Fast software-oriented hash function based on data-dependent lookup operations

Moldovyan N.A., Summerville D.H.

Abstract

The paper considers a method of the construction of the iterated hash function on the bases of the data-dependent lookup operations used previously in the design of the fast software suitable ciphers. To transform encryption function into a block one-way function we use the data-dependent initial condition at each transformation cycle of the round function except the first cycle. The variable initial conditions has been also used to strengthen chaining while constructing the iterated hash function. While fixing initial condition the round function can be transformed into a block cipher suitable to perform fast disk encryption. The size of the input data block of the round function and of the block cipher is parameterized defining their suitability for different practical applications.

Key words: data-dependent operations, hash function, fast cipher, software implementation

1 Introduction

Fast software suitable cryptographic methods are widely used as a basic protection mechanism in modern computer security systems. A special unkeyed checksum-like function is called one-way hash function (or simply hash function). Hash functions suites well for data integrity control as well as for producing message digests in digital signature systems. A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values: $\mathbf{Hash}(M, H_0) = H$, where $M = (M_1, M_2, \ldots, M_n)$

is a hashed message, H_0 is a specified initial value, H is the hash value from M. Since H_0 is a specified, one can write simply $\mathbf{Hash}(M) = H$. Usually hash function has iterated structure. An iterated hash function **Hash** is determined by an easily computable round function **h** from two bit strings M_i and H_{i-1} of respective lengths m and h: $H_i = \mathbf{h}(M_i, H_{i-1})$, where $i = 1, 2, \ldots, n$ and $H = H_n$. Hash function should be strongly collision-free. A hash function is strongly collision-free if it is computationally infeasible to find messages M and M' that $M \neq M'$ and $H = H'$. The strongly collision-free property implies the one-way property [1].

A number of hash functions have been proposed earlier. The well known hash functions are Snefru, N-Hash, MD5, MD4, MD2, and SHA [2, 3].The algorithms MD5 and SHA are widely used. An interesting approach to the iterated hash function design is connected with the use of the block ciphers to construct round functions. Because of the birthday attack the minimum size of the hash value should be 128 bit [1, 2]. A class of hash functions is based on the use of the 128-bit block ciphers [2]. Using more complex constructions allows one to develop 128-bit iterated hash functions based on 64-bit block ciphers [3]. Use of the 128-bit block ciphers is very attractive, however it is connected with the following problem. The majority of the known block ciphers use the key scheduling. Performing the key scheduling in block ciphers is not critical, but in the round hash functions the data blocks or intermediate hash values are used as key. This defines necessity to perform the key scheduling procedures in each round of data hashing that leads to reduction of the performance.

The purpose of this work is to construct a secure and fast software-oriented hash function based on some fast encryption mechanism avoiding key scheduling procedure. To solve this task we use data-dependent table lookup operations as a basic cryptographic primitive. Since in different practical applications it is desirable to use hash values having size from 128 to 256 bit, we shall construct a hash function that allows specifying the hash value size. In the design of such hash function we shall use the round function with flexible input.

Data-dependent subkey selection has been earlier shown to be effective for creation of fast software encryption algorithms [4, 5]. Such mechanism resembles table lookup operations (or simply table lookups), however in some variants it has significant peculiarities. Indeed, the subkey selection depends not on the current data subblock only but on many previous ones also [6]. This variant of the data-dependent subkey selection represents data-dependent lookups operations. Many investigations [4-7,13] show the data-dependent operations to be very effective cryptographic primitive. In present paper we show that data-dependent lookup operations are very efficient for the design of fast software suitable hash functions.

2 Data-dependent table lookups hash function

2.1 Notation

Throughout this paper we will use the following operations, symbols and notation:

