Cryptographic of high Security Hash Functions

C. Krishna Kumar[1], Dr. C. Suyambulingom[2]
[1,] Sathyabama University, Chennai, India, [2]Professor (Rtd.), Dept. of Mathematics, TAU, Coimbatore, India

ABSTRACT

Cryptographic hash functions are an important tool in cryptography to achieve certain security goals such as authenticity, digital signatures, digital time stamping, and entity authentication. They are also strongly related to other important cryptographic tools such as block ciphers and pseudorandom functions. This paper aims to overview the cryptographic hash functions and its strength against attacks.

Keywords- hash functions, Brute-Force Attacks, Cryptanalysis, Birthday Attacks

1. Introduction

Hash functions are functions that compress an input of arbitrary length to a result with a fixed length. If hash functions satisfy additional requirements, they are a very powerful tool in the design of techniques to protect the authenticity of information.

A hash value h is generated by a function H of the form h = H(M) where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value. Figure 1 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

a. The message plus concatenated hash code is encrypted using symmetric encryption. This is identical in structure to the internal error control strategy shown in Figure 2. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality. Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC. That is, E(K, H(M)) is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.

c. Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (Figure 1.b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

d. If confidentiality as well as a digital signature is desired, then the message

plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

e. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

f. Confidentiality can be added to the approach of (Figure 1.e) by encrypting the entire message plus the hash code.

Fig. 1 Basic Uses of Hash Function

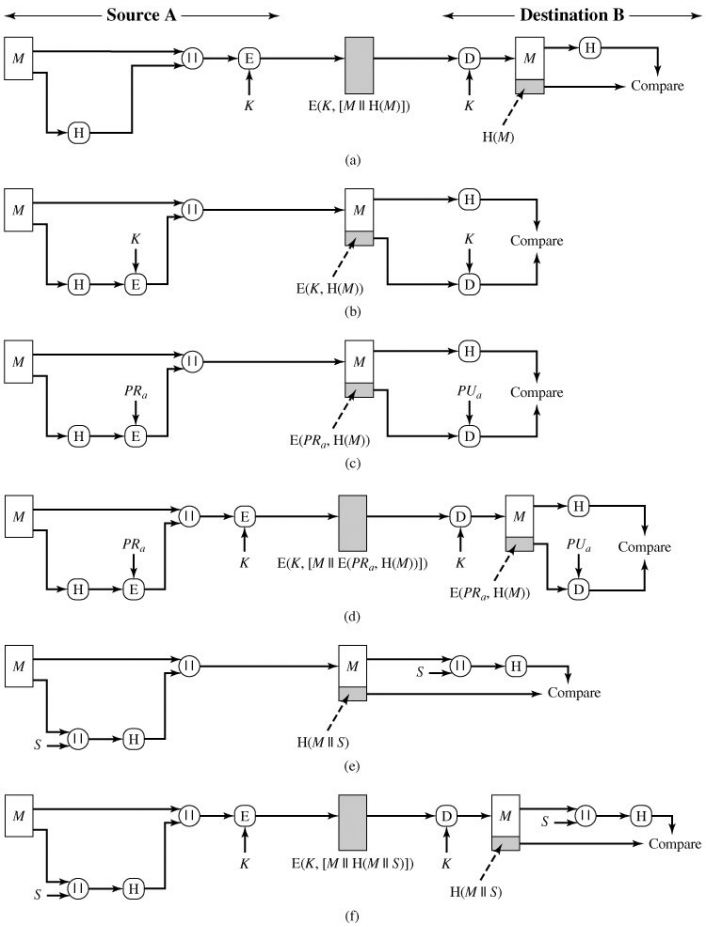
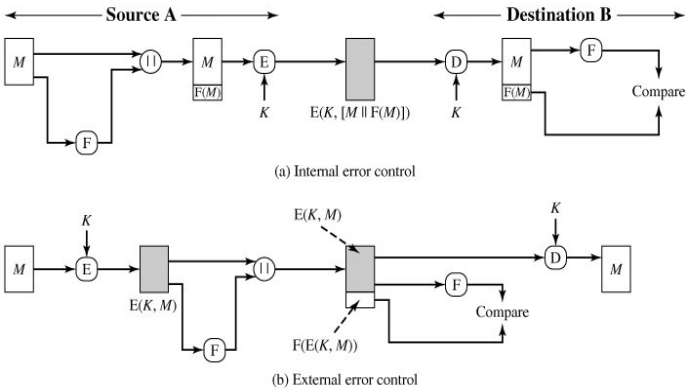


Fig. 2 Internal and External Error Control



2. Requirements for a Hash Function

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as the one-way property.
5. For any given block x, it is computationally infeasible to find $y \neq x$ such that H(y) = H(x). This is sometimes referred to as *weak collision resistance*.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as *strong collision resistance*.

