Dynamic SHA

Zijie Xu

E-mail: xuzijiewz@gmail.com

Abstract. In this paper I describe the construction of Dynamic SHA family of cryptographic hash functions. They are built with design components from the SHA-2 family, but there is function R in the new hash function. It enabled us to achieve a novel design principle: *When message is changed, different rotate right operation maybe done.* It makes the system can resistant against all extant attacks. Dynamic SHA is posted[16].

Key words: Cryptographic hash function, SHA, Dynamic SHA

1 Introduction

The SHA-2 family of hash functions was designed by NSA and adopted by NIST in 2000 as a standard that is intended to replace SHA-1 in 2010 [6]. Since MD5, SHA-0 and SHA-1 was brought out, people have not stopped attacking them, and they succeed. Such as: den Boer and Bosselaers [2,3] in 1991 and 1993, Vaudenay [8] in 1995, Dobbertin [5] in 1996 and 1998, Chabaud and Joux [4] in 1998, Biham and Chen [1] in 2004, and Wang et al. [9–12] in 2005. Most well known cryptographic hash functions such as: MD4, MD5, HAVAL, RIPEMD, SHA-0 and SHA-1, have succumbed to those attacks.

Since the developments in the field of cryptographic hash functions, NIST decided to run a 4 year hash competition for selection of a new cryptographic hash standard [7]. And the new cryptographic hash standard will provide message digests of 224, 256, 384 and 512-bits.

In those attacks, we can find that when different message inputted, the operation in the hash function is no change. If message space is divided many parts, in different part, the calculation is different, the attacker will not know the relationship between message and hash value. The hash function will be secure. To achieve the purpose, I bring in data depend function R to realize the principle.

My Work: By introducing a novel design principle in the design of hash functions, and by using components from the SHA-2 family, I describe the design of a new family of cryptographic hash functions called Dynamic SHA. The principle is:

1. When message is changed, different rotate right operation maybe done.

The principle combined with the already robust design principles present in SHA-2 enabled us to build a compression function of Dynamic SHA that has the following properties:

1. There is not message expansion part.
2. The iterative part has 48 rounds. Message bits have been mixed three times.
3. The iterative part has two different functions.

2 Preliminaries and notation

In this paper I will use the same notation as that of NIST: FIPS 180-2 description of SHA-2 [6].

The following operations are applied to 32-bit or 64-bit words in Dynamic SHA:

1. Bitwise logical word operations:‘$\wedge$’–AND ,‘$\vee$’–OR,‘$\oplus$’–XOR and ‘$\neg$’–Negation.
2. Addition ‘+’ modulo 2^{32} or modulo 2^{64}.
3. The shift right operation, $SHR^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with 0≤n<32 (resp. 0≤n<64).
4.The shift left operation, $SHL^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with 0≤n<32 (resp. 0≤n<64).
5. The rotate right (circular right shift) operation, $ROTR^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with $0 \le n < 32$ (resp. $0 \le n < 64$).
6. The rotate left (circular left shift) operation, $ROTL^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with $0 \le n < 32$ (resp. $0 \le n < 64$).

Depending on the context I will sometimes refer to the hash function as Dynamic SHA, and sometimes as Dynamic SHA-224/256 or Dynamic SHA-384/512.

2.1 Functions

Dynamic SHA include two functions. The two functions are used in compression function.

2.1.1 Function G(x1,x2, x3,t)

Function G operates on three words x1,x2, x3 and an integer t, produces a word y as output. And function G as follow:

$$y = G_t(x1, x2, x3) = \begin{cases} x1 \oplus x2 \oplus x3 & t = 0 \\ (x1 \wedge x2) \oplus x3 & t = 1 \\ (\neg(x1 \vee x3)) \vee (x1 \wedge (x2 \oplus x3)) & t = 2 \\ (\neg(x1 \vee (x2 \oplus x3))) \vee (x1 \wedge \neg x3) & t = 3 \end{cases}$$

Table 2.1 function G for Dynamic SHA

The truth table for logical functions as table 2.2.

x1	x2	x3	f1	f2	f3	f4
0	0	0	0	0	1	1
0	0	1	1	1	0	0
0	1	0	1	0	1	0
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	1
1	1	1	1	0	0	0

Table 2.2. truth table for logical functions

2.1.2 Function R(x1, x2, x3, x4, x5, x6, x7, x8)

Function R operates on eight words x1, x2, x3, x4, x5, x6, x7 and x8, produces a word y as output. And function R as table 2.3.

Dynamic SHA-224/256	t0 = (((((x1 + $x2$) ⊕ $x3$) + $x4$) ⊕ $x5$) + $x6$) ⊕ $x7$ t1 = (SHR^{17}(t0) ⊕ t0) ∧ (2^{17} - 1) t2 = (SHR^{10}(t1) ⊕ t1) ∧ (2^{10} - 1) t = (SHR^{5}(t2) ⊕ t2) ∧ 31 $y = ROTR^{t}(x8)$
Dynamic SHA-384/512	t0 = (((((x1 + $x2$) ⊕ $x3$) + $x4$) ⊕ $x5$) + $x6$) ⊕ $x7$ t1 = (SHR^{36}(t0) ⊕ t0) ∧ (2^{36} - 1) t2 = (SHR^{18}(t1) ⊕ t1) ∧ (2^{18} - 1) t3 = (SHR^{12}(t2) ⊕ t2) ∧ (2^{12} - 1) t = (SHR^{6}(t3) ⊕ t3) ∧ 63 $y = ROTR^{t}(x8)$

Table 2.3. Function R for Dynamic SHA

2.2 Dynamic SHA Constants

Dynamic SHA does not use any constants.

2.3 Preprocessing

Preprocessing in Dynamic SHA is exactly the same as that of SHA-2. That means that these three steps: padding the message M, parsing the padded message into message blocks, and setting the initial hash value, H^0 are the same as in SHA-2. Thus in the parsing step the message is parsed into N blocks of 512 bits (resp. 1024 bits), and the i-th block of 512 bits (resp. 1024 bits) is a concatenation of sixteen 32-bit (resp. 64-bit) words denoted as $M_0^{(i)}, M_1^{(i)}, ..., M_{15}^{(i)}$. Dynamic SHA may be used to hash a message, M, having a length of l bits, where $0 \le l < 2^{64}$.

2.3.1 padding

2.3.1.1 Dynamic SHA-224/256

Suppose that the length of the message M is L bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation L+1+k ≡ 448 mod 512. Then append the 64-bit block that is equal to the number L expressed using a binary representation.

2.3.1.2 Dynamic SHA-384/512

Suppose that the length of the message M is L bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation L+1+k ≡ 896 mod 1024. Then append the 128-bit block that is equal to the number L expressed using a binary representation.

2.4 Initial Hash Value H^0

The initial hash value, H^0 for Dynamic SHA is the same as that of SHA-2 (given in Table 2.4).

Dynamic SHA-224	Dynamic SHA-256	Dynamic SHA-384	Dynamic SHA-512
$H_0^{(0)} = c1059ed8,$ $H_1^{(0)} = 367cd507,$ $H_2^{(0)} = 3070dd17,$ $H_3^{(0)} = f70e5939,$ $H_4^{(0)} = ffc00b31,$ $H_5^{(0)} = 68581511,$ $H_6^{(0)} = 64f98fa7,$ $H_7^{(0)} = befa4fa4,$	$H_0^{(0)} = 6a09e667,$ $H_1^{(0)} = bb67ae85,$ $H_2^{(0)} = 3c6ef372,$ $H_3^{(0)} = a54ff53a,$ $H_4^{(0)} = 510e527f,$ $H_5^{(0)} = 9b05688c,$ $H_6^{(0)} = 1f83d9ab,$ $H_7^{(0)} = 5be0cd19,$	$H_0^{(0)} = cbbb9d5dc1\ \ 059ed8,$ $H_1^{(0)} = 629a292a36\ \ 7cd507,$ $H_2^{(0)} = 9159015a30\ \ 70dd17,$ $H_3^{(0)} = 152fecd8f7\ \ 0e5939,$ $H_4^{(0)} = 67332667ff\ \ c00b31,$ $H_5^{(0)} = 8eb44a8768\ \ 581511,$ $H_6^{(0)} = db0c2e0d64\ \ f98fa7,$ $H_7^{(0)} = 47b5481dbe\ \ fa4fa4,$	$H_0^{(0)} = 6a09e667f3bcc908,$ $H_1^{(0)} = bb67ae8584caa73b,$ $H_2^{(0)} = 3c6ef372fe94f82b,$ $H_3^{(0)} = a54ff53a5f1d36f1,$ $H_4^{(0)} = 510e527fade682d1,$ $H_5^{(0)} = 9b05688c2b3e6c1f,$ $H_6^{(0)} = 1f83d9abfb41bd6b,$ $H_7^{(0)} = 5be0cd19137e2179,$

Table 2.4. The initial hash value, H^0 for Dynamic SHA

2.5 Constants

The Dynamic SHA has three constants (given in table 2.5):

Dynamic SHA-224/256	Dynamic SHA-384/512
$TT_0 = 5A827999,$ $TT_1 = 6ED9EBA1,$ $TT_2 = 8F1BBCDC,$	$TT_0 = 5A82799950\ \ A28BE6,$ $TT_1 = 6ED9EBA15C4DD124,$ $TT_2 = 8F1BBCDC\ \ 6D703EF3,$

Table 2.5. The constants for Dynamic SHA

2.6 Dynamic SHA Hash Computation

The Dynamic SHA hash computation uses functions and initial values defined in previous subsections. So, after the preprocessing is completed, each message block, $M^{(0)}, M^{(1)}, ..., M^{(N)}$, is processed in order, using the steps described algorithmically in Table 2.6.

For i = 1 to N:

{

1.Initialize eight working variables a, b, c, d, e, f, g and h with the $(i-1)^{th}$ hash value:

$$a = H_0^{(i-1)}, \quad b = H_1^{(i-1)}, \quad c = H_2^{(i-1)}, \quad d = H_3^{(i-1)},$$
$$e = H_4^{(i-1)}, \quad f = H_5^{(i-1)}, \quad g = H_6^{(i-1)}, \quad h = H_7^{(i-1)}$$

2. For t=0 to 47

{

$T = R(a,b,c,d,e,f,g,h)$

$h = g$

$g = f$

$f = e$

$e = d$

$d = G(a,b,c,t \wedge 3) + W_{t \wedge 15} + TT_{t>>4}$

$c = b$

$b = a$

$a = T$

}

3.Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}, \quad H_1^{(i)} = b + H_1^{(i-1)}, \quad H_2^{(i)} = c + H_2^{(i-1)}, \quad H_3^{(i)} = d + H_3^{(i-1)},$$
$$H_4^{(i)} = e + H_4^{(i-1)}, \quad H_5^{(i)} = f + H_5^{(i-1)}, \quad H_6^{(i)} = g + H_6^{(i-1)}, \quad H_7^{(i)} = h + H_7^{(i-1)}$$

}

Table 2.6. Algorithmic description of Dynamic SHA hash function.

The algorithm uses 1) a message schedule of sixteen 32-bit (resp. 64-bit) words, 2) eight working variables of 32 bits (resp. 64 bits) , and 3) a hash value of eight 32-bit (resp. 64-bit) words. The final result of Dynamic SHA-256 is a 256-bit message digest and of Dynamic SHA-512 is a 512-bit message digest. The final result of Dynamic SHA-224 and Dynamic SHA-384 are also 256 and 512 bits, but the output is then truncated as 224 (resp. 384) bits. The words of the message schedule are labeled $W_0, W_1, ..., W_{15}$. The eight working variables are labeled a, b, c, d, e, f, g and h and sometimes they are called “state register”. The words of the hash value are labeled $H_0^{(i)}, H_1^{(i)}, ..., H_7^{(i)}$, which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the

final hash value, $H^{(N)}$. Dynamic SHA also uses one temporary words T and three constants.

3 Security of Dynamic SHA

In this section I will make an initial analysis of how strongly collision resistant, preimage resistant and second preimage resistant Dynamic SHA is. I will start by describing our design rationale, then I will discuss the strength of the function against known attacks for finding different types of collisions.

3.0 Cryptographic Hash Functions

After preprocess message, there are some message blocks that include 512(resp.1024) bits.

