Dynamic SHA2

Zijie Xu
E-mail: Hxuzijiewz@gmail.comH

Abstract. In this paper I describe the construction of Dynamic SHA2 family of cryptographic hash functions. They are built with design components from the SHA-2 family, but I use the bits in message as parameters of function G, R and ROTR operation in the new hash function. It enabled us to achieve a novel design principle: *When message is changed, the calculation will be different.* It makes the system can resistant against all extant attacks. Dynamic SHA2 is posted[16]

Key words: Cryptographic hash function, SHA, Dynamic SHA2

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, Dynamic

SHA2 use bits in message as parameter of function G, R and ROTR operation 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 SHA2. The principle is:

When message is changed, the calculation will be different.

The principle combined with the already robust design principles present in SHA-2 enabled us to build a compression function of Dynamic SHA2 that has the following properties:

1. There is not message expansion part.
2. The iterative part includes three parts.
3. The first part includes one round. Mix message words once.
4. The second part includes 9 rounds. Mix no message word.
5. The third part includes 7 rounds. Mix message words 7 times.

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 SHA2:

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 $0 \le n < 32$ (resp. $0 \le n < 64$).
4.The shift left operation, $SHL^{n}(x)$, where x is a 32-bit or 64-bit word and n is an integer with $0 \le n < 32$ (resp. $0 \le n < 64$).
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 SHA2, and sometimes as Dynamic SHA2-224/256 or Dynamic SHA2-384/512.

2.1 Functions

Dynamic SHA2 includes four functions. The 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 SHA2

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

Function R operates on eight words x1, x2, x3, x4, x5, x6, x7, x8 and an integer t. produces one word y as output. Function R as follow:

$$y = ROTR^t(((((((x1 \oplus x2) + x3) \oplus x4) + x5) \oplus x6) + x7) \oplus x8)$$

2.1.3 Function R1(x1,x2,x3,x4,x5,x6,x7,x8)

Function R1 operates on eight words x1, x2, x3, x4, x5, x6, x7, x8. produces one word y as output. Function R1 as table 2.3 show:

2.1.4 Function COMP(hv1,hv2, …,hv8,w(0),w(1),…,w(7),t)

Function COMP operates on sixteen words hv1,hv2, …,hv8, w(0), w(1),…,w(7) and an integer t. Function COMP is defined as table 2.4.

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

Dynamic SHA2-224/256	$t0 = (((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7$ $t1 = (SHR^{17}(t0) \oplus t0) \wedge (2^{17} - 1)$ $t2 = (SHR^{10}(t1) \oplus t1) \wedge (2^{10} - 1)$ $t = (SHR^{5}(t2) \oplus t2) \wedge 31$ $y = ROTR^{t}(x8)$
Dynamic SHA2-384/512	$t0 = (((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7$ $t1 = (SHR^{36}(t0) \oplus t0) \wedge (2^{36} - 1)$ $t2 = (SHR^{18}(t1) \oplus t1) \wedge (2^{18} - 1)$ $t3 = (SHR^{12}(t2) \oplus t2) \wedge (2^{12} - 1)$ $t = (SHR^{6}(t3) \oplus t3) \wedge 63$ $y = ROTR^{t}(x8)$

Table 2.3. function R1 for Dynamic SHA2

2.2 Dynamic SHA2 Constants

Dynamic SHA2 does not use any constants.

2.3 Preprocessing

Preprocessing in Dynamic SHA2 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 SHA2. 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 $M_0^{(i)}, M_1^{(i)}, \ldots\ldots, M_{15}^{(i)}$.