- the term 'word' is used to denote a 32-bit number, and we will use letters: T, C, X etc;
- we shall denote an element of the sequence $\bar{\mathbf{L}}$ by L_i, where $i = 0, 1, 2, \ldots$.
- "$|$" denotes the concatenation operation;
- the sequence of bytes $\bar{\mathbf{L}} = \{l_0, l_1, \ldots, l_n\}$ we shall also interpret as a sequence of the 32-bit words $\bar{\mathbf{L}} = \{L_0, L_1, \ldots, L_s\}$, where $L_j = \{l_{4j}, l_{4j+1}, l_{4j+2}, l_{4j+3}\}$ and $j = 0, 1, \ldots, s$; while interpreting several sequential bytes by a binary number the right byte corresponds to the most significant digits, for example the sequence $\bar{\mathbf{L}}$ is interpreted as number $l_n|...|l_1|l_0$ and four sequential bytes $\{a_1, a_2, a_3, a_4\}$ we interpret as $A = a_4|a_3|a_2|a_1$;
- "$+_f$" ("$-_f$") denotes modulo 2^f addition (subtraction) of words (for example, the equation $j := Z \bmod 2^{11}$ is equivalent to the equation $j := Z +_{11} 0$);

- "$\oplus$" ("$\otimes$") denotes bit-wise exclusive-OR (AND) operation;
- "$\ggg$" denotes right circular rotation of words: the cyclic rotation of word X right by Y bits is denoted "$X^{\ggg Y}$" (note that in such data-dependent rotations only $\log_2 32 = 5$ lower-order bits of Y are used to determine the rotation amount).
- ":=" denotes assign operation;
- "$\leftrightarrow$" denotes exchange operation; for example $W \leftrightarrow V$ produces the same result as three operations $T := W, W := V, V := T$;
- We use the following hexadecimal constants: F = FFFF07FF, a = 0D, P = B25D28A7; 1A62D775, R = 98915E7E; C8265EDF; CDA31E88; F24809DD; B064BDC7; 285DD50D; 7289F0AC; 6F49DD2D.

2.2 Design criteria

We use the following design criteria:

1. Round hash function (RHF) should be suitable for software implementation.
2. The RHF should have flexible input, i.e. it should be possible to assign the input data block size m to be equal Sm_0, where $m_0 = 32$ bits and S natural number $S \geq 4$. This should allow use the hash function in different applications.
3. RHF should be based on data-dependent table lookup operations, which are efficient cryptographic primitive.
4. should have large number of internal states, i.e. it should use sufficient number of internal variables, the last being 32-bit ones.
5. should be based on fast CPU operations such as "$\otimes$", "$\oplus$", "$+_f$", "$-_f$", "$\ggg$" (including data-dependent rotations).

6. Majority of transformations used in RHF should be bijective. (If we represent RHF as a superposition of operations, then only few of them should define non-bijective mapping). Reducing the number of non-bijective operations should help to construct a "collision resistant" hash function.

Elaborating hash function for practical application in computer security software our strategy was oriented to extensive use of the data-dependent lookups. Such hashing mechanism suites well to software implementation. To reduce the number of lookups we shall use a known table of "pseudorandom" numbers, the table size being 2051 bytes. Such size allows one to define selection of the values from the table in the direct dependence on 11 bits of the current data subblock. We use also a simple mechanism defining indirect dependence of lookups on all bits of data block. The table we shall construct using a special preprocessing algorithm.

2.3 Generation of the table and basic procedures

Precomputations are represented by the following algorithm.

Procedure *Table_Z*:

1. Set counter $i = 0$.
2. Calculate 32-byte value $Z'_i = (a^{23+i} \bmod P)^{17} \bmod R$.
3. Increment $i := i + 1$. If $i \neq 64$, then jump to step 2.
4. Form 2051-byte number $T = z'_2|z'_1|z'_0|Z'_{63}|\dots|Z'_0$, where $z'_2|z'_1|z'_0 = Z'_0 +_{24} 0$.
5. Represent T as sequence of bytes $\bar{\mathbf{Z}} = \{z_0, z_1, \dots, z_{2050}\}$.

OUTPUT: table $\bar{\mathbf{Z}}$.

Table $\bar{\mathbf{Z}}$ is used for selection of the words $Q_n = z_{n+3}|z_{n+2}|z_{n+1}|z_n$ the given value n. The round hash function uses the following procedures: ***Initialize*** and ***ChangeNVYU***.

Initialize:

1. Set counter $i := 0$ and number $n := Q_2 +_{11} 0$.
2. Set internal variables: $R := Q_4$; $V := Q_{78}$; $Y := Q_{16}$; $U := Q_{32}$; $N := Q_{64}$.
3. STOP.