Let there exist message blocks M(1),M(2),…,M(n). Let f(h,Mi) is compression function, it is as table 2.6. The operation of the iterated hash function is as follows. First, an b-bit value h(0)=IV is fixed. Then the message blocks are hashed in order. There exist f(h(i-1),M(i))=h(i) i = 1,2,...,n. As table 3.1

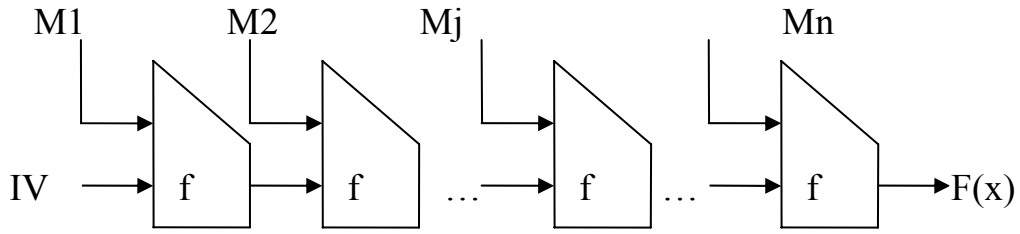

Table 3.1 The iterated construction of compression function f

When someone find collisions, he can randomly guess message blocks except for one block M(j) ,where 0≤j≤n. Then he can calculate out h(j-1) with function f and message blocks M(1),…,M(j-1) , and he can backward function f with message blocks M(j+1),…,M(n) to calculate out h(j). At last he can just find suitable M(j) that mak f(h(j-1),M(j))=h(j) to complete findding collisions. So I will discuss the security of Dynamic SHA in one block.

3.1 Properties of iterative part

In the iterative part, there are 48 rounds. In one round, there are functions G, R, one message word will be mixed.

3.2 Design rationale

The reasons for the first principle: *When message is changed, different rotate right operation maybe done.*

From the definition of function R, it is easy to know when the variable is different, the parameter n in $ROTR^n(x)$ will be different, different rotate left operation will be done.

It can guess the parameter n in $ROTR^n(x)$. Function R is called 48 times in Dynamic SHA, and in first round, the message words is not mixed, so it can just guess the parameter n 48-1=47 times. Then there are $32^{47} = 2^{235}$ (resp. $64^{47} = 2^{282}$) 47-tuple (n(1),…,n(i),…,n(47)), where n(i) is the parameter n of $ROTR^n(x)$ in i-th round.

If someone guess the parameter of function R. There are 2^{235} (resp. 2^{282}) 47-tuple (n(1),…, n(i),…,n(47)). A given 47-tuple define different calculation, so 47-tuple (n(1),…, n(i),…,n(47)) divide the message space into 2^{235} (resp. 2^{282}) parts. In different part, the calculation is different. When message is changed, the 47-tuple (n(1),…, n(i),…,n(47)) maybe change, different rotate right operation maybe done.

Controlling the differentials is hard in Dynamic SHA:

In Dynamic SHA, it is known that when message is changed, the calculation will be different. To analyze Dynamic SHA, it need the unchangeable formulas that represent function R. There are three ways to analyze Dynamic SHA:

1. Guess the parameters of function R. This way is select a part in the message value space. And the message space is divided into 2^{235} (resp. 2^{282}) parts. In different part, the calculation is different. In a part, the average number of message value is $2^{512-235} = 2^{277}$ (resp. $2^{1024-282} = 2^{742}$). Then the average number of collisions for a hash value is $2^{277-224} = 2^{53}$ (resp. 2^{21} , 2^{358} , 2^{230}), it less than

$2^{512-224} = 2^{288}$ (resp. $2^{256}, 2^{640}, 2^{512}$). If attacker selects a part, he will have a calculation. To a calculation, the average number of collisions for a hash value is $2^{512-224} = 2^{288}$ (resp. $2^{256}, 2^{640}, 2^{512}$). If someone develop an algorithm to find collision, then the probability of find the collision is $2^{53-288} = 2^{-235}$ (resp. $2^{-235}, 2^{-282}, 2^{-282}$).

2. Someone can use Algebraic Normal Form (ANF) to represent Dynamic SHA, but the ANFs that represent function R has up to 2^{256} (resp. 2^{512}) monomials. If constitute the Arithmetic function based on ANF, the degree of the Arithmetic function represents function R is up to 256(resp. 512), and has up to 2^{256} (resp. 2^{512}) monomials.
3. Someone can constitute Arithmetic functions to represent Dynamic SHA as in Appendix 2. But the Arithmetic function that represent function R is complex exponential function with round-off instruction. After 48 rounds, the Arithmetic function that represent function R will be very huge.

3.3 Finding Preimages of Dynamic SHA

To a hash function f(·), it need satisfy:

Given hash value H=f(M), it is hard to find message M that meet H=f(M).

There are two ways to find preimages of a hash function:

1,From the definition of Dynamic SHA it follows that from a given hash digest it is possible to perform backward iterative steps by guessing values that represent some relations between working variables of the message words. For that purpose let us use the following notation:

- For every iterative round t = 0,1,...,47, variables that are on the left side of the assignment (equality sign '=') will be denoted by $a_t, b_t, ..., h_t$ while variables that are on the right side of the assignment will be denoted by $a_{t-1}, b_{t-1}, ..., h_{t-1}$.

1. Initialize eight variables $a_{47}, b_{47}, ..., h_{47}$

2.For t=46 to -1

{

$T = a_{t+1}$

$a_t = b_{t+1}$

$b_t = c_{t+1}$

$c_t = G(a_t, b_t, (d_{t+1} - C0_{t+1} - TT_{(t+1)>>4}), (t+1) \wedge 3)$

$d_t = e_{t+1}$

$e_t = f_{t+1}$

$f_t = g_{t+1}$

$g_t = h_{t+1}$

$h_t = R1(a_t, b_t, c_t, d_t, e_t, f_t, g_t, T)$

}

Table 3.2. Backward recurrence expressions of Dynamic SHA. Note that the relations for the variables $C0_t$ are given in (2)

With that notation we can write the backward recurrence expressions as it is done in Table 3.2. Function R1 as table 3.3:

-The initialization of the variables $a = H_0^{i-1}$, $b = H_1^{i-1}$, $c = H_2^{i-1}$, $d = H_3^{i-1}$, $e = H_4^{i-1}$, $f = H_5^{i-1}$, $g = H_6^{i-1}$, $h = H_7^{i-1}$, will be denoted as equations (2):

$$\begin{cases} C0_0 = W_0 \\ ..... \\ C0_{15} = W_{15} \\ C0_{16} = W_0 \\ ..... \\ C0_{31} = W_{15} \\ C0_{32} = W_0 \\ ..... \\ C0_{47} = W_{15} \end{cases} \quad (1)$$

$$\begin{cases} a_{-1} = H_0^{i-1} \\ ..... \\ h_{-1} = H_7^{i-1} \end{cases} \quad (2)$$

Dynamic SHA-224/256	$t0 = (((((a_t + b_t) \oplus c_t) + d_t) \oplus e_t) + f_t) \oplus g_t$ $t1 = (SHR^{17}(t0) \oplus t0) \wedge (2^{17} - 1)$ $t2 = (SHR^{10}(t1) \oplus t1) \wedge (2^{10} - 1)$ $t = (SHR^{5}(t2) \oplus t2) \wedge 31$ $y = ROTR^{32-t}(T)$
Dynamic SHA-384/512	$t0 = (((((a_t + b_t) \oplus c_t) + d_t) \oplus e_t) + f_t) \oplus g_t$ $t1 = (SHR^{36}(t0) \oplus t0) \wedge (2^{36} - 1)$ $t2 = (SHR^{18}(t1) \oplus t1) \wedge (2^{18} - 1)$ $t3 = (SHR^{12}(t2) \oplus t2) \wedge (2^{12} - 1)$ $t = (SHR^{6}(t3) \oplus t3) \wedge 63$ $y = ROTR^{64-t}(T)$

Table 3.3. functions R1 for Dynamic SHA

Now, we have the equations (1) as a one system of 48 equations with 16 unknown variables. It is a system over GF(2) or over $\mathbb{Z}2^{32}$ (resp. $\mathbb{Z}2^{64}$).

The size of $C0_0,...,C0_{47}$ space is $2^{48\times w}$, The size of $W_0,...,W_{15}$ space is $2^{16\times w}$, where w is bit-length of $C0_0,...,C0_{47}$, $W_0,...,W_{15}$. The number of the mwssage value that has same hash value is $2^{16\times w-8\times w} = 2^{8\times w}$. he probability of there is solution for equation (1) is $2^{8\times w-48\times w} = 2^{-40\times w}$.

In first sixteen rounds, all message words are guessed. And there still are 32 words $C0_0,...,C0_{31}$ that had not been guessed. Backwarding iterative steps by guessing values is not better than random guessing in Dynamic SHA.

2, The probability of random guess of finding preimages is 2^{-224} (resp. 2^{-256}, 2^{-384}, 2^{-512}).

3.4 Finding Second Preimages of Dynamic SHA

To a hash function f(·), it need satisfy:

Given M, it is hard to find M'≠ M s.t. f(M) = f(M').

There are five ways to find second preimages of a hash function:

1, Get hash value H=f(M) of message M, and find different message M'≠ M that has hash value H= f(M'). In section 3.3, it is known that it is hard to calculate out the message M' from given hash value H.

2, Given M, and find out the relationship between the difference △M and the difference △H=f(M+△M)-f(M). And find out △M≠0 that make △H=0. To do this, someone will set up some system of equations obtained from the definition of the hash function, then trace forward and backward some initial bit differences that will result in fine tuning and annulling of those differences and finally obtain Second Preimages. It need know the unchangeable formulas that represent hash function f. In Dynamic SHA, when message is changed, the calculation maybe different. To get unchangeable formulas that represent hash function f, it need get ANFs for Dynamic SHA. And the ANFs for function R have up to 2^{256} (resp. 2^{512}) monomials.

3. To get unchangeable formulas that represent hash function f. It can constitute Arithmetic functions to represent Dynamic SHA. And the Arithmetic functions that represent function R is exponential function with round-off instruction. Or someone had to constitute high degree Arithmetic function to represent function R. And the degree of the Arithmetic function is up to 256-degree(resp. 512-degree), and have up to 2^{256} (resp. 2^{512}) monomials.

4. Guess the parameters of function R. This way is select a part in the message value space. And the message space is divided into 2^{235} (resp. 2^{282}) parts. In different part, the calculation is different. In a part, the average number of message value is $2^{512-235}=2^{277}$ (resp. $2^{1024-282}=2^{742}$). Then the average number of collisions for a hash value is $2^{277-224}=2^{53}$ (resp. 2^{21}, 2^{358}, 2^{230}), it less than $2^{512-224}=2^{288}$ (resp. $2^{256},2^{640},2^{512}$). If attacker selects a part, he will have a calculation. To a calculation, the average number of collisions for a hash value is $2^{512-224}=2^{288}$ (resp. $2^{256},2^{640},2^{512}$). If someone develop an algorithm to find collision, then the probability of find the second preimages is $2^{53-288}=2^{-235}$ (resp. $2^{-235},2^{-282},2^{-282}$).

5. The probability of random guess of finding second preimages is 2^{-224} (resp. 2^{-256}, 2^{-384}, 2^{-512}).

3.5 Finding Collisions in Dynamic SHA

To a hash function f(·), it need satisfy:

It is hard to find different M and M’ s.t. f(M) = f (M’).

There are five ways to find Collisions of a hash function:

1, Fix message M, and find different message M’ that has hash value H=f(M). then the problem become find Second Preimages of the hash function.

2. Find out the relationship between the (M, M’) and the difference △H=f(M)-f(M’). And find out (M,M’) that make △H=0. To do this, someone will set up some system of equations obtained from the definition of the hash function, then trace forward and backward some initial bit differences that will result in fine tuning and annulling of those differences and finally obtain collisions. It need know the unchangeable formulas that represent hash function f. In Dynamic SHA, when message is changed, the calculation maybe different. To get unchangeable formulas that represent hash function f, it need get ANFs for Dynamic SHA. And the ANFs for function R have up to 2^{256} (resp. 2^{512}) monomials.

