Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

9th International Conference
on Electrical Engineering
ICEENG 2014

CHA: Concrete Hash Algorithm
A NEW PROPOSED HASH FUNCTION ALGORITHM

By

Hadia M. El Hennawy [1] *Alaa E. Omar* [2] *Salah M. Kholaif* [2]

Abstract:

Cryptographic hash functions are an important tool in cryptography for applications such as digital signature of messages, message authentication, and key derivation. This paper presents a new cryptographic keyed hash function design, named Concrete Hash function Algorithm, CHA-128, which has some difference from the popular hash function designs and used for ensuring integrity of messages to a higher degree to suit the fast growing network applications. The function structure is designed permit a wide variety of implementation tradeoffs. Also CHA is proposed as very smooth 128 bit hash function algorithm, which is designed according to permit a wide variety of computing implementation tradeoffs. By very smooth, it means that the smoothness bound is multiple fixed sequential functions. A valuable comparison between the software performance of CHA-128 design and some popular hash function algorithms designs is performed.

Keywords:

Hash code; Message integrity; SHA-family; Message Digest.

1 Faculty of Engineering, Ain Shams University, Cairo, Egypt.
2 Technical Research Center, Cairo, Egypt.

1. Introduction:

Cryptographic hash functions play an essential role within a wide range of information security mechanisms and protocols. Their most crucial role is to provide data integrity and efficiency for digital signatures and general authentication purposes and key derivation. When two parties are communicating over an insecure channel, they need a method by which the original information sent by the sender can be accepted by the receiver without an uncertainty on possible alteration or changing. The integrity of the message can be verified by the hash functions which involves all the bits of the whole message. It accepts the variable size message as input and produces a fixed size output as the hash code. A change in any bit or bits in the message results in change in the hash code thus providing an indication of message tampering.

For a cryptographic hash function H, it must be able to overcome all known types of cryptanalytic attack. At least, it must avoid the following properties, Preimage: given $y = H(x)$, find x' such that $H(x') = y$, 2nd preimage : given an x and $y = H(x)$ find $x' \neq x$ such that $H(x') = y$, and Collision**:** find x and x' such that $x' \neq x$ and $H(x) = H(x')$.

For an ideal hash function with an m-bit output, finding a preimage or a second preimage requires about 2^m operations and the fastest way to find a birthday or square root attack is approximately $2^{m/2}$ operations [1]. Most popular hash functions are designed using Merkle-Damgaard model [2]. This model simplifies the management of large inputs and produces a fixed length output. The message is viewed as a collection of m-bit blocks. Many hash functions such as Message Digest, MD, family [3] and Secure Hash Algorithm, SHA, family [4] are based on this idea. New proposed hash function, CHA, certainly differs from all hash function algorithms of the MD family (MD5, SHA-1, SHA-2) as it uses a private function construction. Also, CHA-128 is differs from new standard hash algorithm, KECCAK [5], as SHA-3, which uses a sponge function construction. This fact can be considered neither as an advantage nor a disadvantage. Private functions are a new construction that will now, attract more attention from cryptographers. Certainly this is a disadvantage as this construction is not very well analyzed yet. However, it is an advantage that CHA has not the same structure of either MD family or SHA-3, which means that there is no need to have two algorithms with similar construction.

Given this context, it is clear that the community should welcome new approaches to the design of secure hash functions. This paper is structured as follows. Section II describes motivations and design requirements for a new proposed hash function. Next, in Section III, we present our proposed hash algorithm structure. We discussed some of the analysis, performance, and security features of our scheme in Section IV, and finally we gave our conclusion in Section V.

2. Motivation and Design Requirements for CHA

Cryptographic hash functions are a useful building block for several cryptographic applications. The most important are certainly the protection of information authentication and digital signatures. Hash functions are functions that map an input of arbitrary length to a string of fixed length, the hash code. If these mappings satisfy some additional cryptographic conditions, they can be used to protect the integrity of information. Other cryptographic applications where hash functions are useful are the optimization of digital signature schemes, the protection of pass phrases and the commitment to a string without revealing it. In short, they need to be both efficient and secure and in most commercial applications, they need to run quickly in software on all the common hardware platforms.