ChangeNVYU:

1. Transform N and V: $N := N \oplus R$; $V := V +_{32} N$.
2. Transform $N := N \otimes F$.
3. Set $n := N +_{11} 0$.
4. Transform V, N and Y: $V := (V +_{32} Q_n)^{\ggg 11}$; $N := N \oplus V$; $Y := Y +_{32} N$.
5. Transform $N := N \otimes F$.
6. Set $n := N +_{11} 0$.
7. Transform Y and N: $Y := (Y +_{32} Q_n)^{\ggg 11}$; $N := (N +_{32} Y) \otimes F$.
8. Set $n := N +_{11} 0$.
9. Transform $U := [(U \oplus Q_n) +_{32} R]^{\ggg V}$.
10. STOP.

2.4 One-way round function

The one-way round function based on data-dependent lookups is described by the following algorithm.

Algorithm 1.

INPUT: data block M_i represented as sequence of z 32-bit words $\{W_0, W_1, \ldots, W_{z-1}\}$.

1. Set size of input data block $S := z$.
2. Set external counter $j := 6$.

3. Execute procedure ***Initialize***.

4. Set internal counter $v := 0$.

5. Execute procedure ***ChangeNVYU***.

6. Transform the current word: $W_v := (W_v -_{32} V) \oplus U$.

7. Transform the variable R: $R := R +_{32} W_v$.

8. Finish transformation of the word W_v: $W_v := W_v^{\lll V} -_{32} Y$.

9. Increment $v := v + 1$. If $v \neq S$, then jump to step 5.

10. Decrement $j := j - 1$. If $j \neq 0$, then jump to step 4, otherwise **STOP**.

OUTPUT: the converted data block $C_i = W_{z-1}|\ldots|W_1|W_0$.

3 Characterization of the round hash function

The round transformation defined by algorithm 1 is analogous to the encryption procedure of [6] in the case of the known encryption key. One-way character of the round hash function is defined by the fact that initialization of the variables R, V, Y, U, and N is performed before the first cycle only. The set of the initial values of these variables represent the initial condition that is set by the procedure ***Initialize***. Different combinations of the values of variables R, V, Y, U, N define different internal states of the hashing mechanism, which can be described by the 160-bit value $G = R|V|Y|U|N$. Depending on the input data block M_i the value G is changed consecutively $6z$ times, i.e. at each step of the transformation of some word W_v.

The avalanche effect corresponding to the change of the value G spreads well when moving from the word W_i to the word W_{i+1}. It is defined by the fact that each bit of the word W_i defines the selection of at least one value Q from the table $\bar{\mathbf{Z}}$. The round transformation is composed in such a way that any alteration in encryption process are accumulated in variables R, V, Y, U, N. Due to this after the first transformation cycle the table lookup operations depend on all data bits. This causes strong avalanche effect.

Let G_i be the final internal state of the hashing mechanism after M_i is transformed into C_i. One can easy calculate initial value M_i corresponding to C_i, provided the value G_i is known. If the value C_i is chosen at random, i.e. the value M_i is unknown, then it is computationally infeasible to determine the final internal state G_i due to very large number of the internal states (2^{160}). This defines security of the considered round hash function.

Security analysis of the block ciphers based on data-dependent lookup operations [13] shows that such operations represent efficient cryptographic primitive, however estimating security of the hash functions one should take into account the birthday attack. The last defines the minimum size of the data blocks (≥ 128 bit) in the described hash function and the minimum size (≥ 128 bit) of some value G which corresponds to the internal state of the round hash function. In algorithm 1 the size of the value $G = R|V|Y|U|N$ is 160 bit and the values $z \geq 4$ are acceptable. If necessary, taking algorithm 1 as a prototype one can easy compose a new round hash function with more variables. For example, the use of six or eight 32-bit variable defines 192-bit or 256-bit value G, respectively.

