one of the inputs to the state transition function, and is also used to XOR with the re-circulating values in the shift register.
- A vector of eight 32-bit words is received from Panama by means of a Pull operation. In a Pull operation, the 32-bit words numbered 9 through 16 in the state are used as the output, and words 1 through 8 are XORed with the re-circulating values in the shift register. The inputs to the state transition function both come from stages in the shift register, one not used for any special purpose in the Push operation replacing the input, absent from a Pull operation

When Panama is used as a hash function, the message to be hashed, followed by a 1 bit and as many zeroes, are needed to cause the message to occupy an integer number of 256-bit blocks, is input to Panama through a series of Push operations.

Then, after a number of Pull operations with their output discarded, so that the effects of even the last block of the message are fully diffused, the output from a final Pull operation constitutes the hash. The state transition function of Panama operates on seventeen, 32-bit words, numbered 0 through 16. [7]

Following is an instance showing, a 33-byte ASCII input and the corresponding 64 digit PANAMA digest:

Text - There is CHF1500 in the blue box.

Digest-
24d2ef5e157855374196bef71c9d2f99be239eebd7ed44
4262e5447796cdfbf8

2.6 Tiger

Tiger is a cryptographic hash function designed in 1995 for efficiency on 64-bit platforms. The size of a Tiger hash value is 192 bits.

It is strong and fast: as fast as SHA1 on 32-bit processor, and about three times faster on 64-bit (DEC Alpha) processor. It is also expected to be faster than SHA1 on 16 bit processor, since SHA1 is optimized for 32-bit machines, while Tiger is designed to work adequately on many word sizes [8].

In Tiger all the computations are on 64-bit words, in little-endian/2-compliment representation. We use three 64-bit registers called a, b, and c as the intermediate hash values. These registers are initialized to h_0 which is: a = 0x0123456789ABCDEF, b = 0xFEDCBA9876543210, c = 0xF096A5B4C3B2E187 [8].

The core of Tiger basically has core is three rounds, each of which uses eight lookups into 8-to-64 bit S-boxes to provide a strong nonlinear avalanche plus a number of register operation to increase diffusion and make differential attacks harder [8].

Following is an instance showing, a 33-byte ASCII input and the corresponding 48 digit TIGER digest:

Text - There is CHF1500 in the blue box.

Digest-
24d2ef5e157855374196bef71c9d2f99be239eebd7ed44
4262e5447796cdfbf8

3. EXPERIMENTAL RESULTS

We have used HashCalc tool [9] to produce different digests from different algorithms. It supports 12 well-known and documented hash and checksum algorithms: MD2, MD4, MD5, SHA1, SHA256, SHA384, SHA512, RIPEMD160, PANAMA, TIGER, ADLER32, and CRC32.From these we have used 5 popular algorithms such as MD5, SHA1, RIPEMD160, PANAMA, and TIGER.

The experimentation is done on 4 different data values: 1) Text strings of size 32 character. 2) Hexadecimal strings of size 16 character. 3) Text files of size 17 bytes 4) Image files of size 606 KB.

The size of hash basically gives the strength of the hash function, as longer the digest, it is believed that it will be harder to crack it or in other words it will be more difficult to find a different message (m') whose hash is same as that of the original message (m).

However here in this work we are trying to establish the strength of a message digest with respect to its size for constant size digest the strength of the digest is mainly governed by how many unique characters it includes in the digest, as more repetitive character signify that more different messages are likely to produce the similar looking message digest.

Based on the above mentioned hypothesis the experimented results for various popular message digest functions such as

MD5, SHA1, RIPEMD160, PANAMA, and TIGER are presented in figure 1, with respect to digest size.

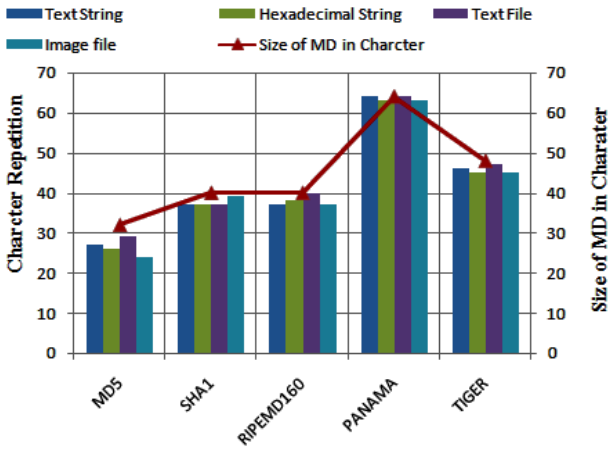

Fig 1: Graph showing number of character repeated in digest of different algorithms

Figure1 cleanly show that MD5 has character repetition percentage of 82.8125, which has minimum characters repeated of all the algorithms analyzed.

SHA1 has character repetition percentage of 93.75. RIPEMD-160 has character repetition percentage of 95. Although both (SHA1 and RIPEMD-160) have output of 40 digits still SHA-1 proves to be a better algorithm in terms of repeated character.

PANAMA has character repetition percentage of 99.21875, which is the worst of all, and almost every character of the digest is repeated. TIGER has character repetition percentage of 95.31225, which is an average value between the extreme cases algorithms.

The MD5 with its limited digest size shows that it performs best as far as avoiding character repletion is concerned. In other words we can say that with limited size constraints the MD5 function as such has good mathematical properties to generate unique digest for similar looking messages.

4. CONCLUSION

On the basis of experimental results we can conclude that MD5 has the least percentage of repeated character, and so contains good mathematical properties to produce differing message digest for similar looking messages even with the constraint of limited digest size.

The conclusion drawn from this work can be helpful in designing more robust message digest functions for digital signature or for further strengthening the existing hash function with large hash size by including the mathematical design properties of MD5 as MD5 even with its size constraint encompasses many good properties to generate unique digests.

5. REFERENCES

[1] Kahate, Atul 2003. Cryptography and Network Security. Second edition. Tata McGraw-Hill Pvt.

[2] Curry, Ian March 2001 An Introduction to Cryptography and Digital Signatures. Version 2.0

[3] Silva, John Edward January 15, 2003 An Overview of Cryptographic Hash Functions and Their Uses. GIAC Security Essentials Practical. Version 1.4b. Option 1

[4] Rivest, R. April 1992. The MD5 Message-Digest Algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc.

[5] CryptoBytes, autumn 1997. The technical newsletter of RSA Laboratories. A division of RSA Data Security, Inc RSA Laboratories Volume 3, Number 2.

[6] Rompay, Bart Van, June 2004. Analysis and Design of Cryptographic Hash functions, Mac algorithms and Block ciphers"

[7] http:// www.quadibloc.com/ crypto/co4821.htm

[8] Ross, Anderson and Eli, Biham, Tiger: A Fast new hash function" by

[9] http://www.slavasoft.com/?source=HashCalc.exe