In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use. NIST required many applications in federal agencies to move to SHA-2 after 2010 because of the weakness. Although no successful attacks have yet been reported on SHA-2, they are algorithmically similar to SHA-1. In 2012, following a long-running competition, NIST selected an additional algorithm, Keccak, for standardization as SHA-3. But the work on SHA-3 is not completely done yet. For KECCAK to achieve security assurance, it is vital that third-party cryptanalysis reviews its security from multiple points of views.

However, introducing a new structure in cryptographic algorithms also involves the risk of unexpected weaknesses.

Clearly, the current situation brings us to the conclusion that it would be prudent to design a new hash algorithm avoiding leakage, continuous cryptanalyze efforts, and hidden risks of current implementations, from these considerations; we believe that our new proposed design of hash function should:

- Satisfy large security margin. At the very least it must be one-way, collision-free and multiplication-free.
- Elegant design: can be used as a replacement for popular hash functions.
- Satisfy high efficiency in both hardware and software implementations.
- Give good general performance and flexibility.

One common use of hash functions is to “destroy” any structure that may exist in the input, while preserving most of its entropy. Validity of using hash functions for entropy extraction is not based on their cryptographic properties but rather on our belief that a good hash function destroys most of the dependencies that may exist in the bits of its input. Finally, we have to think more before we make specific recommendations for specific applications.

3. Description of CHA-128

CHA-128 is a keyed hash function designed for hashing any input message, using secret key that is equal to 128 bits in length, as initial value of stream cipher. CHA-128 algorithm outputs any input message stream into 128-bit block. Also, it is a software-oriented cipher based on 32-bit operations (such as 32-bit XOR and addition modulo 2^{32}), and references to software design small fixed blocks. Accordingly, CHA can be used in many computing applications, and many cryptographic requirements. Software design of CHA-128 is implemented using a uniform C++ programming language, where they have conducted it on a 2.4 GHz Pentium 4 processor and 512 Mbyte RAM Personal Computer.

3.1. Input Block Length and Message Length Management.

An input message is processed as 128-bit blocks. Padding is used to make the length of the original message equal to a value exactly multiple of 128 bits, if its length is more or less. The padding consists of a single 1-bit, followed by as many 0-bits, as required. After padding is added, the original length of the message is calculated and added at the end of the message as a 128-bit value. In the case of a really long message, the length of the message is calculated as the original message length modulo 2^{128}.

3.2 Structure of our Proposed.

The CHA-128 bit algorithm uses four 32-bit working variables, T_0 to T_3, and four 32-bit result variables states, S_0 to S_3; these constitute all the states of the algorithm from round to round. This algorithm also uses four 32-bit intermediate values, T_0, T_1, T_2 and T_3, and number of iterated rounds. An iterative of hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. CHA operates on 128-bit blocks. The size of the output of CHA is the same as that of the underlying hash function, 128 bits in our case, although it can be truncated if desired.

3.2.1 Open Input File

The first operation generally performed on any object, In order to open a file with a stream object, of one of these classes is to associate it to a binary file. This procedure is known as to *open a file*. An open file is represented within a program by a stream object and any input or output operation performed on this stream object will be applied to the physical file associated to it.

3.2.2 Read Key

In cryptography, a **keyed-hash message authentication code** (**HMAC**) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret cryptographic key.

As with any MAC, it may be used to simultaneously verify both the *data integrity* and the *authentication* of a message. Any cryptographic hash function may be used in the calculation of an HMAC. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key. Recall our goal is to build secure message authentication functions from cryptographic hash functions. A first clear obstacle is that while secret keys are an essential ingredient in a message authentication function, most cryptographic hash functions, do not use keys at all. Therefore, we first need to define a way to use cryptographic hash functions in conjunction with a key.