Dynamic SHA2 may be used to hash a message, M, having a length of l

Dynamic SHA2-224/256	$T = R(hv1, hv2, hv3, hv4, hv5, hv6, hv7, hv8, w(t) \wedge 31)$ $hv8 = hv7$ $hv7 = ROTR^{(SHR^{5}(w(t))) \wedge 31}(hv6)$ $hv6 = hv5 + w((t+3) \wedge 7)$ $hv5 = ROTR^{(SHR^{10}(w(t))) \wedge 31}(hv4)$ $hv4 = G(hv1, hv2, hv3, SHR^{30}(w(t))) + w((t+2) \wedge 7)$ $hv3 = hv2$ $hv2 = hv1$ $hv1 = T + w((t+1) \wedge 7)$ $T = R(hv1, hv2, hv3, hv4, hv5, hv6, hv7, hv8, (SHR^{15}(w(t))) \wedge 31)$ $hv8 = hv7 + w((t+7) \wedge 7)$ $hv7 = ROTR^{(SHR^{20} w(t)) \wedge 31}(hv6)$ $hv6 = hv5 + w((t+6) \wedge 7)$ $hv5 = ROTR^{(SHR^{25} w(t)) \wedge 31}(hv4)$ $hv4 = G(hv1, hv2, hv3, t \wedge 3) + w((t+5) \wedge 7)$ $hv3 = hv2 + w(t)$ $hv2 = hv1$ $hv1 = T + w((t+4) \wedge 7)$
Dynamic SHA2-384/512	$T = R(hv1, hv2, hv3, hv4, hv5, hv6, hv7, hv8, w(t) \wedge 63)$ $hv8 = hv7$ $hv7 = ROTR^{(SHR^{6}(w(t))) \wedge 63}(hv6)$ $hv6 = ROTR^{(SHR^{12}(w(t))) \wedge 63}(hv5) + w((t+3) \wedge 7)$ $hv5 = ROTR^{(SHR^{18}(w(t))) \wedge 63}(hv4)$ $hv4 = G(hv1, hv2, hv3, SHR^{62}(w(t))) + w((t+2) \wedge 7)$ $hv3 = ROTR^{(SHR^{24}(w(t))) \wedge 63}(hv2)$ $hv2 = hv1$ $hv1 = T + w((t+1) \wedge 7)$ $T = R(hv1, hv2, hv3, hv4, hv5, hv6, hv7, hv8, (SHR^{30}(w(t))) \wedge 63)$ $hv8 = hv7 + w((t+7) \wedge 7)$ $hv7 = ROTR^{(SHR^{36}(w(t))) \wedge 63}(hv6)$ $hv6 = ROTR^{(SHR^{42}(w(t))) \wedge 63}(hv5) + w((t+6) \wedge 7)$ $hv5 = ROTR^{(SHR^{48}(w(t))) \wedge 63}(hv4)$ $hv4 = G(hv1, hv2, hv3, (SHR^{60}(w(t))) \wedge 3) + w((t+5) \wedge 7)$ $hv3 = hv2 + w(t)$ $hv2 = ROTR^{(SHR^{54}(w(t))) \wedge 63}(hv1)$ $hv1 = T + w((t+4) \wedge 7)$

Table 2.4 function COMP for Dynamic SHA2

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.

Dynamic SHA2-224	Dynamic SHA2-256	Dynamic SHA2-384	Dynamic SHA2-512
$H_0^{(0)} = c1059ed8$, $H_1^{(0)} = 367cd507$, $H_2^{(0)} = 3070dd17$, $H_3^{(0)} = f70e5939$, $H_4^{(0)} = ffc00b31$, $H_5^{(0)} = 68581511$, $H_6^{(0)} = 64f98fa7$, $H_7^{(0)} = befa4fa4$,	$H_0^{(0)} =$ 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 2.5 The initial hash value, H^0 for Dynamic SHA

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 2.5).

```
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:
        $a = H_0^{(i-1)}$,  $b = H_1^{(i-1)}$,  $c = H_2^{(i-1)}$,  $d = H_3^{(i-1)}$,
        $e = H_4^{(i-1)}$,  $f = H_5^{(i-1)}$,  $g = H_6^{(i-1)}$,  $h = H_7^{(i-1)}$
2. Iterative part
2.1 The first iterative part
    $COMP(a,b,c,d,e,f,g,h,w_0,w_1,w_2,w_3,w_4,w_5,w_6,w_7,0)$
    $COMP(a,b,c,d,e,f,g,h,w_8,w_9,w_{10},w_{11},w_{12},w_{13},w_{14},w_{15},0)$
2.2 The second iterative part
For t=0 to 8
{
    $T = R1(a,b,c,d,e,f,g,h)$
    $h = g$
    $g = f$
    $f = e$
    $e = d$
    $d = c$
    $c = b$
    $b = a$
    $a = T$
}
2.3 The third iterative part
For t=1 to 7
{
    $COMP(a,b,c,d,e,f,g,h,w_0,w_1,w_2,w_3,w_4,w_5,w_6,w_7,t)$
    $COMP(a,b,c,d,e,f,g,h,w_8,w_9,w_{10},w_{11},w_{12},w_{13},w_{14},w_{15},t)$
}

3.Compute the $i^{th}$ intermediate hash value $H^{(i)}$ :
    $H_0^{(i)} = a + H_0^{(i-1)}$,  $H_1^{(i)} = b + H_1^{(i-1)}$,  $H_2^{(i)} = c + H_2^{(i-1)}$,  $H_3^{(i)} = d + H_3^{(i-1)}$,
    $H_4^{(i)} = e + H_4^{(i-1)}$,  $H_5^{(i)} = f + H_5^{(i-1)}$,  $H_6^{(i)} = g + H_6^{(i-1)}$,  $H_7^{(i)} = h + H_7^{(i-1)}$
}
```

Table 2.6 Algorithmic description of Dynamic SHA2 hash function.