3. To get unchangeable formulas that represent hash function f. It can constitute Arithmetic functions to represent Dynamic SHA. And the Arithmetic functions that represent function R is exponential function with round-off instruction. Or someone had to constitute high degree Arithmetic function to represent function R. And the degree of the Arithmetic function is up to 256-degree(resp. 512-degree), and have up to 2^{256} (resp. 2^{512}) monomials.

4. Guess the parameters of function R. This way is select a part in the message value space. And the message space is divided into 2^{235} (resp. 2^{282}) parts. In different part, the calculation is different. In a part, the average number of message value is $2^{512-235}=2^{277}$ (resp. $2^{1024-282}=2^{742}$). Then the average number of collisions for a hash value is $2^{277-224}=2^{53}$ (resp. 2^{21} , 2^{358} , 2^{230}), it less than $2^{512-224}=2^{288}$ (resp. $2^{256}, 2^{640}, 2^{512}$). If attacker selects a part, he will have a calculation. To a calculation, the average number of

collisions for a hash value is $2^{512-224} = 2^{288}$ (resp. $2^{256}, 2^{640}, 2^{512}$). If someone develop an algorithm to find collision, then the probability of find the collision is $2^{53-288} = 2^{-235}$ (resp. $2^{-235}, 2^{-282}, 2^{-282}$).

5. The attack base on the birthday paradox. the workload for birthday attack is of O(2^{112}) (resp. O(2^{128}) O(2^{192}) O(2^{256})).

3.6 Finding collisions in the reduced compression function of Dynamic SHA

If the bits in message are mixed one time, the system will be weak, someone can backward Dynamic SHA as table 3.2 show.

If there are more than 32 rounds, the bits in message are mixed at least twice, if attacker backward Dynamic SHA as table 3.2 show, he will have a system of 32 equation with 16 unknown variables, The probability of there is solution for the system is 2^{-512} (resp. 2^{-1024}). And the message space is divided into 2^{155} (resp. 2^{186}) parts, in a part, there are 2^{357} (resp. 2^{838}) message values. The average number of collisions is 2^{133} (resp. 2^{101}, $2^{454}, 2^{326}$). To a calculation, The average number of collisions is 2^{288} (resp. 2^{256}, $2^{640}, 2^{512}$). If an algorithm is developed to find collision for a calculation, then the probability of find the collision is 2^{-155} (resp. $2^{-155}, 2^{-186}, 2^{-186}$).

3.7 Security of message digest truncations

3.7.1 Security of message digest truncations of Dynamic SHA-224

The final result of Dynamic SHA-224 include eight working variables a,b,c,d,e,f,g,h, it iclude 256 bits. The output of Dynamic SHA-224 include seven working variables a,b,c,d,e,f,g, it iclude 224 bits.

So the length of the final result of Dynamic SHA-224 is 256, and the length of the output of Dynamic SHA-224 are 224. The size of the space of final result of Dynamic SHA-224 is 2^{256}, The size of the space of output of Dynamic SHA-224 is 2^{224}. To given output 7-tuple(a', b', c', d', e', f', g'), there exist 2^{32} working variables value h that make 8-tuple (a', b', c', d', e', f', g', h) has same output 7-tuple(a', b', c', d', e', f', g').

To a given output of Dynamic SHA-224, there are 2^{32} final result that has the given output. And the probability of find out a message that has the given final result is 2^{-256}. So the probability of find out a message that has the given given output is $2^{-256} \times 2^{32} = 2^{-224}$.

3.7.2 Security of message digest truncations of Dynamic SHA-384

The final result of Dynamic SHA-384 include eight working variables a,b,c,d,e,f,g,h, it iclude 512 bits. The output of Dynamic SHA-384 include six working variables a,b,c,d,e,f, it iclude 384 bits.

So the length of the final result of Dynamic SHA-384 is 512, and the length of the output of Dynamic SHA-384 are 384. The size of the space of final result of Dynamic SHA-384 is 2^{512}, The size of the space of output of Dynamic SHA-384 is 2^{384}. To given output 6-tuple(a', b', c', d', e', f'), there exist $2^{2\times64} = 2^{128}$ 2-tuple (g,h) that make 8-tuple (a', b', c', d', e', f', g, h) has same output 6-tuple(a', b', c', d', e', f').

To a given output of Dynamic SHA-384, there are 2^{128} final result that has the given output. And the probability of find out a message that has the given final result is 2^{-512}. So the probability of find out a message that has the given given output is $2^{-512} \times 2^{128} = 2^{384}$.

4 Improvements

There are some improvements for Dynamic SHA:

1. To reduce the times that message bits mixed, the message words are mixed three times. To get higher security, it can increase the number of message words mixed times. It will increase the times that message bits are mixed.

2. Function G can be design as data-dependent function. And it will increase system calculation and the number of the parts that message value space had been divided.

An examlep as follow:

The new function G operates on three words x1,x2, x3, produces a word y as output. function G include two function.

And function G1 operates on two words x1,x2 and produce an integer t. function G1 as follow:

Dynamic SHA-224/256	$t0 = x1 + x2$ $t1 = SHR^{16}(t0) \oplus t0$ $t2 = SHR^{8}(t1) \oplus t1$ $t3 = SHR^{4}(t2) \oplus t2$ $t = (SHR^{2}(t2) \oplus t2) \wedge 3$
Dynamic SHA-384/512	$t0 = x1 + x2$ $t1 = SHR^{32}(t0) \oplus t0$ $t2 = SHR^{16}(t1) \oplus t1$ $t3 = SHR^{8}(t2) \oplus t2$ $t4 = SHR^{4}(t3) \oplus t3$ $t = (SHR^{2}(t4) \oplus t4) \wedge 3$

Table 4.1. Function G1 for Dynamic SHA

Function G2 operates on three words x1,x2, x3 and an integer t that produced in function G1, function G2 as follow:

$$y = G_t(x1, x2, x3) = \begin{cases} x1 \oplus x2 \oplus x3 & t = 0 \\ (x1 \wedge x2) \oplus x3 & t = 1 \\ (\neg(x1 \vee x3)) \vee (x1 \wedge (x2 \oplus x3)) & t = 2 \\ (\neg(x1 \vee (x2 \oplus x3))) \vee (x1 \wedge \neg x3) & t = 3 \end{cases}$$

Table 4.2 function G2 for Dynamic SHA

The ANFs and Arithmetic functions that represent message expansion has up to $2^{65}(resp.2^{129})$ monomials. The degree of Arithmetic functions that represent message expansion is up to 65(resp.129), and has up to $2^{65}(resp.2^{129})$ monomials.

3. In Keyed Hash function, the initial hash value is random variable to attacker. If Dynamic SHA is used in Keyed Hash function, by theorem 3, it is easy know that the probability of hash value is 2^{-224} (resp. 2^{-256} 2^{-384} 2^{-512}).

There are some ways that we can adopt to get random initial hash value, for example: $IV_i = IV_{i-1} + c$, IV_i is i-th initial hash value, c is constant and c is odd number. To do this, it need new communication protocol.

4. If some algorithms that based on Arithmetic functions are developed to break Dynamic SHA. The message expansions will increase the degree of the Arithmetic function that represents Dynamic SHA. If the message expansions is data depend function, the degree of the Arithmetic function that represents the message expansions maybe be up to 512(resp.1024). It will increase the ability that resists differential analysis

The message expansion maybe makes some hash values have more probability than other hash value. With improvement 3, all hash value will have same probability.

An examlep as follow:

Use a data-depend function as message expansion and the iterative part include 64 rounds. The message expansion and the fourth iterative part as follow:

Dynamic SHA-224/256	$t0 = ((((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7) + x8$ $t1 = ((((((x9 + x10) \oplus x11) + x12) \oplus x13) + x14) \oplus x15) + x16$ $p(i) = SHR^{4\times i}(t0) \wedge (15) \qquad 0 \le i \le 7$ $p(i) = SHR^{4\times(i-8)}(t1) \wedge (15) \qquad 8 \le i \le 15$ $t2 = \sum_{i=0}^{15} p(i)$ $w_{i+15} = w_i \oplus w_{p(i)\oplus t2} \qquad 0 \le i \le 15$
Dynamic SHA-384/512	$t0 = ((((((((((((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7) +$ $+ x8) \oplus x9) + x10) \oplus x11) + x12) \oplus x13) + x14) \oplus x15) + x16$ $p(i) = SHR^{4\times i}(t0) \wedge (15) \qquad 0 \le i \le 15$ $t1 = \sum_{i=0}^{15} p(i)$ $w_{i+15} = w_i \oplus w_{p(i)\oplus t1} \qquad 0 \le i \le 15$

Table 4.3. message expansion for Dynamic SHA

$w_i \quad 0 \le i \le 15$ are message words and $w_i \quad 16 \le i \le 31$ are message expansion words, and a new constant $TT_3 = 8f1bbcdc$ (resp. $TT_3 = 8f1bbcdc7a6d76e9$) will be used, and the iterative part as table 4.4.

The ANFs and Arithmetic functions that represent message expansion has up to $2^{512}(resp.2^{1024})$ monomials. The degree of Arithmetic functions that represent message expansion is up to 512(resp.1024).

2. For t=0 to 63

{ $T = R(a,b,c,d,e,f,g,h)$

$h = g$

$g = f$

$f = e$

$e = d$

$d = G(a,b,c,t \wedge 3) + W_{t \wedge 31} + TT_{t>>4}$

$c = b$

$b = a$

$a = T$

}

Table 4.4 the iterative part for Dynamic SHA

5. Support of HMAC, randomized hashing function and Pseudo-random function

Dynamic SHA can be used in different situation, such as: HMAC, randomized hashing function and Pseudo-random function.

5.1 Support of HMAC

5.1.1 Constitute HMAC with Dynamic SHA

If there is a hash function H(.), the size of message block is b. The definition of HMAC is:

$$HMAC(M) = H((K^+ \oplus opad) \| H((K^+ \oplus ipad) \| M))$$

Where:

ipad = 00110110... repeat 0x36 64(resp.128) times.

opad = 01011100… repeat 0x5c 64(resp.128) times.

K = user key.

K^+ = pad (b-len(K)) '0' to user key K. len(K) is length of user key K.

M = message that input HMAC.

|| = connection operation.

From the definition of HMAC, it is known that it can use Dynamic SHA-224/256/384/512 to constitute HMAC that produce 224(resp. 256,

384, 512)-bit message authentication code.

If the size of message block of hash function H is b, and the bit-length of hash value is n. The steps as follow:

1. pad '0' to the key K, and get the K^+ that include b bits.
2. let $S_i = ipad \oplus K^+$ and $S_o = opad \oplus K^+$
3. get h1=H(S_i||M). M is message.
4. get HMAC=H(S_o||h1)

5.1.2 Security of HMAC

Bellare, Canetti, R. and Krawczyk[BELL96a] had define (ε,t,q,L)-weakly collision-resistant as follow:

Definition 5.1: We say that a family of keyed hash functions f is (ε,t,q,L)-weakly collision-resistant if any adversary that is not given the key k, is limited to spend total time t, and sees the values of the function Fk computed on q messages m1,m2,...,mq of its choice, each of length at most L, cannot find messages m and m' for which Fk(m) = Fk(m') with probability better than ε .

Bellare, Canetti, R. and Krawczyk[BELL96a] had proved the theorem as follow:

Theorem 5.1 If the keyed compression function f is an ε_f,q,t,b)-secure MAC on messages of lengthbbits, and the keyed iterated hash F is (ε_F,q,t,L)-weakly collision-resistant then the NMAC function is an (ε_{f+F},q,t,L)-secure MAC.

Because the attacker need at least 2^{112} (resp. 2^{128}, 2^{192}, 2^{512}) different message to find collision of Dynamic SHA. By theorem 5.1, it is known that if someone want to find collision of HMAC that constituted with Dynamic SHA, he need 2^{112} (resp. 2^{128}, 2^{192}, 2^{512}) different (message, MAC) that produced with same key. And the attacker has not the key, he can not produce these (message, MAC) off-line. On a 1 Gbit/sec communication link, one would need more than 2^{81} seconds to process all the data required by such an attack.

5.2 Support of randomized hashing function

5.2.1 randomized hashing function

In Draft NIST SP 800-106[17], Randomized Hashing function RF is as follow:

$RF(rv,m)=F(rv \,||\, M \oplus Rv \,||\, PL(rv))$

Where:

RF = Randomized Hashing function

rv = a random bit string that bit-length<1025

m = input message

F = hash function.