The most common approach for a keyed hash function is to input the key as part of the data hashed by the function, e.g., hashing data, x, using key, k, is performed by applying the hash function, F, to the concatenation of k and x. This approach is using the key value as the IV of the encryption algorithm function. In our function procedure we use a random and secret key value known only to the two communicated parties, and then applying the necessary confusion and diffusion of our algorithm as shown below, which add some significant analytical advantages. Notice that keyed hash functions are the same as the original iterated hash function but with input key value. Reading the key file is a process to create an array to hold the keyed hash value read from the predefined file, and prepare it for four 32-bit result variables to be compatible with CHA blocks bit size.

3.2.3 Initial States

Before hash computation begins, the working variables, State [0] to State [3], are initialized with a predefined random four 32-bit words in order as shown:

State [0] = key [2] , State [1] = key [0], State [2] = key [3], State [3] = key [1];

These values are identical to the initial values for iterated hash function. These words were obtained from random and secret values known only to the parties.

3.2.4 Read Data and Update Hash Computation

Message preparation should take place before hash computation begins. This preparation consists of five steps: reading the input message, calculating the message lengths, padding the message M if necessary, dividing the padded message P to the requisite length (multiple of 128-bit), and finally calculating the number of, 128-bit words, message blocks. If the length of the message M is an exact multiple of 128 bits, no padding is added and the padded message P is identical to the original message M. Otherwise, the complement of the last bit of the message shall be appended, with almost zeroes, repeatedly until the resulting length reaches the next exact multiple of 128 bits, of the block size. After a message has been prepared to the appropriate length, as described above, it must be parsed into a number of 128-bit words before the hash computation can begin.

After the message has been prepared and the variables initialized, perform the following computations for each 32-bit word in the prepared message:

- Apply the Compression Function, with Preliminary Intermediate Values

The intermediate values, T [0] to T [3], are initialized by combining the first four 32-bit blocks of data with two different 32-bit blocks of the initialized states and other two different 32-bit blocks of the random input keys to produce four intermediate 32-bit words in order as shown in equation (1),

T [0] = data [0] + key [0], T [1] = data [1] + state [0], T [2] = data [2] + state [1], T [3] = data [3] + key [3] (1)

- Rotation of the Intermediate value:

Generally, we do *not* want to use a cryptographic hash for a hash table, an algorithm that's *very* fast by cryptographic standards is still excruciatingly slow by hash table standards. Also, we want to ensure that every bit of the input will affect the result; one easy way to do that is to use the non-linear rotation function. A rotation function is used to rotate 32-bit input to produce 32-bit value from the previous step by certain rotation factor, R. **Note that**, the rotation factor, R, can be one of the values (0, 1, 5, 7, 19, 22), from Shannon.

- Calculate Multiplexing function value of the rotated Value:

One of the master functions of CHA algorithm design is the multiplexer function. Choice of the output multiplexed bits is made based on non-linear multiplexer function operation depending on the 32-bit selecting point, C. The output 32-bit is selected depending on the two inputs 32 bit groups, A and B, to the multiplexer function, which are changed each round.

- Compute Multiplication function for the multiplexed 32-bit to 64-bit values:

The multiplication function will extend a pair of 32-bit input values *A* and *B* in binary notation into 64-bit values, as a product, which concatenate two output halves of 32-bit values, **Low** and **High**, and then updates the new state values.

- Update Hash Computation and Finalize Hash Computations:

Finally, we reuse multiplexing and multiplying functions to update inter-state values using various complex combinations.

These five steps constitute one round form specific number of iterations of the algorithm. After repeating these steps for each word in the prepared message, the resulting 128-bit message digest of the message M is State0 || State1 || State2 || State3, Human-readable output is generated most significant byte first.

Discussion: In a pure block cipher, the hash result variables, State0 to State3, would be updated only once per block. However, doing that has been shown to be a serious flaw in the CHA-128 design hash functions. Consequently, these variables are updated every round iteration in the algorithms presented here.

4. Security Analysis and Performance Evaluation for CHA:

It is designed to be elegant design, large security margin, good general performance, excellent efficiency in hardware implementations, and for its flexibility.