The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value

(Figure 1. e). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message M and the hash code C = $H(S_{AB}||M)$. The attacker then inverts the hash function to obtain $S_{AB}||M = H(C)$. Because the attacker now has both M and $S_{AB}||M$, it is a trivial matter to recover S_{AB}.

The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used (Figures 1.b and 1.c). For these cases, the opponent can read the message and therefore generate its hash code. However, because the opponent does not have the secret key, the opponent should not be able to alter the message without detection. If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack.

3. Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as follows:

$$C_i = b_{i1} \oplus b_{i1} \oplus \ldots \oplus b_{im}$$

where

C_i = i^{th} bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = i^{th} bit in jth block

$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^n. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{128}, the hash function on this type of data has an effectiveness of 2^{112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:
    a. Rotate the current hash value to the left by one bit.
    b. XOR the block into the hash value.

This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input. Figure 3 illustrates these two types of hash functions for 16-bit hash values.

Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message, as in Figures 1.b and 1.c. Given a message, it is an easy matter to produce a new message that yields that hash code:

Simply prepare the desired alternate message and then append an n-bit block that forces the new message plus block to yield the desired hash code.

Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message as well as the hash code are encrypted (Figure 1.a). But you must be careful. A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode. We can define the scheme as follows: Given a message consisting of a sequence of 64-bit blocks X_1, X_2,..., X_N, define the hash code C as the block-by-block XOR of all blocks and append the hash code as the final block:

$$C=X_{N+1}=X_1 \oplus X_2 \oplus \ldots \oplus X_N$$

Next, encrypt the entire message plus hash code, using CBC mode to produce the encrypted message Y_1, Y_2,..., Y_{N+1}. [Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." IEEE Communications Magazine, September 1988.] points out several ways in which the ciphertext of this message can be manipulated in such a way that it is not detectable by the hash code. For example, by the definition of CBC, we have

$$X_1 = IV \oplus D(K, Y_1)$$

$$X_i = Y_{i1} \oplus D(K, Y_i)$$

$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

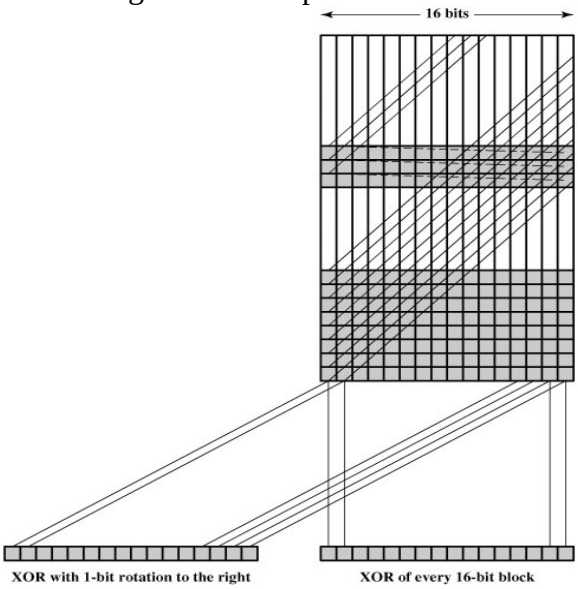
But X_{N+1} is the hash code:

$$X_{N+1} = X_1 \oplus X_2 \oplus \ldots \oplus X_N$$

$$= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \ldots \oplus [Y_{N1} \oplus \ldots \oplus D(K, Y_N)]$$

Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted.

Fig. 3 Two Simple Hash Functions



4. Security of Hash Functions

We can group attacks on hash functions into two categories: brute-force attacks and cryptanalysis.

4.1. Brute-Force Attacks

The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

- One-way: For any given code h, it is computationally infeasible to find x such that H(x) = h.
- Weak collision resistance: For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).
- Strong collision resistance: It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

For a hash code of length n, the level of effort required, as we have seen is proportional to the following:

One way	2^n
Weak collision resistance	2^n

Strong collision resistance	$2^{n/2}$

If strong collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value $2^{n/2}$ determines the strength of the hash code against brute-force attacks. Oorschot and Wiener presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. However, even 160 bits is now considered weak.