M = pad '1' and some '0' to m.

1.if m longer than (|rv|-2), just pad '1'.

2. if m shorter than (|rv|-1), just pad '1' and some '0' to make length M equal |rv|.

Rv =repeat rv some times, and truncated as |M|

PL(rv) = 16-bit binary string that describe the bit-length of rv

|x| = bit-length of x.

|| = connection operation.

1. rv is a random bit string that bit-length<1025, m is message.
2. let rlen=|rv|
3. if (|m|>(rlen-2)) then M1=m||'1'
4. if (|m|>(rlen-2)) then M1=m||'1'
5. if (|m|<(rlen-1)) then M1=m||'1'||$\underbrace{\text{'0'}||\ldots||\text{'0'}}_{\text{rlen-|m|-1}}$
6. let Rv1= $\underbrace{rv||\ldots||rv}_{|M|/rlen+1}$
7. let Rv= Rv1 truncated as |M1|
8. let M=rv||(M1 ⊕ Rv) ||16bitlen(rv)
9. return M

Table 5.1 function PM for Randomized Hashing function

1. A = n. Comment: A is an integer.
2. For (integer i = 15 down to 0)
    2.1 B = A mod 2. Comment: B is an integer,
    2.2 If (B = 0), then
        bi = "0". Comment: bi is a single "0" bit.
      Else
        bi = "1" . Comment bi is a single "1" bit.
    2.3 A = $\lfloor A/2 \rfloor$
3. 16bit=b0||b1||…||b15;
4. return 16bit;

Table 5.2 function PL for Randomized Hashing function

There exist two function PM anf PL as table 5.1 and 5.2 show.

Function PM operate on a message m and a random bit string rv, produce a new message M.

Function PL operate on an integer, produce a 16-bit bit string.

From the definition of Randomized Hashing function, it is easy known that Dynamic SHA can be used function F in the definition. Dynamic SHA are used in Randomized Hashing function as follow:

RF(rv,m)=F(PM(rv,m)), where F is Dynamic SHA.

5.2.2 Security of randomized hashing function

If the randomized hashing function is constituted with hash function F.

Then if someone have a message m1 and a random bit string rv1, then he can find (rv2,m2) that make RF(rv1,m1)= RF(rv2,m2) as table 5.3 show(the bit order is started from 1.):

1. set m2len is the length of the message m2 and r2len the length of the random bit string rv2. where r2len<1025.
2. if m2len>r2len-2,
    2.1 find a (r2len+m2len+1) bit-length message m3 that make RF(rv1,m1)=F(m3||PL(r2len)). And the (r2len+m2len+1)-th

bit of m3 b1 and the (((m2len+1) mod r2len)+1)-th bit of m3 b2 satisfy the follow requirement: b1⊕b2=1.

2.2 the first r2len bits is random bit string rv2.

2.3. let Rv2= $\underbrace{rv2||\ldots||rv2}_{|m3|/r2len+2}$

2.4 let Rv= Rv2 truncated as (m2len+r2len)

2.5 let m3'= m3⊕Rv

2.6 from the (r2len+1)-th bit to(m2len+r2len)-th bit of m3' is m2.

2.7 return m2 and rv2

3. if m2len<r2len-1

3.1 find a (r2len+r2len) bit-length message m3 that make RF(rv1,m1)=F(m3||PL(r2len)). And m3 satisfy the follow two requirement:

3.1.A: the (r2len+m2len+1)-th bit of m3 b1 and the (m2len+1)-th bit of m3 b2 satisfy the follow requirement: b1⊕b2=1.

3.1.B: if m3=($b_1,b_2,\ldots,b_{2*r2len}$), then ($b_{m2len+2},\ldots,b_{r2len}$)=($b_{m2len+r2len+2},\ldots,b_{2*r2len}$)

3.2 the first r2len bit is random bit string rv

3.3. let Rv= rv2||rv2

3.4 let m3'= m3⊕Rv

3.5 from the (r2len+1)-th bit to(m2len+r2len)-th bit of m3' is m2.

3.6 return m2 and rv2

4. if not find message m3 that make RF(rv1,m1)=F(m3||PL(r2len)). Set the length of the message m2 and the length of the random bit string rv2. goto step 2.

In this way to find a message m2 and a random bit string rv2 (rv2,m2)≠(rv1,m1) that make RF(rv1,m1)= RF(rv2,m2), it need find the Preimages of hash function F or Second Preimages of hash function F. it is hard find Preimages or Second Preimages of Dynamic SHA. The probability of finding Preimages or second preimages is 2^{-224} (resp.

2^{-256}, 2^{-384}, 2^{-512}).

5.3 Support of Pseudo-random function

In section 10 of NIST SP 800-90[18], NIST has publish "Deterministic Rrandom Bit Generator(DRBG) Mechanisms Based on Hash Functions."

5.3.1 Support of HMAC based Pseudo-random function

In section 10.1.2 of NIST SP 800-90[18], NIST has publish "HMAC_DRBG.". It specify a construction of Pseudo-random function that base on HMAC.

Here I specify a construction of Pseudo-random function based on the "HMAC_DRBG." of NIST SP 800-90[18]. And the HMAC specified in setction 5.1 will de used in the construction of Pseudo-random function.

5.3.1.1 Functions

Three function are used in the construction of HMAC based Pseudo-random function.

5.3.1.1.1 Function *Updata(provided_data, K, V)*

Function *Updata* operate on three bit strings *provided_data, K, V*. and produce a new key *K* and a new string *V*. Function *Updata* as table 5.3 show:

1. *K*=HMAC(*K*, *V*||0x00|| *provided_data*)
2. *V*=HMAC(*K*, *V*).
3. If (provided_data=NULL), then Return *K*, *V*
4. *K*=HMAC(*K*,*V*||0x01|| *provided_data*)
5. *V*= HMAC(*K*, *V*)
6. Return *K*, *V*

Table 5.3 Function Updata of HMAC-based Pseudo-random function

5.3.1.1.2 Function *Instantiate (entropy_input, nonce, personalization_string)*

Function *Instantiate* initialize some system parameters, when HMAC

based Pseudo-random function start.

Function *Instantiate* operate on three bit strings *entropy_input, nonce, personalization_string.*

entropy_input is a string of bits obtained from the source of entropy input.

nonce is a bit string.

a. An unpredictable value with at least 56 (resp. 128, 96, 256) bits of entropy.

b. A value that is expected to repeat no more often than a 56 (resp. 128, 96, 256)-bit random string would be expected to repeat.

personalization_string is a string received from the consuming application..

Function *Instantiate* produce a key K , a string V and an integet *reseed_counter* . Function *Instantiate* as table 5.4 show:

1. *seed_material* = *entropy_input* || *nonce* || *personalization_string*.
2. K = 0x00 00...00. Comment: outlen bits.
3. V = 0x01 01...01. Comment: outlen bits.
Comment: Update Key and V.
4. (K, V) = Update (*seed_material*, ***K***, V).
5. *reseed_counter* = 1.
6. Return *K, V, reseed_counter*

Table 5.4 Function Instantiate of HMAC-based Pseudo-random function

5.3.1.1.3 Function *Reseed (V, K, reseed_counter, entropy_input, additional_input)*

If too many pseudo-random number were produced with same parameters, someone will have enouhg data to attack the system. So after produce some pseudo-random number, the system parameters must be reseted. HMAC based Pseudo-random function will reset system parameters after produce no more than 2^{48} pseudo-random number.

The function of function *Reseed* is reset system parameters. Function *Reseed* operate on four bit strings *V, K, entropy_input, additional_input* and an integer *reseed_counter.*. Produce two new bit strings *V, K,* and an

new integer *reseed_counter*. Function *Reseed* is as table 5.5 show:

1. *seed_material* = *entropy_input* || *additional_input*.
2. (*K*, *V*) = Update (*seed_material*, *K*, *V*).
3. *reseed_counter* = 1.
4. Return *V*, *K*, *reseed_counter*

Table 5.5 Function Reseed of HMAC-based Pseudo-random function

5.3.1.2 HMAC based Pseudo-random function

When HMAC based Pseudo-random function start, system will call tunction *Instantiate* to initialize some system parameters. And then pseudo-random number will be produced as follow steps:

1. If *reseed_counter* > 2^{48}, then return an indication that a reseed is required.
2. If *requested_number_of_bits*> 2^{19}, then return an signal that the *requested_number_of_bits* is error.
3. If *additional_input*≠ Null, then (*Key, V*) = Update (*additional_input, Key, V*).
4. *temp* = Null.
5. While (len (temp) < *requested_number_of_bits*) do:
   5.1 *V* = HMAC (*Key, V*).
   5.2 *temp* = *temp* || *V*.
6. *returned_bits* = Leftmost *requested_number_of_bits* of *temp*.
7. (*Key, V*) = Update (*additional_input, Key, V*).
8. *reseed_counter* = *reseed_counter* + 1.
9. Return *returned_bits*, and the new values of *Key, V* and *reseed_counter.*

In the steps:

reseed_counter is the number of pseudo-random number had been produced.

additional_input is a string received from the consuming application.

Key is the key will be used in HMAC.

V is the bit string will be hashed in HMAC.

requested_number_of_bits is the number of bits of the pseudo-random numberwill be produced. *requested_number_of_bits* no bigger than 2^{35}.

returned_bits is the produced pseudo-random number.

In the process of produce pseudo-random number, The values of V and Key are the critical values. So it must prevent from reveal the values of V and Key.

5.3.1.3 Security of Pseudo-random function based HMAC

In the Pseudo-random function based HMAC, the key and the 'message' data V is protected. If someone attack the system, he must enough data that produced with same key, but in the Pseudo-random function based HMAC, the max number of the bits that produced with same key is 2^{35}, and every time the max number of bits requested is 2^{19}. The max number of bits that produced with same Key is $2^{35+19} = 2^{54}$. To find collision of HMAC that constituted with Dynamic SHA, it need at least 2^{112} (resp. 2^{128}, 2^{192}, 2^{512}) different (message, MAC) that produced with same key. So the attacker can not get enough (message, MAC) to find collision of HMAC that constituted with Dynamic SHA.

The attacker can test all (Key, V) to find the (Key, V) that is used, but the bit-length of Key and V is hash value of hash function. Then the bit-length of (Key, V) is $224 + 224 = 448$ (resp. $256 + 256 = 512$, $384 + 384 = 768$, $512 + 512 = 1024$).

5.3.2 Support of non-HMAC based Pseudo-random function

In section 10.1.1 of NIST SP 800-90[18], NIST has publish "Hash_DRBG". It specify a construction of Pseudo-random function that not base on HMAC.

Here I specify a construction of Pseudo-random function based on the "Hash_DRBG" of NIST SP 800-90[18].

5.3.2.1 Functions

Four function are used in the construction of non-HMAC based

Pseudo-random function

5.3.2.1.1 Function *Hash_df (input_string, no_of_bits_to_return)*

Function *Hash_df* operate on three bit strings *input_string* and an integer *no_of_bits_to_return.* And produce a string *requested_bits.*. Function *Hash_df* as table 5.6 show:

1. *temp* = the Null string.
2. len=$\frac{no\_of\_bit\_to\_return}{outlen}$.
3. *counter* = 0x01
4. For i = 1 to len do

Comment : In step 4.1, *no_of_bits_to_return* is used as a 32-bit string.

4.1 *temp*=*temp*||Hash(*counter*||*no_of_bits_to_return*|| *input_string*).

4.2 *counter* = *counter* + 1.

5. *requested_bits* = Leftmost (*no_of_bits_to_return*) of temp.
6. Return *requested_bits*

Table 5.6 Function Hash_df of non-HMAC based Pseudo-random function

The *outlen* in table 5.6 is the bit-length of the hash function.

5.3.2.1.2 Function *Instantiate (entropy_input, nonce, personalization_string)*

Function *Instantiate* initialize some system parameters, when HMAC based Pseudo-random function start.