4.1 Security Analysis

In this section, we present the goals we have set for the security of CHA-128. These claims result from the considerable safety margin taken with respect to most known attacks. We do however realize that it is impossible to make non-speculative statements on things unknown.

4.1.1 Designers' statement on the absence of hidden weaknesses:

In spite of any analysis, doubts might remain regarding the presence of trapdoors deliberately introduced in the standard or even published algorithm. Therefore we do hereby declare that there are no hidden weaknesses inserted by us in the CHA primitive.

4.1.2 Designers' statement of Not Known Idea:

The main difficulty in cryptanalyzing CHA-128 comes from the fact that our design choices are fairly different from those in the MD-Family and SHA-Family hash functions. CHA-128 therefore provides diversity with respect to existing standards. The attacker who tries to break CHA-128 should analyze the primitive algorithm function structure where the difference ideas, which would make the attacks more difficult.

4.1.3 Design Confusion and Diffusion:

While combining the outputs from the three main core functions, orthogonal logic operations are used to create confusion and diffusion [16] which adds to the security. Partial collisions do not produce full collisions for same length messages. This property ensures that attacks based on establishing and maintaining a collision in the Bijection Selection data must fail.

4.1.4 Design Strict Avalanche Criteria:

There is a strong avalanche effect [17], in that each message, when an input is changed slightly, the output changes significantly, during block iteration process.

4.1.5 Design Output Expected Strength, Deferential Cryptanalysis:

Assume we take as hash result the value of any n-bit substring of the full CHA-128 output. The single step operation ensures that changing a small number of bits in the message affects many bits during the various passes. Together with the strong avalanche, it helps CHA-128 to resist differential attacks [18].

Moreover, it is infeasible to detect systematic correlations between any linear combination of input bits and any linear combination of bits of the hash result. It is also infeasible to predict what bits of the hash result will change value when certain input bits are flipped, i.e. CHA-128 is resistant against differential attacks.

4.1.6 Pre-Image Resistance:

The construction of the initial state values prevents meet-in-the-middle birthday attacks that find preimage of the hash function [19].

4.1.7 Design Output Birthday Attack:

The birthday attack is an attack that can discover collisions in hashing algorithms. It is based on the Birthday Paradox, which states that if there are 23 people in a room, assuming that birthdays are distributed evenly over 365 days, with no leap years, the odds are slightly greater than 50% that two will share the same birthday.

By using three non linear functions during each 128-bit block steps, it has been made difficult to construct a differential characteristic with high probability. Also, the keying of the CHA-128 design is irregular (rather than constant as for standard CBC) so the application of some birthday paradox style attacks is prevented.

4.2 Performance Evaluation:

4.2.1 Efficiency estimates

Using the reference C implementation on a 1 GHz Pentium III platform, we observe that many factors explain the observed performance. First, a 32-bit or 64-bit processor was used to test a native 32-bit implementation; better results are expected by merely running the speed measurement on Pentium processor. Second, it seems that the pipe parallelism capabilities of the Pentium may be partially used; this may reflect an optimization of the implementation of 32-bit arithmetic support by the C compiler, and might be better by an assembler implementation. Third, there is no tables employed in the reference implementation, are quite small, and the built-in processor cache must be enough to hold the application, the data being hashed, and the hashing code at once, thus increasing processing speed.

4.2.2 Flexibility

CHA-128 inherits the flexibility of the special purpose functions constructions. It is much more extendable than most modern hashing functions. Even though is not specifically oriented toward any platform, it is rather efficient on many of them, its structure favoring extensively single round execution of the component mappings. At the same time, it does not require excessive storage space (either for code or for tables), and can therefore be efficiently implemented in quite constrained environments like smart cards. CHA can achieve higher performance if it is applied on modern computers with larger processors cache memory. It does not use expensive or unusual instructions that must be built in the processor. The mathematical simplicity of the primitive resulting from the design algorithm tends to make processing speed faster. The instance proposed for CHA-128 makes use of a single round for all security strengths. This cut down implementation costs compared to hash function families making use of two (or more) primitives, like the SHA-2 family.