2.5 Dynamic SHA2 Hash Computation

The Dynamic SHA2 hash computation uses functions and initial values defined in previous subsections. So, after the preprocessing is completed, each message block, $M^{(0)}, M^{(1)}, \ldots\ldots, M^{(N)}$, is processed in order, using the steps described algorithmically in Table 2.6.

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 SHA2-256 is a 256-bit message digest and of Dynamic SHA2-512 is a 512-bit message digest. The final result of Dynamic SHA2-224 and Dynamic SHA2-384 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 $W_0, W_1, \ldots, W_{15}$. 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 $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$, 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 SHA2 also uses one temporary words T.

3 Security of Dynamic SHA2

In this section I will make an initial analysis of how strongly collision resistant, preimage resistant and second preimage resistant Dynamic SHA2 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

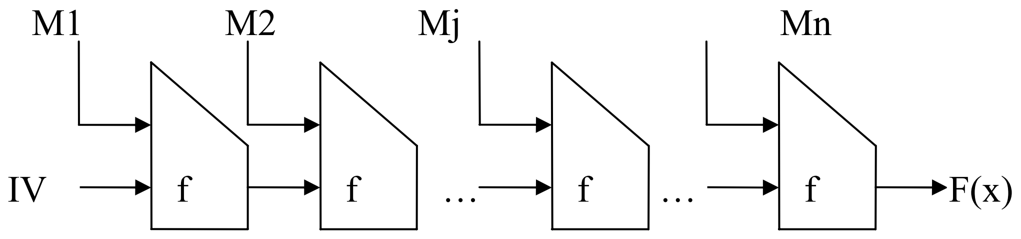

Table 3.1 The iterated construction of compression function f

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 SHA2 in one block.

3.1 Properties of iterative part

The iterative part includes three parts.

3.1.1 Properties of iterative part one

In iterative part one, all message bits have been mixed. And function COMP is called twice. All bits in message words W_0,W_8 have been used as parameters of function G, R and ROTR operation.

3.1.2 Properties of iterative part two

It is relatively easy to prove the following Theorem:

Theorem 1: The iterative part two of Dynamic SHA2 is a bijection $\xi:\{0,1\}^{8\times w}\rightarrow\{0,1\}^{8\times w}$. working variables are w-bit words.

Proof. Let hv=(a, b, c, d,e, f, g, h). where a, b, c, d,e, f, g, h are working variables before iterative part two. And hv1= (a1, b1, c1, d1,e1, f1, g1, h1), where a1, b1, c1, d1,e1, f1, g1, h1 are working variables after iterative part two.

The working variables are b-bit words. Then we have the function F(hv)=hv' and $F:\{0,1\}^{8\times w}\rightarrow\{0,1\}^{8\times w}$.

It is enough to known that, to a given hv', there is a hv1 make F(hv')=hv1.

To a given hv1', it is easy to backward the iterative part two and compute the unique value for hv1. So to a given hv1', there is a hv1 make F(hv1)=hv1'.

So the iterative part two of Dynamic SHA2 is a bijection $\xi:\{0,1\}^{8\times w}\rightarrow\{0,1\}^{8\times w}$ □

After iterative part one, all bits in message have been mixed. From the definition of function R1, 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 call function R1 9 times, all bits in working variables that before iterative part two will affect all bits in working variables that after iterative part two. So all message bits will affect all bits in last hash value.

3.1.3 Properties of iterative part three

In iterative part three, all message bits will be mixed seven times. And function COMP is called fourteen times. All bits in message words $W_1,W_2,W_3,W_4,W_5,W_6,W_7,W_9,W_{10},W_{11},W_{12},W_{13},W_{14},W_{15}$ have been used as parameters of function G, R and ROTR operation.

In iterative part one and three, all bits in message have been used as parameters of function G, R and ROTR operation. This will divide message space into 2^{512} (resp. 2^{1024}) parts.

3.2 Design rationale

The reasons for principle: *When message is changed, the calculation will be different.*

From the definition of function G, R and ROTR operations, it is easy to know all bits in message have been used as parameters of function G, R and ROTR operation. One bit different in message, different logical function or different ROTR operation will be done, and it will make the calculation different. Different message will lead to different calculation, these different calculations divide message space into 2^{512} (resp. 2^{1024}) parts. In a part, there is $2^{512-512}=1$ (resp. $2^{1024-1024}=1$) message value.

Why Dynamic SHA2 does not have constants?

The reasons why I decided not to use any constants is that Dynamic SHA2 is secure enough.