There are possible different variants to define the round function with the use of algorithm 1. The simplest one is shown in Fig. 1a, where G_0 corresponds to the initial condition and box $\mathbf{E}$ denotes the transformation function defined by algorithm 1. This variant can be described as follows:

$$H_i = \mathbf{h}(M_i, H_{i-1}) = \mathbf{E}_{G_0}(M_i) \oplus H_{i-1} = C_i \oplus H_{i-1},$$

where $i = 1, 2, \ldots, n$ and the value H_0 is specified. The hash value from message M is $H = H_n$. Figure 1b describes the hash function corresponding to the formula:

$$H_i = \mathbf{h}(M_i, H_{i-1}) = \mathbf{E}_{G_0}(M_i \oplus H_{i-1}) \oplus M_i.$$

Figure 1c shows the hash function described by the formula:

$$H_i = \mathbf{h}(M_i, H_{i-1}) = \mathbf{E}_{G_0}(M_i \oplus H_{i-1}) \oplus H_{i-1}.$$

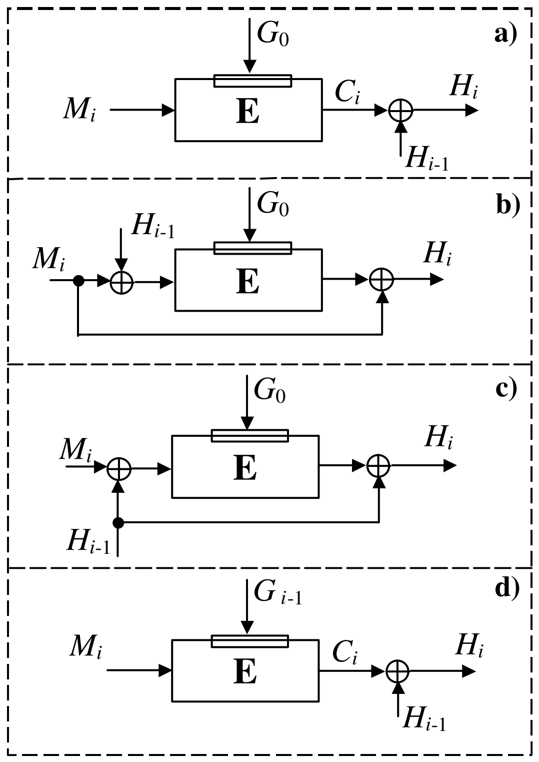

Figure 1. Round functions with different variants of the chaining mechanism

The mechanism of the data transformation used in algorithm 1 is very close to that of the data encryption based on data-dependent lookup operations, therefore it is easy to modify algorithm 1 into some block encryption function with flexible input. For this purpose one should use key-dependent table $\bar{\mathbf{Z}}$ and replace in step 10 the phrase "then jump to step 4" for "then do procedure ***Reorder*** and jump step 3", where procedure ***Reorder*** is the following one.

Reorder

1. Set counter $v := 0$.
2. $W_v \leftrightarrow W_{z-v-1}$
3. If z is even, then jump to step 5.
4. If $v \neq (z-1)/2$, then increment $v := v+1$ and jump to step 2, otherwise **STOP**.
5. If $v \neq z/2$, then increment $v := v+1$ and jump to step 2, otherwise **STOP**.

The extended encryption key $\bar{\mathbf{Q}}$ (that is key-dependent table used instead of $\bar{\mathbf{Z}}$) can be generated in the following way. The secret key is repeated a necessary number of times until the 2052 bytes sequence is received. Let $\bar{\mathbf{Q}}'$ denote such sequence. Then the 2051-byte auxiliary table $\bar{\mathbf{H}}$ is generated using sequence $\bar{\mathbf{Q}}'$ and the table $\bar{\mathbf{Z}}$:

$$\bar{\mathbf{H}} = \left(\bar{\mathbf{Q}}' \bmod 2^{16408}\right) \oplus \bar{\mathbf{Z}}.$$

Then, setting value $S = 513$ and using algorithm 1 with table $\bar{\mathbf{H}}$ instead of $\bar{\mathbf{Z}}$ the sequence $\bar{\mathbf{Q}}'$ is transformed into extended encryption key $\bar{\mathbf{Q}}$. This transformation is described below as ***Form_Key***. The encryption key $\bar{\mathbf{Q}}$ represents some ordered sequence of bytes $q[i]$: $\bar{\mathbf{Q}} = \{q_i\}$, where $i = 0, 1, \ldots, 2051$. While performing encryption the following subkeys $Q_j = q_{j+3}|q_{j+2}|q_{j+1}|q_j$, where $j = 0, 1, \ldots, 2047$, are used.