4.2.3 Performance Evaluation comparisons

In this section we compare the performance of the public SHA and proposed CHA algorithms on the basis of different parameters like message digest length, and block size, as shown in Table-1. Implementations were written in high level language software programming visual studio.net to test these hash Algorithms. For experiment, Intel Core i5 2.40 Ghz, 4 GB of RAM and Window-7 Home Basic SP1, have used in which performance data is collected.

Table (1): *Timing Comparison between Proposed CHA, three SHA (SHA-160, SHA-176 and SHA-192) algorithm for 15 KB file*

Algorithms	Hash Timing (in second)	Message Digest
CHA-128	0.125	4B 2A 1D 3C 08 FF D3 4E BB 32 FE 01 3C B6 36 20
SHA-160	0.791	D4 11 0A D3 B2 F6 2A 06 C8 0E 3A F1 1F A8 1B 8D F5 48 66 E6
SHA-176	0.940	FB E4 B7 93 BA 05 02 FB 2E C2 10 D2 F6 4A 05 8B 0D 83 10 B7 2C C9
SHA-192	2.120	B3 23 75 FD 55 23 28 59 EB CD 79 0C CF 66 9A 20 6D 8B 39 30 16 49 59 3E

Table-1 is showing execution time comparison between CHA, and three SHA hash functions designed by NSA (National Security Agency) which are the set of cryptographic hash function. The first one is SHA-1 produces message digest that is of 160 bits long. Later in SHA-1 has been identified security flaws, namely that a possible mathematical weakness might exist. This point out that stronger hash function would be desirable. The second one is SHA-176 [20] that has more strength than the existing SHA-160. The last hash function is SHA-192 [21] has been designed to satisfy the different level of enhanced security and to resist the advanced SHA attacks.

4.3 Security Performance Trade-Off:

The problem of modeling and analyzing software architectures for cryptographic systems is usually addressed through the introduction of sophisticated modeling notations and powerful tools to solve such models and provide feedback to software engineers. The increasing complexity of software systems creates large effort to jointly analyze their non-functional attributes in order to identify potential tradeoffs among them (e.g. increased availability can lead to performance degradation) [22].
From the previous performance and security results, our proposed design can in fact provide the best tradeoff between security and performance properties.

5. Conclusions:

We observe that hash functions are in widespread use in information processing applications more so than any other cryptographic topics, additionally there are a few ideas of its internal structural diversity. This leads to the serious possibility of a single point of failure in cryptographic security. Thus there is a requirement for new secure and efficient hash design schemes, which motivated our proposal of internal F-functions as a new class of hash algorithms. In this paper we have put forward a new hash function called CHA-128 has been designed with improved security and reasonable speed. CHA-128 is much more scalable than most modern hashing functions. Even though is not specifically oriented toward any platform, it is rather efficient on many of them. At the same time, it does not require excessive storage space (either for code or for tables), and can therefore be efficiently implemented in quite constrained environments like smart cards, although it can benefit from larger cache memory available on modern processors to achieve higher performance. It does not use expensive or unusual instructions that must be built in the processor. Also, it has an acceptable hash length; which provides increased protection against birthday attacks. Finally, as usual when proposing a new cryptographic algorithm design, we urge all cryptographers, graduates, and interested people in this field to study the strength of CHA-128, we will appreciate attacks, analysis and any other comments.

References:

[1] Mihir Bellare, Tadayoshi Kohno: *Hash Function Balance and Its Impact on Birthday Attacks*. EUROCRYPT 2004: pp401–418, 2004.

[2] J.S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. *Merkle–Damgård Revisited: How to Construct a Hash Function.* Advances in Cryptology – CRYPTO '05 Proceedings, Lecture Notes in Computer Science, Vol. 3621, Springer-Verlag, pp. 21–39, 2005.

[3] M.J.B. Robshaw, *MD2, MD4, MD5, SHA and Other Hash Functions*, Technical Report TR-101, version 4.0, RSA Laboratories, 1995.