Controlling the differentials is hard in Dynamic SHA2:

In Dynamic SHA2, it is known that when message is changed, the calculation will be different. To analyze Dynamic SHA2, it need the unchangeable formulas that represent function describe function G, R and data-depend ROTR operation. There are three ways to analyze Dynamic SHA2:

1. Guess the parameters of function G, R and ROTR operation. The parameters of function G, R and ROTR operation divide message space into 2^{512} (resp. 2^{1024}) parts. In this way, someone select a part in the message value space. And there is only one message value in a part. He can not find collisions in the same part.
2. Someone can use Algebraic Normal Form (ANF) to represent Dynamic SHA2, but the ANFs that represent function R, R1 has up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials. If constitute the Arithmetic function based on ANF, the degree of the Arithmetic function represents function R, R1 and G is 261, 256, 5 (resp. 518, 512, 5), there are up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials in Arithmetic function represents function R, R1.
3. Someone can constitute Arithmetic functions to represent Dynamic SHA2 as in Appendix 2. But the Arithmetic function that represents function R and data-depend ROTR operation is complex exponential function with round-off instruction. After iterative parts, the Arithmetic function that represents function R and data-depend ROTR operation will be very huge.

3.3 Finding Preimages of Dynamic SHA2

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 SHA2, 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 part.

To do this, it needs the parameter of the ROTR operation and function G, R in Dynamic SHA2. But in Dynamic SHA2, when message changed, the parameter of the ROTR operation and function G, R will change. So attacker had to guess the parameter that will be used in Dynamic SHA2. From the definition of Dynamic SHA2, it is know that all bits in message are used as the parameter of the ROTR operation and function G, R. When attacker completes guessing parameters, he has guessed all bits in message.

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 SHA2

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 SHA2, when message is changed, the calculation is different. To get unchangeable formulas that represent hash function f, it need get ANFs for Dynamic SHA2. And the ANFs that represent function R, R1 has up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials.

3. To get unchangeable formulas that represent hash function f. It can constitute Arithmetic functions to represent Dynamic SHA2. And the Arithmetic functions that represent function R, R1 and G are exponential functions. Or someone had to constitute 261,256-degree (resp. 518, 512-degree) Arithmetic function to represent function R, R1 , and there are up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials in the Arithmetic function.
4. Guess the parameters of function G, R and ROTR operation. In this way, a part in the message value space is selected. And there is only one message value in a part. It can not find second preimages in the same part.
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 SHA2

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 finding 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 SHA2, when message is changed, the calculation is different. To get unchangeable formulas that represent hash function f, it need get ANFs for Dynamic SHA2. And the ANFs that represent function R,R1 has up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials.

3. To get unchangeable formulas that represent hash function f. It can constitute Arithmetic functions to represent Dynamic SHA2. And the Arithmetic functions that represent function R, R1 and G are exponential functions. Or someone had to constitute 261, 256 -degree (resp. 518, 512-degree) Arithmetic function to represent function R , and there are up to 2^{261}, 2^{256} (resp. 2^{518}, 2^{512}) monomials in the Arithmetic function..
4. Guess the parameters of function G, R and ROTR operation. This way is select a part in the message value space. And there is only one message value in a part. It can not find collisions in the same part.
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 SHA2

If the message bits are mixed less twice. The system will be weak, someone can backward Dynamic SHA2 as table E.2 show.

If the message bits are mixed at least twice, message word w_0, w_1, w_7, w_8 are used as the parameter of the ROTR operation and function G, R. It can backward iterative part as follow:

1. At first, there exist function COMPA and R1A:

   1.1 Function COMPA operates on sixteen words hv1, hv2,…,hv8, x0,x1,x1, x2, x3, x4, x5, x6, x7 and an integer t. Function COMPA as table 3.2, 3.3, 3.4 show.

1.2. Function R1A operates on eight words x1,x1, x2, x3, x4, x5, x6, x7, x8. produces one word y as output. Function R1 as table 3.2 and table 3.3.

$$T = hv1_1 - x4$$
$$hv1_0 = hv2_1$$
$$hv2_0 = hv3_1 - x0$$
$$hv3_0 = G(hv1_0, hv2_0, (hv4_1 - x5), t \wedge 3)$$
$$hv4_0 = ROTR^{32\text{-}((\mathrm{SHR}^{25}\mathrm{x0}) \wedge 31)}(hv5_1)$$
$$hv5_0 = hv6_1 - x6$$
$$hv6_0 = ROTR^{32\text{-}((\mathrm{SHR}^{20}\mathrm{x0}) \wedge 31)}(hv7_1)$$
$$hv7_0 = hv8_1 - x7$$
$$T = ROTR^{32\text{-}((\mathrm{SHR}^{15}\mathrm{x0}) \wedge 31)}(T)$$
$$hv8_0 = ((((((hv1_0 \oplus hv2_0) + hv3_0) \oplus hv4_0) + hv5_0) \oplus hv6_0) + hv7_0) \oplus T$$
$$T = hv1_0 - x1$$
$$hv1_{-1} = hv2_0$$
$$hv2_{-1} = hv3_0$$
$$hv3_{-1} = G(hv1_{-1}, hv2_{-1}, (hv4_0 - x2), SHR^{30}(x0))$$
$$hv4_{-1} = ROTR^{32\text{-}((\mathrm{SHR}^{10}\mathrm{x0}) \wedge 31)}(hv5_0)$$
$$hv5_{-1} = hv6_0 - x3$$
$$hv6_{-1} = ROTR^{32\text{-}((\mathrm{SHR}^{5}\mathrm{x0}) \wedge 31)}(hv7_0)$$
$$hv7_{-1} = hv8_0$$
$$T = ROTR^{32\text{-}(\mathrm{x0} \wedge 31)}(T)$$
$$hv8_{-1} = ((((((hv1_{-1} \oplus hv2_{-1}) + hv3_{-1}) \oplus hv4_0) + hv5_{-1}) \oplus hv6_{-1}) + hv7_{-1}) \oplus T$$