Function *Instantiate* operate on three bit strings *entropy_input, nonce, personalization_string.*

entropy_input is a string of bits obtained from the source of entropy input.

nonce is a bit string. *nonce* is either:

a. An unpredictable value with at least 56 (resp. 128, 96, 256) bits of entropy.

b. A value that is expected to repeat no more often than a 56 (resp. 128, 96, 256)-bit random string would be expected to repeat.

personalization_string is a string received from the consuming application..

seedlen is a constant depend the hash function,

if Dynamic SHA-224/256 *seedlen* = 440

if Dynamic SHA-384/512 *seedlen* = 888

Function *Instantiate* produce a key K , a string V and an integet *reseed_counter* . Function *Instantiate* as table 5.7 show:

1. *seed_material* = *entropy_input* || *nonce* || *personalization_string*.
2. *seed* = Hash_df (*seed_material, seedlen*).
3. *V* = *seed*.
4. *C* = Hash_df ((0x00 || *V*), *seedlen*).
5. *reseed_counter* = 1.
6. Return *V, C,* and *reseed_counter*

Table 5.7 Function Instantiate of HMAC-based Pseudo-random function

5.3.2.1.3 Function *Reseed (V, K, reseed_counter, entropy_input, additional_input)*

If too many pseudo-random number were produced with same parameters, someone will have enouhg data to attack the system. So after produce some pseudo-random number, the system parameters must be reseted. HMAC based Pseudo-random function will reset system parameters after produce no more than 2^{48} pseudo-random number.

The function of function *Reseed* is reset system parameters. Function *Reseed* operate on four bit strings *V, K, entropy_input*, *additional_input* and an integer *reseed_counter.* Prodece two bit strings *V, K,* and an new integer *reseed_counter.* Function *Reseed* is as table 5.8 show:

1. *seed_material* = 0x01 || *V* || *entropy_input* || *additional_input.*
2. *seed* = Hash_df (*seed_material, seedlen*).
3. *V* = *seed*.
4. *C* = Hash_df ((0x00 || *V*), *seedlen*). Comment: Preceed with a byte of all zeros.
5. *reseed_counter* = 1.
6. Return *V, C,* and *reseed_counter*.

Table 5.8 Function Reseed of HMAC-based Pseudo-random function

The *seedlen* in table 5.8 is a constant depend the hash function,

if Dynamic SHA-224/256 *seedlen* = 440

if Dynamic SHA-384/512 *seedlen* = 888

5.3.2.1.4 Function *Hashgen (requested_number_of_bits, V)*

The function of function *Hashgen* operate on one bit strings *V* and an integer *requested_number_of_bits.* Prodece a bit strings *returned_bits.* Function *Hashgen* is as table 5.9 show:

1. $m = \frac{no\_of\_bit\_to\_return}{outlen}$.
2. *data* = *V*.
3. *W* = the Null string.
4. For i = 1 to m
   - 4.1 *wi* = Hash (*data*).
   - 4.2 *W* = *W* || *wi*.
   - 4.3 *data* = (*data* + 1) mod $2^{seedlen}$.
5. *returned_bits* = Leftmost (*requested_no_of_bits*) bits of *W*
6. Return *returned_bits*.

Table 5.9 Function Reseed of HMAC-based Pseudo-random function

The *seedlen* in table 5.9 is a constant depend the hash function,

if Dynamic SHA-224/256 *seedlen* = 440

if Dynamic SHA-384/512 *seedlen* = 888

5.3.2.2 non-HMAC based Pseudo-random function

When non-HMAC based Pseudo-random function start, system will call tunction *Instantiate* to initialize some system parameters. And then pseudo-random number will be produced as follow steps:

1. If *reseed_counter* > 2^{48}, then return an indication that a reseed is required.
2. If *requested_number_of_bits*> 2^{19}, then return an signal that the *requested_number_of_bits* is error.
3. If *additional_inpu*≠Null, then do
   - 3.1 *w* = Hash (0x02 || *V* || *additional_input*).
   - 3.2 *V* = (*V* + *w*) mod $2^{seedlen}$.
4. (*returned_bits*) = Hashgen (*requested_number_of_bits, V*).
5. *H* = Hash (0x03 || *V*).
6. *V* = (*V* + *H* + *C* + *reseed_counter*) mod $2^{seedlen}$.

7. *reseed_counter* = *reseed_counter* + 1.
8. Return *returned_bits*, the new value of *V, C,* and *reseed_counter*

In the steps:

reseed_counter is the number of pseudo-random number had been produced.

additional_input is a string received from the consuming application.

C is seedlen bits that is updated during each call to the Pseudo-random function

V is *seedlen* bits that depends on the *seed.*

requested_number_of_bits is the number of bits of the pseudo-random numberwill be produced.

returned_bits is the produced pseudo-random number.

In the process of produce pseudo-random number, The values of V and C are the critical values. So it must prevent from reveal the values of V and C.

5.3.2.3 Security of non-HMAC based Pseudo-random function

In the non-HMAC based Pseudo-random function, the string C and the 'message' V is protected.

If someone attack the system, he need know the the string C and the 'message' V. The attacker can find the (C,V) that will produce the same Pseudo-random number he has. To do this, he need two successive (V1,V2), then he can calculate out the C, and test (C, V1), if the (C,V1) do not produce the same Pseudo-random number, the attacker had to find other (V1,V2) again. So the attacker must find Preimages of Dynamic SHA at first. The probability of random guess of finding 512(resp.1024)-bit preimages of is 2^{-224} (resp. 2^{-256}, 2^{-384}, 2^{-512}). The bie-length of V is 2^{440} (resp. 2^{888}), even someone has an algorithm to find all messages that have same hash value of Dynamic SHA, he had to find the V from 2^{216} (resp. 2^{184}, 2^{496}, 2^{376}) messages.

6 Security of Dynamic SHA with length extension attack and multicollision attack

6.1 Security of Dynamic SHA with length extension attack

length extension attack can be used to attack keyed-hash function. It make attacker can attacker keyed-hash function without the key.

If there exist keyed-hash function H(K, M), where K is key, M is messahe, and h(hv0. m) is hash function of H(.), and Initial Hash Value of h(hv0. m) is hv0, message of h(hv0. m) is m. The length extension attack is as follow:

Let pad(m) is pad '1' , '0' and the bit-length of message m as section 2.3.1.

If attacker have a pair (hv, M), Then attacker can find collision as follow step:

1. Find a any bit string w,.
2. Constitute new message M'=M||pad(M)||w.
3. Calculate h(H(K,M),w).

If attacker can find the w that make H(K,M)=h(H(K,M),w), he will find a collision that make H(K,M)= H(K,M') without know the key K.

In the attack step, we can find that attacker must find preimages of Dynamic SHA. And the probability of random guess of finding preimages of is 2^{-224} (resp. 2^{-256}, 2^{-384}, 2^{-512}).

6.2 Security of Dynamic SHA with multicollision attack

Joux [19] has developed an algorithm to find a 2^r -way collision for a classical iterated hash function. If the probability of finding collision of a hash function is ε. The probability of finding a 2^r -way collision for the hash function is ε^r.

The probability of finding collision of Dynamic SHA is 2^{-112} (resp. 2^{-128}, 2^{-192}, 2^{-256}). Then the probability of finding 2^r -way collision of Dynamic SHA is $2^{-112\times r}$ (resp. $2^{-128\times r}$, $2^{-192\times r}$, $2^{-256\times r}$). And the complexity of find a 2^r -way collision of Dynamic SHA is $O(r\times 2^{112})$ (resp. $O(r\times 2^{128})$, $O(r\times 2^{192})$, $O(r\times 2^{256})$).

7 Conclusions

Ronald L Rivest[14] had designed RC5, RC5 include data-depend function, it make it hard to analyse RC5. And William Stallings[15] has mentioned that data-depend function will make cipher system has good nonlinear, and composite function of Boolean functions and Arithmetic functions also make cipher system has good nonlinear. Dynamic SHA carry out the two suggestions. It make Dynamic SHA is more nonlinear than SHA-2.

Data-depend function function R divided the message space into many parts, in different part, the calculation is different.

And based on components from the family SHA-2, I have introduced the principle in the design of Dynamic SHA: *When message is changed, different rotate right operation maybe done.* And I bring in data depend function R to realize the principle. The principle enabled us to build a compression function of Dynamic SHA that the iterative part has 48 rounds, it is more robust and resistant against generic multi-block collision attacks, and it is resistant against generic length extension attacks.

References

1. E. Biham and R. Chen, “Near-collisions of SHA-0,” Cryptology ePrint Archive, Report 2004/146, 2004. http://eprint.iacr.org/2004/146

2. B. den Boer, and A. Bosselaers: “An attack on the last two rounds of MD4”, CRYPTO 1991, LNCS, 576, pp. 194-203, 1992.

3. B. den Boer, and A. Bosselaers: “Collisions for the compression function of MD5”, EUROCRYPT 1993, LNCS 765, pp. 293-304, 1994.

4. F. Chabaud and A. Joux, “Differential collisions in SHA-0,” Advances in Cryptology, Crypto98, LNCS, vol.1462, pp.56-71, 1998.

5. H. Dobbertin: “Cryptanalysis of MD4”, J. Cryptology 11, pp. 253-271, 1998.

6. NIST, Secure Hash Signature Standard (SHS) (FIPS PUB 180-2), United States of American, Federal Information Processing Standard (FIPS) 180-2, 2002 August 1.

7. NIST Tentative Timeline for the Development of New Hash Functions,

http://csrc.nist.gov/groups/ST/hash/timeline.html
8. S. Vaudenay, “On the need for multipermutations: Cryptanalysis of MD4 and SAFER”, Fast Software Encryption- FSE95, LNCS 1008, pp. 286–297, 1995.
9. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, “Cryptanalysis of the Hash Functions MD4 and RIPEMD”, EUROCRYPT 2005, LNCS 3494, pp. 1–18, 2005.
10. X. Wang and H. Yu , “How to Break MD5 and Other Hash Functions”, EUROCRYPT 2005, LNCS 3494, pp. 19–35, 2005.
11. X. Wang, H. Yu, Y. L. Yin “Effcient Collision Search Attacks on SHA-0”, CRYPTO 2005, LNCS 3621, pp. 1–16, 2005.
12. X. Wang, Y. L. Yin, H. Yu, “Collision Search Attacks on SHA-1”, CRYPTO 2005, LNCS 3621, pp. 17–36, 2005.
13. Gupta and Sarkar “Computing Walsh Transform from the Algebraic Normal Form of a Boolean Function”
http://citeseer.ist.psu.edu/574240.html
14 Ronald L Rivest “The RC Encryption Algorithm”
http://people.csail.mit.edu/rivest/Rivest-rc5.pdf
15. William Stallings “Cryptography and Network Security Principles and Practices, Third Edition”, ISBN 7-5053-9395-2
16. Xu ZiJie “Dynamic SHA” http://eprint.iacr.org/2007/476
17. Draft NIST SP 800-106 “Randomized Hashing for Digital Signatures”
http://csrc.nist.gov/publications/drafts/800-106/2nd-Draft_SP800-106_July2008.pdf
18. NIST SP 800-90 “Recommendation for Random Number Generation Using Deterministic Random Bit Generators”
http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf
19. A. Joux. Multicollision on Iterated Hash Function. Advances in Cryptology, CRYPTO 2004, Lecture Notes in Computer Science 3152.

[BELL96a] Bellare, M., Canetti, R., and Krawczyk, H. “Keying Hash Functions for MessageAuthentication.” Proceedings, CRYPTO ’96,

August 1996; New York: Springer-Verlag. An expanded version is available at http://www.cse.ucsd. edu/users/mihir.

Appendix 1: Constitute Boolean functions to represent function.

We can use Algebraic Normal Form (ANF) to represent function. Gupta and Sarkar[13] have studied it.

Let n≥r≥1 be integers and let $F:\{0,1\}^n \to \{0,1\}^r$ be a vector valued Boolean function. The vector valued function F can be represented as an r-tuple of Boolean functions $F=(F^{(1)},F^{(2)},...,F^{(r)})$, where $F^{(s)}:\{0,1\}^n \to \{0,1\}(s=1,2,...,r)$, and the value of $F^{(s)}(x_1,x_2,...,x_n)$ equals the value of the s-th component of $F(x_1,x_2,...,x_n)$. The Boolean functions $F^{(s)}(x_1,x_2,...,x_n)$ can be expressed in the Algebraic Normal Form (ANF) as polynomials with n variables $x_1,x_2,...,x_n$ of kind $a_0 \oplus a_1x_1 \oplus ... \oplus a_nx_n \oplus \oplus a_{1,2}x_1x_2 \oplus ... \oplus a_{n-1,n}x_{n-1}x_n \oplus ... \oplus a_{1,2,...,n}x_1,x_2,...,x_n$, where $a_\lambda \in \{0,1\}$. Each ANF has up to 2^n monomials, depending of the values of the coefficients a_λ.