Procedure *Form_Key*

1. Initialize parameter $z = 513$.
2. Using algorithm 1 with table $\bar{\mathbf{H}}$ instead of $\bar{\mathbf{Z}}$, convert $\bar{\mathbf{Q}}'$.
3. The output of step 2 is taken as $\bar{\mathbf{Q}}$.

Algorithm 2.
INPUT: data block M_i represented as sequence of z 32-bit words $\{W_0, W_1, \ldots, W_{z-1}\}$.

1. Set size of input data block $S = z$.
2. Set external counter $j := 6$.
3. Using key-dependent table $\bar{\mathbf{Q}}$, execute procedure ***Initialize***.
4. Set internal counter $v := 0$.
5. Using key-dependent table $\bar{\mathbf{Q}}$, execute procedure ***ChangeN-VYU***.
6. Transform the current word: $W_v := (W_v -_{32} V) \oplus U$.
7. Transform the variable R: $R := R +_{32} W_v$.
8. Finish transformation of the word W_v: $W_v := W_v^{\lll V} -_{32} Y$.
9. Increment $v := v + 1$. If $v \neq S$, then jump to step 5.
10. Decrement $j := j - 1$. If $j \neq 0$, then execute procedure ***Reorder*** and jump to step 3, otherwise **STOP**.

OUTPUT: the round value of hash function $H_i = W_{z-1}|\dots|W_1|W_0$.

One can say that the round hash function is constructed on the basis of this block encryption algorithm. One-way character of the round hash function is caused by the fact that the procedure ***Initialize*** is executed only once per one data block. Since final values of variables R, V, Y, U, N, and n are not known it is computationally difficult to reconstruct input corresponding to the given output. Such mechanism of the transformation of the block cipher into a round hash function is different from that used earlier [1-3]. The investigation of statistic properties of algorithms 1 and 2 has been carried out with tests proposed in [8]: (1) the average number of output bits changed when changing one input bit; (2) the degree of completeness; (3) the degree of the avalanche effect; (4) the degree of strict avalanche criterion. Our statistic experiments have shown that 2 rounds are quite sufficient to get uniform correlation between input and output bits.

4 Disk encryption

Due to possibility to specify the size of the input data block algorithm 2 suites well to perform disk encryption and to other applications requiring fast software encryption. In modern computer security systems disk encryption is used as basic mechanism to protect data, therefore this kind of encryption attracts much attention of different cryptographers. Regarding requirements to algorithms of the disk encryption one can put forward the following ones: high security, high speed while implementing in software, and the 512-byte blocksize. When encrypting data blocks having large size it is quite difficult to provide full avalanche plaintext and ciphertext. One can build large block cipher using some underlying cipher with standard 64-bit (DES, Blowfish [4], RC5 [9]) or 128-bit blocksize (RC6, MARS. TWOFISH, Rijndael [10]). However to obtain strong randomization over the whole 4096-bit sector at least two encryptions both of them being performed in the chaining mode. First encryption is performed forwards and the second one is performed backwards. Large-blocksize ciphers BEAR and LEON [11] can be adopted for disk encryption, but their performance corresponding to "small" size of the sector is comparatively low. are required. The very fast stream cipher SEAL [7] providing a strong PRNG suits well for disk encryption. When using SEAL one can consider the entire disk as a single contiguous array of bytes and XORed with respective elements of the key stream generated at the output of PRNG. To perform readings or writings of specific sectors of the disk the appropriate portion of the output can be directly generated without the need to generate elements of the key stream corresponding to preceding bytes. However against some attacks [12] such simple disk encryption with SEAL is ineffective. Secure disk encryption with SEAL can be implemented in more complicated way considered in [12]. Unfortunately such application of SEAL requires the ciphertext to be larger than the plaintext, harming the natural format of the file system. Another approach consists in the design of block ciphers which natural blocksize is 4096 bits. A variant of such ciphers can be get from the algorithm 2 selecting the value $z = 128$. Because of sufficiently large number of the elementary encryption steps corresponding to the transformation of the 32-bit words the number of the encryption rounds can be reduced to four getting higher performance.

5 Conclusion

In the proposed round hash function we have used the procedure which is very close to the data encryption based on the data-dependent lookup operations. The novel feature is the use of the data-dependent initial conditions in each iteration of the round hash function except the first iteration. Due to this feature the round hash function is some one-way transformation. Thus, we have avoided the use of any key scheduling, the encryption procedure is used as a hashing one though.