Table 3.2 function COMPA for Dynamic SHA2-224/256

$$T = hv1_1 - x4$$
$$hv1_0 = ROTR^{64-((\mathrm{SHR}^{54}\mathrm{x0})\wedge 63)} hv2_1$$
$$hv2_0 = hv3_1 - x0$$
$$hv3_0 = G(hv1_0, hv2_0, (hv4_1 - x5), SHR^{60}(x0) \wedge 3)$$
$$hv4_0 = ROTR^{64-((\mathrm{SHR}^{48}\mathrm{x0})\wedge 63)}(hv5_1)$$
$$hv5_0 = ROTR^{64-((\mathrm{SHR}^{42}\mathrm{x0})\wedge 63)}(hv6_1 - x6)$$
$$hv6_0 = ROTR^{64-((\mathrm{SHR}^{36}\mathrm{x0})\wedge 63)}(hv7_1)$$
$$hv7_0 = hv8_1 - x7$$
$$T = ROTR^{64-((\mathrm{SHR}^{30}\mathrm{x0})\wedge 63)}(T)$$
$$hv8_0 = ((((((hv1_0 \oplus hv2_0) + hv3_0) \oplus hv4_0) + hv5_0) \oplus hv6_0) + hv7_0) \oplus T$$
$$T = hv1_0 - x1$$
$$hv1_{-1} = hv2_0$$
$$hv2_{-1} = ROTR^{64-((\mathrm{SHR}^{24}\mathrm{x0})\wedge 63)}(hv3_0)$$
$$hv3_{-1} = G(hv1_{-1}, hv2_{-1}, (hv4_0 - x2), SHR^{62}(x0))$$
$$hv4_{-1} = ROTR^{64-((\mathrm{SHR}^{18}\mathrm{x0})\wedge 63)}(hv5_0)$$
$$hv5_{-1} = ROTR^{64-((\mathrm{SHR}^{12}\mathrm{x0})\wedge 63)}(hv6_0 - x3)$$
$$hv6_{-1} = ROTR^{64-((\mathrm{SHR}^{6}\mathrm{x0})\wedge 63)}(hv7_0)$$
$$hv7_{-1} = hv8_0$$
$$T = ROTR^{64-(\mathrm{x0}\wedge 63)}(T)$$
$$hv8_{-1} = ((((((hv1_{-1} \oplus hv2_{-1}) + hv3_{-1}) \oplus hv4_0) + hv5_{-1}) \oplus hv6_{-1}) + hv7_{-1}) \oplus T$$

Table 3.3 function COMPA for Dynamic SHA2-384/512

Dynamic SHA2-224/256	$\mathrm{t0} = (((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7$ $\mathrm{t1} = (SHR^{17}(\mathrm{t0}) \oplus \mathrm{t0}) \wedge (2^{17} - 1)$ $\mathrm{t2} = (SHR^{10}(\mathrm{t1}) \oplus \mathrm{t1}) \wedge (2^{10} - 1)$ $\mathrm{t} = (SHR^{5}(\mathrm{t2}) \oplus \mathrm{t2}) \wedge 31$ $y = ROTR^{32-t}(x8)$
Dynamic SHA2-384/512	$\mathrm{t0} = (((((x1 + x2) \oplus x3) + x4) \oplus x5) + x6) \oplus x7$ $\mathrm{t1} = (SHR^{36}(\mathrm{t0}) \oplus \mathrm{t0}) \wedge (2^{36} - 1)$ $\mathrm{t2} = (SHR^{18}(\mathrm{t1}) \oplus \mathrm{t1}) \wedge (2^{18} - 1)$ $\mathrm{t3} = (SHR^{12}(\mathrm{t2}) \oplus \mathrm{t2}) \wedge (2^{12} - 1)$ $\mathrm{t} = (SHR^{6}(\mathrm{t3}) \oplus \mathrm{t3}) \wedge 63$ $y = ROTR^{64-t}(x8)$

Table 3.4. function R1A for Dynamic SHA2

2. Base on function COMPA and R1A, we can backword the iterative steps as follow:

   2.1 Initialize eight last variables a_{16}, b_{16} ,..., h_{16} , eight first variables a_{-1} , b_{-1} ,..., h_{-1} and eight message words w_0 , w_1 ,..., w_7 .