Function R

Function R operates on eight words x1, x2, x3, x4, x5, x6, x7 and x8 and produces a word y as output. So we have $R:\{0,1\}^{8\times w} \to \{0,1\}^w$, It is easy to know that one-bit different in words x1, x2, x3, x4, x5, x6, x7 will make the different rotate right operation be done. So the bit in output maybe changed. And when one-bit different in word x8, the bit in output maybe changed. So the ANFs to represent function R has up to $2^{8\times w}$ monomials, where w is bit length of the word.

Appendix 2: Constitute Arithmetic functions to represent function.

Gupta and Sarkar [13] had studied how to use Algebraic Normal Form (ANF) to represent function. In a similar way, all function will be represented as polynomials.

In appendix 2, the following operations are used:

1. $abs(x)$ is absolute value of x
2. $\lfloor x \rfloor$ is round-off instruction on x
3. "+" is arithmetic addition.
4. "-" is arithmetic subtraction.
5. "$\times$" is arithmetic multiplication.

1. Constitute Arithmetic functions to represent Boolean function:

In Boolean function, 1 is True, 0 is False.

1. To one bit word. The Boolean function can represented with arithmetic functions as follow:

operand	function	arithmetic function
x,y	$z = x \oplus y$	$z = x + y - 2 \times x \times y$
x,y	$z = x \wedge y$	$z = x \times y$
x,y	$z = x \vee y$	$z = x + y - x \times y$
x	$z = \neg x$	$z = 1 - x$

Tables B.1 represent Boolean function with arithmetic function

To Boolean polynomial, it can replace every calculation of polynomial base on table B.1.

2. To n-bit word.

If there are three n-bit words x, y, z. if there exist $z = f(x, y)$ where f is Boolean function that in table B.1.

x, y, z are n-bit words. Let

$$x = \sum_{i=0}^{n-1} x_i \times 2^i$$
$$y = \sum_{i=0}^{n-1} y_i \times 2^i$$
$$z = \sum_{i=0}^{n-1} z_i \times 2^i$$

where x_i, y_i, z_i is i-th bit of word x, y, z. There exists $z_i = f(x_i, y_i)$, where $0 \le i \le n-1$.

To Boolean polynomial, it can replace every calculation base on table B.1 for every bit of variables.

3. If function F includes a series functions $f_0, ..., f_{t-1}$ as follow:

$$z(x, y, k) = \begin{cases} f_0(x, y) & k = 0 \\ ... & \\ f_{t-1}(x, y) & k = t-1 \end{cases}$$

Then it can represent as follow:

$$z(x, y, k) = \sum_{i=0}^{t-1} (2^{abs(i-k)} - \left| \frac{2^{abs(k-i)}}{2} \right| \times 2) \times (f_i(x, y))$$

Base on above-mentioned three ways, it can represent Boolean function with arithmetic functions. And there exists:

Theorem 1. If $x \in \{0,1\}$, there exists $x^k = x \quad k > 0$.

Proof.

If x=0, $x^k = 0^k = 0 = x$

If x=1, $x^k = 1^k = 1 = x$ □

2. Constitute Arithmetic functions to represent function with ANF

Functions $F : \{0,1\}^n \to \{0,1\}^r$ can be expressed in the ANF as polynomials with n variables $x_1, x_2, ..., x_n$ of kind $a_0 \oplus a_1 x_1 \oplus ... \oplus a_n x_n \oplus a_{1,2} x_1 x_2 \oplus ... \oplus a_{n-1,n} x_{n-1} x_n \oplus \oplus a_{1,2,...,n} x_1 .. x_n$, where $a_\lambda \in \{0,1\}$. If replace every calculation in the ANF base on table B.1 and simplified by theorem 1, it can constitute Arithmetic functions to represent ANF. The Arithmetic functions will be polynomials with n variables $x_1, x_2, ... x_n$ of kind $b_0 + b_1 \times x_1 + ... + b_n \times x_n + ... + b_{n-1,n} \times x_{n-1} \times x_n + ... + b_{1,2,...,n} \times x_1 .. \times x_n$, where b_λ are integer. The Arithmetic functions have up to 2^n monomials. The degree of Arithmetic functions is up to n. And there

exists $f=\sum_{i=0}^{r-1}F^{(s)}(x_1,x_2,...,x_n)\times 2^i$, where f is r-bit word.

3. Constitute Arithmetic functions to represent SHR operation:

The shift right operation $SHR^k(x)$ can be represented as follow:

$$y=SHR^k(x)=\left|\frac{x}{2^k}\right| \qquad (2.0)$$

4. Constitute Arithmetic functions to represent data-depend function G:

The function G can be represented as follow:

$$y(x1,x2,x3,t)=\sum_{i=0}^{3}(2^{abs(i-t)}-\left|\frac{2^{abs(t-i)}}{2}\right|\times 2)\times(G_i(x1,x2,x3)) \qquad (2.1)$$

By Theorem 1 and table B.1, function $G_i(x1,x2,x3)$ can be represented as follow:

$$G_t(x1,x2,x3)=\begin{cases}\sum_{i=0}^{w-1}(x1_i+x2_i+x3_i-2\times x1_i\times x2_i-2\times x1_i\times x3_i- & \\ 2\times x2_i\times x3_i+4\times x1_i\times x2_i\times x3_i)\times 2^i & t=0\\ \sum_{i=0}^{w-1}(x1_i\times x2_i+x3_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=1\\ \sum_{i=0}^{w-1}(1-x1_i-x3_i+2\times x1_i\times x3_i+x1_i\times x2_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=2\\ \sum_{i=0}^{w-1}(1-x2_i-x3_i+2\times x2_i\times x3_i+x1_i\times x3_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=3\end{cases} \qquad (2.2)$$

$x1_i,x2_i,x3_i$ is i-th bit of x1, x2, x3. In system (2.2), it is known that G_i are cubic equations, and has 7, 3, 6, 6 monomials. It is easy to know that the equation (2.1) is cubic equation. It is hard to represented equation (2.1) with linear function. And there exists:

$$\frac{d^3(y)}{d(x1_i)d(x2_i)d(x3_i)}=c$$

And c is constant.

5. Constitute Arithmetic functions to represent data-depend function R:

There are two ways to constitute Arithmetic functions to represent data-depend function R:

1. Constitute ANFs that represent function R. And replace the Boolean function base on table B.1. In this way, it will constitute huge Arithmetic function. The ANFs represents function R has up to 2^{256} (resp. 2^{512}) monomials. By theorem 1 and the input has 261(resp. 518) bits, so the degree of the Arithmetic function represents function R is up to 256(resp. 512), and has up to 2^{256} (resp. 2^{512}) monomials. There exiset:

$$\frac{d^{bn}(y)}{d(x_0)....d(x_i)....d(x_{bn-1})} = c$$

where c is constant, x_i is i-th input bit of function R, bn is bit number of input, and bn equal 256(resp. 512).

2. At first, there exist rotate right (circular right shift) operation $ROTR^k(x)$, where x is n-bit word, and $0 \le k < n$. It can represent $y = ROTR^k(x)$ as follow:

$$\begin{aligned} & y = ROTR^k(x) \\ & = \left|\frac{x}{2^k}\right| + (x - \left|\frac{x}{2^k}\right| \times 2^k) \times 2^{n-k} \\ & = x \times 2^{n-k} - \left|\frac{x}{2^k}\right| \times (2^n - 1) \end{aligned} \qquad (2.3)$$

If function $y = ROTR^k(x)$ is not data-depend function, the k in equation (2.3) is constant, and equation (2.3) is linear equation. The derivative function of linear equation is constant. This means the difference of function value depend on the difference of input and the difference of function value dose not depend on the input. In SHA-2, the ROTR operation is not data-depend function, it can constitute linear equation to represent the ROTR operation in SHA2.

If function $y = ROTR^k(x)$ is data-depend function, the k in equation (2.3) is variable, and equation (2.3) is exponential function. And equation (2.3) will be exponential function with round-off instruction. It is hard to represent exponential function with linear equation. The derivative function of exponential function is exponential function. This means the difference of function value depend the difference of input and input. When the input changes, the different of function value maybe change. In Dynamic SHA, function R is data-depend function. And if use equation

(2.3) represents function R, the k is function of working variables a,b,c, d, e, f, g, and $k = K(a,b,c,d,e,f,g,h)$ as table B.2, the equation (2.2) will be complex exponential function. After several rounds, equation (2.3) will be iteration function with equation (2.3), it will be very huge and complex, and there exists no mathematical theory that reduces the size of equation (2.3). It is hard to analyses Dynamic SHA that includes function R.

Dynamic SHA-224/256	$t0 = (((((a + b) \oplus c) + d) \oplus e) + f) \oplus g$ $t1 = (SHR^{17}(t0) \oplus t0) \wedge (2^{17} - 1)$ $t2 = (SHR^{10}(t1) \oplus t1) \wedge (2^{10} - 1)$ $k = (SHR^{5}(t2) \oplus t2) \wedge 31$
Dynamic SHA-384/512	$t0 = (((((a + b) \oplus c) + d) \oplus e) + f) \oplus g$ $t1 = (SHR^{36}(t0) \oplus t0) \wedge (2^{36} - 1)$ $t2 = (SHR^{18}(t1) \oplus t1) \wedge (2^{18} - 1)$ $t3 = (SHR^{12}(t2) \oplus t2) \wedge (2^{12} - 1)$ $k = (SHR^{6}(t3) \oplus t3) \wedge 63$

Table B.2. function K for Dynamic SHA

Compare the Arithmetic function that represent SHA2, The Arithmetic function that represent functions in Dynamic SHA include exponential function. Or the Arithmetic function that represents functions in Dynamic SHA has higher degree than the Arithmetic function that represents functions in SHA2. This make it is harder to analyses Dynamic SHA.

Appendix 3: Function G and Function R.

Let $p(x)$ is probability of x.

Function G

Function G operates on three words x1,x2, x3 and an integer t, produces a word y as output. t is constant, x1,x2, x3 are w-bit words and $0 \le t < w$. And function G as follow:

$$y = G_t(x1, x2, x3) = \begin{cases} x1 \oplus x2 \oplus x3 & t = 0 \\ (x1 \wedge x2) \oplus x3 & t = 1 \\ (\neg(x1 \vee x3)) \vee (x1 \wedge (x2 \oplus x3)) & t = 2 \\ (\neg(x1 \vee (x2 \oplus x3))) \vee (x1 \wedge \neg x3) & t = 3 \end{cases}$$

If x1, x2, x3 are random and uncorrelated. There is:

$$p(y) = \sum_{i1=0}^{2^w-1} \sum_{i2=0}^{2^w-1} \sum_{i3=0}^{2^w-1} p(y \mid (x1_{i1}, x2_{i2}, x3_{i3})) \times p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3})$$

$$p(y) = p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3}) \times \sum_{i1=0}^{2^w-1} \sum_{i2=0}^{2^w-1} \sum_{i3=0}^{2^w-1} p(y \mid (x1_{i1}, x2_{i2}, x3_{i3}))$$

To a given y', there are $2^{2\times w}$ 2-tuple (x1,x2), there is relation:

$$x3 = \begin{cases} x1 \oplus x2 \oplus y' & t = 0 \\ (x1 \wedge x2) \oplus y' & t = 1 \\ (\neg(x1 \vee y')) \vee (x1 \wedge (x2 \oplus y')) & t = 2 \\ (\neg(x1 \vee (x2 \oplus y'))) \vee (x1 \wedge \neg y') & t = 3 \end{cases}$$

it can compute the value for x3. x1, x2, x3 are random and uncorrelated variable, there is

$$p(x1) = p(x2) = p(x3) = 2^{-w}$$

Then there is:

$$p(y) = p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3}) \times \sum_{i1=0}^{2^w-1} \sum_{i2=0}^{2^w-1} \sum_{i3=0}^{2^w-1} p(y \mid (x1_{i1}, x2_{i2}, x3_{i3})) = 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{2\times w} = 2^{-w}$$

If x1, x2, x3 are random and uncorrelated, function G will produce random word and $p(y) = 2^{-w}$.