[4] Helena Handschuh, "SHA-0, SHA-1, SHA-2 (Secure Hash Algorithm)", Encyclopedia of Cryptography and Security, pp 1190-1193, 2011.

[5] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche - *The Keccak reference Submission to NIST* (Round 3), 2011. http://keccak.noekeon.org/Keccak-submission-3.pdf. Accessed 10/08/2013.

[6] Bart Preneel, Rene Govaerts, and Joos Vandewalle, "*Hash functions based on block ciphers: a synthetic approach*", Appeared in Advances in Cryptology { CRYPTO 1993, Lecture Notes in Computer Science 773, D. R. Stinson (Ed.), Springer-Verlag, pp. 368-378, 1993}, Springer-Verlag, 1993.

[7] Rivest, R.L.: *The MD4 Message Digest Algorithm.* In Menezes, A., Vanstone, S.A., eds.: CRYPTO. Volume 537 of Lecture Notes in Computer Science., Springer PP 303–311, 1990.

[8] R.L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* M.I.T. Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[9] H. Dobbertin, A. Bosselaers, Preneel, "*RIPEMD-160, a strengthened version of RIPEMD. Fast Software Encryption*", LNCS 1039, D. Gollmann, Ed., Springer-Verlag, pp. 71-82, 1996.

[10] Christophe De Cannière, Christian Rechberger, " *Finding SHA-1 Characteristics: General Results and Applications* ", Advances in Cryptology – ASIACRYPT 2006. Lecture Notes in Computer Science Volume 4284, 2006, pp 1-20.

[11] Florian Mendel, Tomislav Nad, Martin Schläffer, "*Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions*" , Advances in Cryptology – ASIACRYPT 2011. Lecture Notes in Computer Science Volume 7073, pp 288-307, 2011.

[12] Ralph C. Merkle, "*A fast software one-way hash function*", Journal of Cryptology, Volume 3, Issue 1, pp 43-58, 1990.

[13] Gaoli Wang, "*Collision Attack for the Hash Function Extended MD4*", Information and Communications Security. Lecture Notes in Computer Science Volume 7043, pp 228-241, 2011.

[14] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. "*Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*". Cryptology ePrint Archive, Report 2004/199, 2004. http://eprint.iacr.org/2004/199. Accessed 10/08/2013.

[15] B. Preneel, "*Collision resistance*", Encyclopedia of Cryptography and Security, pp 221-222, 2011.

[16] Baris Coskun, Nasir Memon, "*Confusion/Diffusion Capabilities of Some Robust Hash Functions*", In Proceedings of the 40th Annual Conference on Information Sciences and Systems {(CISS'06)} (March 2006).

[17] Rajeev Sobti, G.Geetha , "*Cryptographic Hash Functions: A Review*", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 2, March 2012.

[18] Vincent Rijmen, Bart Preneel, "*Improved Characteristics for Differential Cryptanalysis of Hash Functions Based on Block Ciphers*", Fast Software Encryption, LNCS 1008, B. Preneel Ed., Springer-Verlag, PP. 242-248, 1995.

[19] Rogaway, P.; Shrimpton, T. "*Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance*". *Fast Software Encryption, Volume 11, (2004)* (Springer-Verlag), pp.371-388. Retrieved 17 November 2012.

[20] Piyush Garg1, Namita Tiwari," *Evolution of Sha-176 Algorithm*", IOSR Journal of Computer Engineering (IOSRJCE), ISSN: 2278-0661 Volume 2, Issue 2, PP 18-22, (July-Aug. 2012).

[21] Thulasimani Lakshmanan1 and Madheswaran Muthusamy, "*A Novel Secure Hash Algorithm for Public Key Digital Signature Schemes* ", The International Arab Journal of Information Technology, Vol. 9, No. 3, May 2012.

[22] Jindal, P., Singh, B.: *Study And Performance Evaluation Of Security-Throughput Tradeoff With Link Adaptive Encryption Scheme*. ;CoRR abs / 1211.5080 (2012).