   2.2 Input (a_{-1} , b_{-1} ,..., h_{-1} , w_0 , w_1 ,..., w_7 ,0) into function COMP, then we have a_1 , b_1 ,..., h_1 .

   2.3 Guess w_8 , w_9 ,..., w_{15} . and input (a_{16} , b_{16} ,..., h_{16} , w_8 , w_9 ,..., w_{15} ,1) into function COMPA, then we have a_{14} , b_{14} ,..., h_{14} .

   2.4 Input (a_{14} , b_{14} ,..., h_{14} , w_0 , w_1 ,..., w_7 ,1) into function COMPA, then we have a_{12} , b_{12} ,..., h_{12} .

   2.5 From a_{12} , b_{12} ,..., h_{12} , it can backword as follow:

```
For t=8 to 0
{
    $T = a_{t+4}$
    $a_{t+3} = b_{t+4}$
    $b_{t+3} = c_{t+4}$
    $c_{t+3} = d_{t+4}$
    $d_{t+3} = e_{t+4}$
    $e_{t+3} = f_{t+4}$
    $f_{t+3} = g_{t+4}$
    $g_{t+3} = h_{t+4}$
    $h_{t+3} = R1A(a_{t+3}, b_{t+3}, c_{t+3}, d_{t+3}, e_{t+3}, f_{t+3}, g_{t+3}, T)$
}
```

   Then we have a_3 , b_3 ,..., h_3 .

   2.6 Operate on a_3 , b_3 ,..., h_3 and a_1 , b_1 ,..., h_1 as table E.2 that in Appendix 5. then we have w_8' , w_9' ,..., w_{15}' .

   2.7 Compare w_8' , w_9' ,..., w_{15}' and w_8 , w_9 ,..., w_{15} . If (w_8' , w_9' ,..., w_{15}')= (w_8 , w_9 ,..., w_{15}), then we find a collision. If (w_8' , w_9' ,..., w_{15}') $\neq$ (w_8 , w_9 ,..., w_{15}), we had to guess (w_8 , w_9 ,..., w_{15}) again. The size of the space of (w_8' , w_9' ,..., w_{15}' , w_8 , w_9 ,..., w_{15}) is $2^{16\times32} = 2^{512}$ (resp.

$2^{16\times 64}=2^{1024}$) and the size of the space of (w_8 , w_9 ,..., w_{15}) is 2^{256} (resp. 2^{512}). So the probability of (w_8' , w_9' ,..., w_{15}')= (w_8 , w_9 ,..., w_{15}) is 2^{-256} (resp. 2^{-512}).

So if the message bits are mixed at least twice, the probability of find the collision is less than 2^{-128} (resp. 2^{-256}).

3.7 Security of message digest truncations

3.7.1 Security of message digest truncations of Dynamic SHA2-224

The final result of Dynamic SHA2-224 include eight working variables a,b,c,d,e,f,g,h, it iclude 256 bits. The output of Dynamic SHA2-224 include seven working variables a,b,c,d,e,f,g, it iclude 224 bits.

So the length of the final result of Dynamic SHA2-224 is 256, and the length of the output of Dynamic SHA2-224 are 224. The size of the space of final result of Dynamic SHA2-224 is 2^{256}, The size of the space of output of Dynamic SHA2-224 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 SHA2-224, 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 SHA2-384

The final result of Dynamic SHA2-384 include eight working variables a,b,c,d,e,f,g,h, it iclude 512 bits. The output of Dynamic SHA2-384 include six working variables a,b,c,d,e,f, it iclude 384 bits.

So the length of the final result of Dynamic SHA2-384 is 512, and the length of the output of Dynamic SHA2-384 are 384. The size of the space of final result of Dynamic SHA2-384 is 2^{512}, The size of the space of output of Dynamic SHA2-384 is 2^{384}. To given output 6-tuple(a', b', c', d', e', f'), there exist $2^{2\times 64}=2^{128}$ 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 SHA2-384, 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 SHA2:

1. There is no any constant in Dynamic SHA2. Use constants will increase system security.

2. In Keyed Hash function, the initial hash value is random variable to attacker. If Dynamic SHA2 is used in Keyed Hash function, by theorem 4, 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.