The use of the data-dependent initial conditions allows one to strengthen the chaining mechanism of the iterated hash function. For this reason one can use the fixed initial conditions (set by the procedure ***Initialize***) only for transforming the first data block of the message $M = (M_1, M_2, \ldots, M_n)$. While transforming each subsequent data block one can define new initial values of the variables R, V, Y, U, and N depending on preceding data blocks. Figure 1d shows the use of the data-dependent initial conditions as an additional chaining mechanism. This variant of the hash function construction is described as follows:

$$H_i = \mathbf{h}(M_i, H_{i-1}, G_{i-1}) = \mathbf{E}_{G_{i-1}}(M_i) \oplus H_{i-1},$$

where the value G_{i-1} corresponds to the internal state of the round hash function after performing transformation of the data block M_{i-1}. Since G_{i-1} depends on M_{i-1}, we have variable initial condition for $i = 2, 3, ..., n$ (G_0 is specified). The practical use of such new chaining mechanism is very attractive, since more efficient chaining should provide more security against differential cryptanalysis.

Acknowledgement: This research was supported by EOARD/AFRL grant #1994p.

References

1. D.R.Stinson, *Cryptography. Theory and practice.*- New York, CRC Press, 1995.-434 p.

2. A.J.Menezes, P.C. von Oorschot, S.A. Vanstone. *Handbook of Applied Cryptography.* – New York, CRC Press, 1996.- 777 p.

3. B.Schneier, *Applied Cryptography, Second Eddition,* New York, John Wiley & Sons, Inc., 1966.-790 p.

4. B.Schneier, *Description of a new variable-length key, 64-bit block cipher (Blowfish) //* 1st Int. Workshop "Fast Software Encryption". Proc./. Springer-Verlag LNCS. 1994. V. 809. P.191-204.

5. Moldovyan A.A., Moldovyan N.A. *Flexible block ciphers with provably inequivalent cryptalgorithm modifications//Cryptologia.* 1998. V. XXII. No. 2. P.134-140.

6. Moldovyan A. A., Moldovyan N. A. *Software encryption algorithms for transparent protection technology // Cryptologia,* January 1998, Volume XXII No. 1. P. 56-68.

7. Rogaway Ph., Coppersmith D. *A software-optimized encryption algorithm // Journal of Cryptology.* 1998. Vol.11. No 4. P. 273-287.

8. B. Preneel, A. Bosselaers, V. Rijmen, *et al. Comments by the NESSIE Project on the AES Finalists. 24 may 2000.* (http://www.nist.gov/aes).

9. R.L. Rivest: *The RC5 encryption algorithm. Fast Software Encryption* – FSE'94 Proceedings. Springer-Verlag, LNCS Vol.1008. (1995) 86-96.

10. Proceeding of 1st Advanced encryption standard candidate conference, Venture, California, Aug 20-22 1998, (http://www.nist.gov/aes)

11. Anderson R., Biham E. *Two practical and provably secure block ciphers: BEAR and LION* // 3d Int. Conference FSE'96 Proc./ Springer-Verlag LNCS, 1996, vol.1039. P.113-120.

12. P. Crowley. *Mercy: A fast large block cipher for disk sector encryption* // 7th Int. werkshop, FSE 2000 Proc. / Springer-Verlag LNCS, 2001, vol.1978.P.49-63.

13. N.D. Goots, B.V. Izotov, A.A.Moldovyan, and N.A.Moldovyan. *Modern cryptography: Protect your data with fast block ciphers.* // Wayne, A-LIST Publishing, 2003.-400 p. (www.alistpublishing.com).

N.A.Moldovyan, D.H. Summerville, Received March 29, 2002

N.A.Moldovyan,
Specialized Center of Program Systems "SPECTR",
Kantemirovskaya str., 10, St. Petersburg 197342, Russia;
Phone/fax: 7 – 812 – 2453743
E–mail: *info@cobra.ru*

D.H. Summerville,
Binghamton University
Watson School Electrical and Computer Engineering,
PO Box 6000
Binghamton NY
Phone: 607 – 777 – 2942
E–mail: *dsummer@binghamton.edu*