Function R

Function y=R(x1,x2,x3,x4,x5,x6,x7,x8) operates on eight words x1,x2,x3,x4,x5,x6,x7 and x8, produce a word as output as table 2.3. x1, x2, x3, x4, x5, x6, x7, x8 are w-bit words. If x1, x2, x3, x4, x5, x6, x7, x8 are random and uncorrelated.

There exists:

$$p(t0) = \sum_{i1=0}^{2^w-1}\sum_{i2=0}^{2^w-1}\cdots\sum_{i7=0}^{2^w-1} p(t0 \mid (x1_{i1},..., x7_{i7})) \times p(x1_{i1}) \times ... \times p(x7_{i7})$$

$$p(t0) = p(x1_{i1}) \times ... \times p(x7_{i7}) \times \sum_{i1=0}^{2^w-1}\sum_{i2=0}^{2^w-1}\cdots\sum_{i7=0}^{2^w-1} p(t0 \mid (x1_{i1},..., x7_{i7}))$$

To given value t0', There is $2^{7\times w}$ 7-tuple (x1',x2',x3',x4',x5',x6',x7'), There is relation: $x7 = (((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus t0$. it can compute the value for x7. and x1, x2, x3, x4, x5, x6, x7 are random and uncorrelated variable. There exists:

$$p(x1_{i1}) = p(x2_{i2}) = p(x3_{i3}) = p(x4_{i4}) = p(x5_{i5}) = p(x6_{i6}) = p(x7_{i7}) = 2^{-w}$$

$$p(t0') = p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3}) \times p(x4_{i4}) \times p(x5_{i5}) \times p(x6_{i6}) \times p(x7_{i7}) \times 2^{6\times w} = 2^{-w}.$$

t0 is w-bit word, let t is $\log_2^w$-bit word, let: t0=($t0_0,...,t0_{w-1}$) and t=($t_0,...,t_{\log_2^w-1}$), $t0_i, t_i$ is i-th bit of t0 and t, and there is

Dynamic SHA-224/256	$\begin{cases} t_0 = t0_0 \oplus t0_5 \oplus t0_{10} \oplus t0_{17} \oplus t0_{22} \oplus t0_{27} \oplus t0_{15} \\ t_1 = t0_1 \oplus t0_6 \oplus t0_{11} \oplus t0_{18} \oplus t0_{23} \oplus t0_{28} \oplus t0_{16} \\ t_2 = t0_2 \oplus t0_7 \oplus t0_{12} \oplus t0_{19} \oplus t0_{24} \oplus t0_{29} \\ t_3 = t0_3 \oplus t0_8 \oplus t0_{13} \oplus t0_{20} \oplus t0_{25} \oplus t0_{30} \\ t_4 = t0_4 \oplus t0_9 \oplus t0_{14} \oplus t0_{21} \oplus t0_{26} \oplus t0_{31} \end{cases}$
Dynamic SHA-384/512	$\begin{cases} t_0 = t0_0 \oplus t0_6 \oplus t0_{12} \oplus t0_{18} \oplus t0_{24} \oplus t0_{30} \oplus t0_{36} \oplus t0_{42} \oplus t0_{48} \oplus t0_{54} \oplus t0_{60} \\ t_1 = t0_1 \oplus t0_7 \oplus t0_{13} \oplus t0_{19} \oplus t0_{25} \oplus t0_{31} \oplus t0_{37} \oplus t0_{43} \oplus t0_{49} \oplus t0_{55} \oplus t0_{61} \\ t_2 = t0_2 \oplus t0_8 \oplus t0_{14} \oplus t0_{20} \oplus t0_{26} \oplus t0_{32} \oplus t0_{38} \oplus t0_{44} \oplus t0_{50} \oplus t0_{56} \oplus t0_{62} \\ t_3 = t0_3 \oplus t0_9 \oplus t0_{15} \oplus t0_{21} \oplus t0_{27} \oplus t0_{33} \oplus t0_{39} \oplus t0_{45} \oplus t0_{51} \oplus t0_{57} \oplus t0_{63} \\ t_4 = t0_4 \oplus t0_{10} \oplus t0_{16} \oplus t0_{22} \oplus t0_{28} \oplus t0_{34} \oplus t0_{40} \oplus t0_{46} \oplus t0_{52} \oplus t0_{58} \\ t_5 = t0_5 \oplus t0_{11} \oplus t0_{17} \oplus t0_{23} \oplus t0_{29} \oplus t0_{35} \oplus t0_{41} \oplus t0_{47} \oplus t0_{53} \oplus t0_{59} \end{cases}$

And there is relation:

Dynamic SHA-224/256	$\begin{cases} t0_0 = t_0 \oplus t0_5 \oplus t0_{10} \oplus t0_{17} \oplus t0_{22} \oplus t0_{27} \oplus t0_{15} \\ t0_1 = t_1 \oplus t0_6 \oplus t0_{11} \oplus t0_{18} \oplus t0_{23} \oplus t0_{28} \oplus t0_{16} \\ t0_2 = t_2 \oplus t0_7 \oplus t0_{12} \oplus t0_{19} \oplus t0_{24} \oplus t0_{29} \\ t0_3 = t_3 \oplus t0_8 \oplus t0_{13} \oplus t0_{20} \oplus t0_{25} \oplus t0_{30} \\ t0_4 = t_4 \oplus t0_9 \oplus t0_{14} \oplus t0_{21} \oplus t0_{26} \oplus t0_{31} \end{cases}$
Dynamic SHA-384/512	$\begin{cases} t0_0 = t_0 \oplus t0_6 \oplus t0_{12} \oplus t0_{18} \oplus t0_{24} \oplus t0_{30} \oplus t0_{36} \oplus t0_{42} \oplus t0_{48} \oplus t0_{54} \oplus t0_{60} \\ t0_1 = t_1 \oplus t0_7 \oplus t0_{13} \oplus t0_{19} \oplus t0_{25} \oplus t0_{31} \oplus t0_{37} \oplus t0_{43} \oplus t0_{49} \oplus t0_{55} \oplus t0_{61} \\ t0_2 = t_2 \oplus t0_8 \oplus t0_{14} \oplus t0_{20} \oplus t0_{26} \oplus t0_{32} \oplus t0_{38} \oplus t0_{44} \oplus t0_{50} \oplus t0_{56} \oplus t0_{62} \\ t0_3 = t_3 \oplus t0_9 \oplus t0_{15} \oplus t0_{21} \oplus t0_{27} \oplus t0_{33} \oplus t0_{39} \oplus t0_{45} \oplus t0_{51} \oplus t0_{57} \oplus t0_{63} \\ t0_4 = t_4 \oplus t0_{10} \oplus t0_{16} \oplus t0_{22} \oplus t0_{28} \oplus t0_{34} \oplus t0_{40} \oplus t0_{46} \oplus t0_{52} \oplus t0_{58} \\ t0_5 = t_5 \oplus t0_{11} \oplus t0_{17} \oplus t0_{23} \oplus t0_{29} \oplus t0_{35} \oplus t0_{41} \oplus t0_{47} \oplus t0_{53} \oplus t0_{59} \end{cases}$

To a given $\log_2^w$ – tuple t'=(t'_0 ,..., $t'_{\log_2^w - 1}$), there is $2^{w-\log_2^w}$ $(w-\log_2^w)-\text{tuple}(t0_{\log_2^w},...,t0_{w-1})$. To a given $(w-\log_2^w)-tuple$ $(t0'_{\log_2^w},...,t0'_{w-1})$, it can compute the $\log_2^w$ – tuple for $(t0_0,...,t0_{\log_2^w-1})$. And there is

$$p(t') = \sum_{i=0}^{2^w-1} p(t' | t0(i)) \times p(t0) = 2^{w-\log_2^w} \times p(t0) = 2^{w-\log_2^w} \times 2^{-w} = 2^{-\log_2^w} = w^{-1}.$$

x1, x2, x3, x4, x5, x6, x7, x8 are random and uncorrelated words, and t is produced from x1, x2, x3, x4, x5, x6, x7. To $y = ROTR^t(x8)$, there is relation $x8 = ROTR^{w-t}(y)$. To a given value y', there are w value t, to a given t', it can compute the value for x8. And there is:

$$p(y) = \sum_{i=0}^{w-1} \sum_{i8=0}^{2^w-1} p(y | (t_i, x8_{i8})) \times p(t_i) \times p(x8_{i8}) = w \times w^{-1} \times 2^{-w} = 2^{-w}$$

If x1, x2, x3, x4, x5, x6, x7, x8 are random and uncorrelated words, function R will produce random word and $p(y) = 2^{-w}$.

In function R, one bit difference in x1, x2, x3, x4, x5, x6, x7, different ROTR will be done, this will make every bit in x8 maybe changed.

Appendix 4: Some thing about Dynamic SHA

1. Why Dynamic SHA use function G and function R

As mentioned at appendix 3, if the variables are random word, function G and R will produced random word.

Function G operates on three words, produces a word as output. What function G does is produce confusion word.

Function R operates on eight words, produces a word as output. What function R do is confuse the place of bit.

The reason that Dynamic SHA use function G and function R is:

1. To get better property of spreading, Dynamic SHA use function G and R.
2. With function G, function R makes it hard to find out the relationship between message change and bit place change. If function G is data-depend function, function G and R will make it hard to find out the relationship between message value and what logical function is called and where the bit is placed. And it is hard to analyze data-depend function with differential analysis. It needs construction Arithmetic function to describe function, the degree of the Arithmetic function is up to 256(resp. 512). Or construction exponential function to describe function R. And the ANFs that describe function R has up to 2^{256} (resp. 2^{512}) monomials.

2. Why there are 48 rounds in Dynamic SHA

The reason that there are 48 rounds in Dynamic SHA is as follow:

1. It is easy to know mix message words too many times will reduce the randomness of variable of function G and R. So the message bits are mixed no many times in Dynamic SHA.

2. It is easy to backward function R and G, so it must repeatedly mix message words. When someone backward iterative steps by guessing message word, then he will has one system include equations like (1), the system has 48 equations with 16 unknown variables.

So message words will be mixed three times in Dynamic SHA, and in one round, one message word will be mixed. So there are 48 rounds in

Dynamic SHA.

3. Why there is no message expansion part in Dynamic SHA

Mixing message words many times will reduce the randomness of variable of function G and R. and the degree of Arithmetic function that describe function R is up to 256(resp.512), and there are up to 2^{256} (resp. 2^{512}) monomials in the Arithmetic function. It is enough now.

4. It is hard to analyses Dynamic SHA

Dynamic SHA include function R. To analyses Dynamic SHA, it need unchangeable representation of function R. As mentioned in Appendix 1 and Appendix 2, the ANFs that describe function R has up to 2^{160} ,(resp. 2^{320}) monomials, and the Arithmetic functions that describe function R is exponential function with round-off instruction like equation (2.3) or arithmetic function that the degree is up to 256 (resp. 512).

5. Avalanche of Dynamic SHA.

From the definition of function R, it is enough to known that all bits in working variables a,b,c,d,e,f,g will affect all bits in temporary words T.

After 16 rounds, all message bits are mixed. There are 32 rounds after all message bits are mixed. And after call function R 9 times, all bits in working variables that before 17-th round will affect all bits in working variables after 26-th round. Some bits in message will not affect all bits in last hash value. So all message bits will affect all bits in last hash value.

Appendix 5: Spreading of Dynamic SHA

Let:

1. hv(i)=(a(i), b(i), c(i), d(i),e(i), f(i), g(i), h(i)). Where a(i), b(i), c(i), d(i),e(i), f(i), g(i), h(i) are working variables at i-th round.

2.MW1=(W(0),W(1),W(2),W(3),W(4),W(5),W(6),W(7)),
MW2=(W(8),W(9),W(10),W(11),W(12),W(13),W(14),W(15))
W(j) is the message word.

3. $H_i(MW1,\ MW2) = hv(i) \qquad 15 \le i$.