4.2. Cryptanalysis

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, indicated in Figure 4. This structure, referred to as an iterated hash function, was proposed by Merkle and is the structure of most hash functions in use today. The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

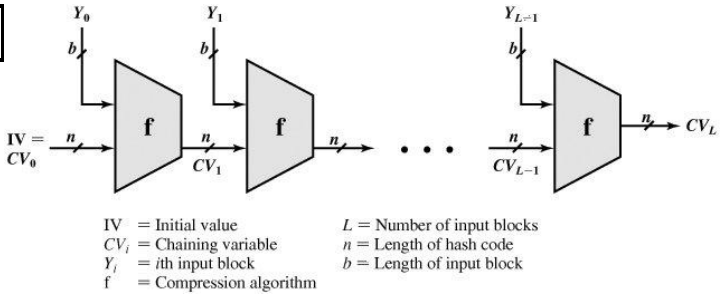


Fig. 4 General Structure of Secure Hash Code

The hash algorithm involves repeated use of a *compression function*, f, that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, b > n; hence the term compression. The hash function can be summarized as follows:

CV_o = IV = initial n-bit value

CV_i = $f(CV_{i1}, Y_{i1})$ $1 \le i \le L$

$H(M)$ = CV_L

where the input to the hash function is a message M consisting of the blocks Y_o, Y_1,..., Y_{L1}.

The motivation for this iterative structure stems from the observation by Merkle and Damgard that if the compression function is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size. The converse is not necessarily true.

Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f. Once that is done, the attack must take into account the fixed value

of IV. The attack on f depends on exploiting its internal structure. Typically, as with symmetric block ciphers, f consists of a series of rounds of processing, so that the attack involves analysis of the pattern of bit changes from round to round.

Keep in mind that for any hash function there must exist collisions, because we are mapping a message of length at least equal to twice the block size b (because we must append a length field) into a hash code of length n, where $b \geq n$. What is required is that it is computationally infeasible to find collisions.

5. Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted message M (Figure 1.b or 1.5c), then an opponent would need to find an M' such that H(M') = H(M) to substitute another message and fool the receiver. On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message.

However, a different sort of attack is possible, based on the birthday paradox. Yuval proposed the following strategy:

1. The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key (Figure 1.c).
2. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the real one.
3. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
4. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32}.

The generation of many variations that convey the same meaning is not difficult. For example, the opponent could insert a number of "space-space-backspace" character pairs between words throughout the document. Variations could then be generated by substituting "space-backspace-space" in selected instances. Alternatively, the opponent could simply reword the message but retain the meaning.

CONCLUSION

As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack. That is, an ideal hash algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort. The attacks that have been mounted on hash functions are rather complex and beyond our scope.

REFERENCES

1. van Oorschot, P., and Wiener, M. "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms." Proceedings,

Second ACM Conference on Computer and Communications Security, 1994.
2. I. Bellefroid and K. Beyen, "Evaluatie van de Cryptografische Veiligheid van Anti-Virus Paketten (Evaluation of the Security of Anti-Virus Software – in Dutch)," ESAT Laboratorium, Katholieke Universiteit Leuven, Thesis grad. eng., 1992.
3. Meyer, C., and Schilling, M. "Secure Program Load with Modification Detection Code." Proceedings, SECURICOM 88, 1988.
4. Nechvatal, J. "Public Key Cryptography."
5. Simmons, G., ed. Contemporary Cryptology: The Science of Information Integrity. Piscataway, NJ: IEEE Press, 1992.
6. Menezes, A.; van Oorschot, P.; and Vanstone, S. Handbook of Applied Cryptography. Boca Raton, FL: CRC Press, 1997.
7. Damgard, I. "A Design Principle for Hash Functions." Proceedings, CRYPTO '89, 1989; published by Springer-Verlag.
8. C.H. Meyer and S.M. Matyas,"Cryptography: a New Dimension in Data Security," Wiley & Sons, 1982.
9. C.M. Adams and S.E. Tavares, "The structured design of cryptographically good S-boxes," Journal of Cryptology, Vol. 3, No. 1, 1990, pp. 27–41.
10. Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." IEEE Communications Magazine, September 1988.
11. Merkle, R. "One Way Hash Functions and DES." Proceedings, CRYPTO '89, 1989; published by Springer-Verlag.
12. Dobbertin, H. "The Status of MD5 After a Recent Attack." CryptoBytes, Summer 1996.
13. S. Babbage, "On the relevance of the strict avalanche criterion," Electronic Letters, Vol. 26, No. 7, 1990, pp. 461–462.
14. Yuval, G. "How to Swindle Rabin." Cryptologia, July 1979.
15. Davies, D., and Price, W. Security for Computer Networks. New York: Wiley, 1989.
16. Rabin, M. "Digitalized Signatures." In Foundations of Secure Computation, DeMillo, R.; Dobkin, D.; Jones, A.; and Lipton, R., eds. New York: Academic Press, 1978.
17. Bellare, M., and Rogaway, P. "Collision-Resistant Hashing: Towards Making UOWHF's Practical." Proceedings, CRYPTO '97, 1997; published by Springer-Verlag.