3. If some algorithms that based on Arithmetic functions are developed to break Dynamic SHA2. The message expansions will increase the degree of the Arithmetic function that represents Dynamic SHA2. 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 2, all hash value will have same probability.

An examlep as follow:

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

Dynamic SHA2-224/256	$t0=((((((x1+x2)\oplus x3)+x4)\oplus x5)+x6)\oplus x7)+x8$ $t1=((((((x9+x10)\oplus x11)+x12)\oplus x13)+x14)\oplus x15)+x16$ $p(i)=SHR^{4\times i}(t0)\wedge(15) \qquad 0\le i\le 7$ $p(i)=SHR^{4\times(i-8)}(t1)\wedge(15) \qquad 8\le i\le 15$ $t2=\sum_{i=0}^{15}p(i)$ $w_{i+15}=w_i\oplus w_{p(i)\oplus t2} \qquad 0\le i\le 15$
Dynamic SHA2-384/512	$t0=((((((((((((((x1+x2)\oplus x3)+x4)\oplus x5)+x6)\oplus x7)+$ $+x8)\oplus x9)+x10)\oplus x11)+x12)\oplus x13)+x14)\oplus x15)+x16$ $p(i)=SHR^{4\times i}(t0)\wedge(15) \qquad 0\le i\le 15$ $t1=\sum_{i=0}^{15}p(i)$ $w_{i+15}=w_i\oplus w_{p(i)\oplus t1} \qquad 0\le i\le 15$

Table 4.1. 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 the iterative part will include four part, the fourth iterative part as follow:

```
2.4 The fourth iterative part
For t=0 to 7
{
    $COMP(a,b,c,d,e,f,g,h,w_{16},w_{17},w_{18},w_{19},w_{20},w_{21},w_{22},w_{23},t)$
    $COMP(a,b,c,d,e,f,g,h,w_{24},w_{25},w_{26},w_{27},w_{28},w_{29},w_{30},w_{31},t)$
}
```

Table 4.2. the fourth iterative part for Dynamic SHA2

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

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

Dynamic SHA2 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 SHA2

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 SHA2-224/256/384/512 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 SHA2. By theorem 5.1, it is known that if someone want to find collision of HMAC that constituted with Dynamic SHA2, 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)=$\mathrm{F}(\mathrm{r}v \,||\, \mathrm{M} \oplus Rv \,||\, PL(rv))$

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 $M1=m||'1'||\underbrace{'0'||\ldots||'0'}_{rlen-|m|-1}$
6. let $Rv1= \underbrace{rv||\ldots||rv}_{|M|/rlen+1}$
7. let Rv= Rv1 truncated as |M1|
8. let M=rv||(M1 $\oplus$ 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 SHA2 can be used function F in the definition. Dynamic SHA2 are used in Randomized Hashing function as follow:

RF(rv,m)=F(PM(rv,m)), where F is Dynamic SHA2.

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 $\oplus$ b2=1.
   - 2.2 the first r2len bits is random bit string rv2.
   - 2.3. let Rv2= $rv2\underbrace{||\ldots||}_{|m3|/r2len+2}rv2$
   - 2.4 let Rv= Rv2 truncated as (m2len+r2len)
   - 2.5 let m3'= m3 $\oplus$ 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 $\oplus$ 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 $\oplus$ 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 SHA2. 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 $\neq$ 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 SHA2, 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 SHA2.

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 SHA2-224/256 *seedlen* = 440

if Dynamic SHA2-384/512 *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 SHA2-224/256 *seedlen* = 440

if Dynamic SHA2-384/512 *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 SHA2-224/256 *seedlen* = 440

if Dynamic SHA2-384/512 *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 $additional\_inpu \neq$ Null, then do
    3.1 $w$ = Hash (0x02 || $V$ || $additional\_input$).
    3.2 $V = (V + w) \bmod 2^{seedlen}$ .
4. ($returned\_bits$) = Hashgen ($requested\_number\_of\_bits$, $V$).
5. $H$ = Hash (0x03 || $V$).
6. $V = (V + H + C + reseed\_counter) \bmod 2^{seedlen}$ .
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 SHA2 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 SHA2, he had to find the V from 2^{216} (resp. 2^{184}, 2^{496}, 2^{376}) messages.

6 Security of Dynamic SHA2 with length extension attack and multicollision attack

6.1 Security of Dynamic SHA2 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 SHA2. 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 SHA2 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 SHA2 is 2^{-112} (resp. 2^{-128}, 2^{-192}, 2^{-256}). Then the probability of finding 2^r -way collision of Dynamic SHA2 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 SHA2 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 nonlinear, and composite function of Boolean functions and Arithmetic functions also make cipher system nonlinear. Dynamic SHA2 carries out the two suggestions.