Divide the iterative part into some parts, and a part includes 8 rounds. The iterative part will be as table E.1. hv(-1) is initiation of eight working variables

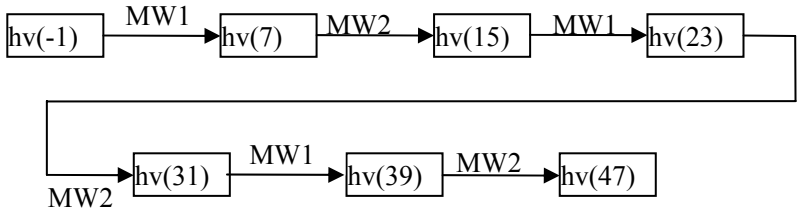

Table E.1. Iterative part of Dynamic SHA

And there is:

$$H_{15}(MW1,MW2) = hv(15)$$
$$H_{23}(MW1,MW2) = hv(23)$$
$$H_{31}(MW1,MW2) = hv(31)$$
$$H_{39}(MW1,MW2) = hv(39)$$
$$H_{47}(MW1,MW2) = hv(47)$$

At first there are two theorems:

Theorem 2. *Let:*

1. hv(i)=(a(i), b(i), c(i), d(i),e(i), f(i), g(i), h(i)). Where a(i), b(i), c(i), d(i),e(i), f(i), g(i), h(i) are working variables at i-th round. working variables are b-bit word.

2.MW=(W(i),W(i+1),W(i+2),W(i+3),W(i+4),W(i+5),W(i+6), W(i+7)), where W(j) is the word mixed in j-th round.

3. H(hv(i-1),MW)=hv(i+7)
hv(i), MW are random and uncorrelated.

Then there are:

(1),p(hv(i+7))$= 2^{-8\times b}$

(2),p(hv(i+7)|MW)$= 2^{-8\times b}$

(3),p(hv(i+7)| hv(i-1))$= 2^{-8\times b}$

Proof.

(1),(2)

Before i-th round, we have (hv(i-1),MW), after 8 rounds, MW is mixed, then we have hv(i+7), so we have $H : \{0,1\}^{16\times b} \to \{0,1\}^{8\times b}$.

There are $2^{8\times b}$ MW. To given MW', there is $H_{MW'}(\mathrm{hv(i-1)}) = \mathrm{hv(i+7)}$ and $H_{MW'} : \{0,1\}^{8\times b} \to \{0,1\}^{8\times b}$:

To given hv(i-1)', there is relation $H_{MW'}(\mathrm{hv(i-1)'}) = \mathrm{hv(i+7)}$, it can compute the value for hv(i+7).

To given hv(i+7)', it is easy to backward iterative steps as table 3.2 show. Then there is relation $H_{MW'}'(\mathrm{hv(i+7)}) = \mathrm{hv(i-1)}$, it can compute the value for hv(i).

hv(i), MW are random and uncorrelated, there is $p(hv(i-1)) = 2^{-8\times b}$, $p(MW) = 2^{-8\times b}$. So there is:

$$p(hv(i+7)) = \sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i+7)\,|\,(hv(i-1)_{i1}, MW(i2))) \times p(hv(i-1))_{i1} \times p(MW(i2))$$

$$p(hv(i+7)) = p(hv(i-1)) \times p(MW) \times \sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i+7)\,|\,(hv(i-1)_{i1}, MW(i2)))$$

$$p(hv(i+7)) = 2^{-8\times b} \times 2^{-8\times b} \times 2^{8\times b} = 2^{-8\times b}$$

$$p(hv(i+7)\,|\,MW) = \sum_{i1=0}^{2^b-1} p((hv(i+7)\,|\,MW)\,|\,hv(i-1)_{i1}) \times p(hv(i-1)_{i1}) = 2^{-8\times b} = 2^{-8\times b}$$

(3)

To given hv(i-1)':

To a given hv(i+7)', it is easy to backward iterative steps as table 3.2 show. We will have a system of 8 equations with 8 unknown variables. It is easy to compute the value for MW.

To a given MW', it is easy to compute the value hv(i+7).

MW are random, there exist $p(MW) = 2^{-8\times b}$

So there exist:

$$p(hv(i+7)\mid hv(i-1)) = \sum_{i1=0}^{2^b-1} p((hv(i+7)\mid hv(i-1))\mid MW(i1))\times p(MW(i1))$$
$$= 2^{-8\times b} = 2^{-8\times b}$$

□

By theorem 2, it is easy to know that:

In 8 round, to a given hv(i-1)', mix different message words MW, the hv(i+7) will be different.

In 8 round, mix given message words MW', if the hv(i-1) is different, the hv(i+7) will be different.

The relationship is as Table E.2, $hv(i-1)_j$ is j-th value in hv(i-1) space. $hv(i+7)_j$ is j-th value in hv(i+7) space. MW_j is j-th value in MW space.

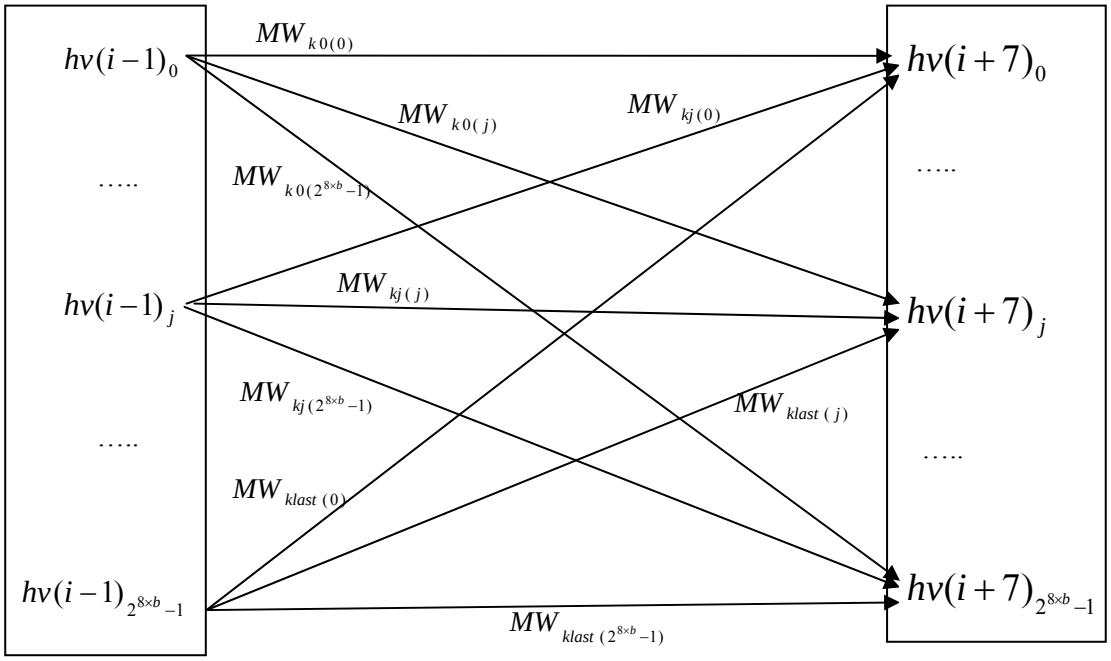

Table E.2 relationship between hv(i-1), hv(i+7), MW

Theorem 3. *In Dynamic SHA, there exist:*

(1) $p(hv(j)) = 2^{-8\times b}$

(2),$p(hv(j)|MW1) = 2^{-8\times b}$

(3),$p(hv(j)|MW2) = 2^{-8\times b}$

$j = 15 + 8\times k \quad k = 0,1,2,.....$

Proof.

hv(-1), MW1 and MW2 are random and uncorrelated, so there exist:

p(hv(-1)) = $2^{-8\times b}$

p(MW1) = $2^{-8\times b}$

p(MW2) = $2^{-8\times b}$.

To simplification, Let F(hv(i-8),MW)=hv(i), MW is mixed words MW1 or MW2.

To a given hv(i)' $i=15+8\times k \quad k=0,1,2,.....$, there are $2^{16\times b}$ 2-tuple(MW1,MW2).

To a given 2-tuple(hv(i)',MW1'), there are $2^{8\times b}$ MW2. To a given 2-tuple (hv(i)',MW2'), there are $2^{8\times b}$ MW1.

To a given 3-tuple(hv(i)',MW1',MW2'), It is easy to backward iterative steps, and it is easy to compute the value for hv(-1), and the hv(-1) make $H_i(\text{hv(-1)},\ \text{MW1}'\ ,\text{MW2}'\ ) = \text{hv(i)}'$.

So there exist:

$$p(hv(i)) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i)\,|\,(hv(-1)_{i0}, MW1_{i1}, MW2_{i2}))\times p(hv(-1)_{i0}, MW1_{i1}, MW2_{i2})$$

$$p(hv(i)) = p(hv(-1), MW1, MW2)\times\sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i)\,|\,(hv(-1)_{i0}, MW1_{i1}, MW2_{i2}))$$

$$p(hv(i)) = 2^{-8\times b}\times 2^{-8\times b}\times 2^{-8\times b}\times 2^{8\times b}\times 2^{8\times b} = 2^{-8\times b}$$

$$p(hv(i)\,|\,MW1) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i)\,|\,MW1)\,|\,(hv(-1)_{i0}, MW2_{i1}))\times p(hv(-1)_{i0}, MW2_{i1})$$

$$p(hv(i)\,|\,MW1) = p(hv(-1), MW2)\times\sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i)\,|\,MW1)\,|\,(hv(-1)_{i0}, MW2_{i1}))$$

$$p(hv(i)\,|\,MW1) = 2^{-8\times b}\times 2^{-8\times b}\times 2^{8\times b} = 2^{-8\times b}$$

$$p(hv(i)\,|\,MW2) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i)\,|\,MW2)\,|\,(hv(-1)_{i0}, MW1_{i1}))\times p(hv(-1)_{i0}, MW1_{i1})$$

$$p(hv(i)\,|\,MW2) = p(hv(-1), MW1)\times\sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i)\,|\,MW2)\,|\,(hv(-1)_{i0}, MW1_{i1}))$$

$$p(hv(i)\,|\,MW2) = 2^{-8\times b}\times 2^{-8\times b}\times 2^{8\times b} = 2^{-8\times b}$$

□

Theorem 4. *In Dynamic SHA, to a given hv(-1), there exist:*

(1) p(hv(23)| hv(-1))= $2^{-8\times b}$

Proof. To simplification, Let F(hv(i-8),MW)=hv(i), MW is mixed words MW1 or MW2.

To a given 2-tuple (hv(23)',hv(-1)'), there are $2^{8\times b}$ MW1.

To a given 2-tuple (hv(23)', MW1'), by theorem 2, there is p(hv(23)| MW1) = $2^{-8\times b}$, so to a given 2-tuple(hv(23)', MW1'), there is $2^{-8\times b}\times 2^{8\times b}=1$ hv(15) that make F(hv(15),MW1')= hv(23)'.

To a given 2-tuple (hv(-1)', MW1'), from the definition of iterative steps, it is enough to know that there is a hv(7) that make F(hv(-1)',MW1')= hv(7).

To a given 2-tuple (hv(7)', hv(15)'), by theorem 2, there is p(hv(15)| hv(7)) = $2^{-8\times b}$, so to a given 2-tuple(hv(7)', hv(15)'), there is $2^{-8\times b}\times 2^{8\times b}=1$ MW2 that make F(hv(7)', MW2) = hv(15)' .

So there exist:

$$p(hv(23)\,|\,hv(-1)) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(23)\,|\,hv(-1))\,|\,(MW1_{i0}, MW2_{i1}))\times p(MW1_{i0}, MW2_{i1})$$

$$p(hv(23)\,|\,hv(-1)) = p(MW1, MW2)\times\sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(23)\,|\,hv(-1))\,|\,(MW1_{i0}, MW2_{i1}))$$

$$p(hv(23)\,|\,hv(-1)) = 2^{-8\times b}\times 2^{-8\times b}\times 2^{8\times b} = 2^{-8\times b}$$

□

By theorem 3 and 4, it is enough to know that:

1. When hv(-1) is random variable, the probability of hash value is $2^{-8\times b}$.
2. To a given hv(-1), the probability of different hash value maybe different.

After 23-th round, the message has been mixed, the mixed message words and working variables value are not uncorrelated, it is hard to analyze the probability of hash value. To get better property of spreading, Dynamic SHA adopts ways as follow:

1. Function G and R are used in Dynamic SHA. When the variables of function G, R are random and uncorrelated, function G, R will produce random value.