Function G, R and data-depend ROTR operations 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 SHA2: *When message is changed, the calculation will be different.* And I bring in data depend function G, R and data-depend ROTR operations, and use bits in message as parameters of function G, R and and data-depend ROTR operations. These steps realize the principle. The principle enabled us to build a compression function of Dynamic SHA2 that has not new variable, the iterative part include three iterative parts, 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”

Hhttp://citeseer.ist.psu.edu/574240.htmlH

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 SHA2” http://eprint.iacr.org/2008/146

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 $F:\{0,1\}^n \rightarrow \{0,1\}^r$ 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 $F^{(s)}:\{0,1\}^n \rightarrow \{0,1\}(s=1,2,...,r)$, 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 six words x1,x2,x3,x4,x5,x6,x7,x8 and an integer t and produces a word y as output, where $0 \le t < w$. So we have $R:\{0,1\}^{8\times w+\log_2^w} \rightarrow \{0,1\}^w$, It is easy to know that one-bit different in words x1,x2,x3,x4,x5,x6,x7,x8. Because the parameter of the rotate right operation is depend on message. With different message, different rotate right operation will be done. So the bit in output maybe changed.

So the ANFs to represent function R have up to $2^{8\times w} \times w$ monomials, where w is bit length of the word.

Function G

Function G operates on six words x1,x2,x3 and an integer t and produces a word y as output, where $0 \le t < 4$. So we have $R:\{0,1\}^{3\times w+2} \rightarrow \{0,1\}^w$.

If function G is not data depend function, the integer t is constant. When i-th bit in words x1,x2,x3 change, i-th bit in output maybe change. Then the ANFs to represent function R have up to 2^3 monomials.

If function G is not data depend function, the integer t is variable. It is easy to know that one-bit different in integer t, different logical will be called, every bit in output maybe change. One-bit different in words x1,x2,x3, a bit in output maybe change. Then the ANFs to represent function R have up to $2^{3+2} = 2^5$ monomials.

Function R1

Function R1 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 R1 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 this 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. 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. 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 function F 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 2. In GF(2), there exists $x^k = x \quad k > 0$.
Proof. In GF(2), $x \in \{0,1\}$.
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 $a_0 \oplus a_1x_1 \oplus ... \oplus a_nx_n \oplus a_{1,2}x_1x_2 \oplus ... \oplus a_{n-1,n}x_{n-1}x_n \oplus ... \oplus a_{1,2,...,n}x_1..x_n$, where $a_\lambda \in \{0,1\}$. If replace every calculation in the ANF base on table B.1 and simplified by theorem 2, 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:

$$y = SHR^k(x) = \left|\frac{x}{2^k}\right| \qquad (1.0)$$

If operation $y = SHR^k(x)$ is not data-depend operation, the k in equation (1.0) is constant, and equation (1.0) is linear equation. The derivative function of linear equation is constant.

If operation $y = SHR^k(x)$ is data-depend operation, the k in equation (1.0) is variable. And equation (1.0) will be exponential function with round-off instruction. It is hard to represent exponential function with linear equation.

4. 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^{261} (resp. 2^{518}) monomials. By theorem 2 and the input has 261(resp. 518) bits, so the highest degree monomial of the Arithmetic function is $\prod_{i=0}^{260} x_i$ (resp. $\prod_{i=0}^{517} x_i$), where x_i is i-th input bit. The degree of the Arithmetic function represents function R is up to 261(resp. 518). There exists:

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

where c is constant, x_i is i-th input bit of function R, bn is bit number of input, and bn equal 261(resp. 518).

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:

$$\begin{aligned}
y &= ROTR^{k}(x) \\
&= \left|\frac{x}{2^{k}}\right| + (x - \left|\frac{x}{2^{k}}\right| \times 2^{k}) \times 2^{n-k} \\
&= x \times 2^{n-k} - \left|\frac{x}{2^{k}}\right| \times (2^{n} - 1)
\end{aligned} \tag{1.1}$$

If function $y = ROTR^{k}(x)$ is not data-depend function, the k in equation (1.1) is constant, and equation (1.1) 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 (1.1) is variable. And equation (1.1) 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 SHA2, function R is data-depend function. And if use equation (1.1) represents function R, the equation (1.1) will be complex exponential function. After several rounds, equation (1.1) will be iteration function with equation (1.1), it will be very huge and complex, and there exists no mathematical theory that reduces the size of equation (1.1). It is hard to analyses Dynamic SHA2 that includes function R.

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

By Theorem 2 and table B.1, function $G_{i}(x1, x2, x3)$ can be represented as follow:

$$G_t(x1,x2,x3)=\begin{cases}\sum_{i=0}^{w-1}(x1_i+x2_i+x3_i-2\times x1_i\times x2_i-2\times x1_i\times x3_i-\\ \quad 2\times x2_i\times x3_i+4\times x1_i\times x2_i\times x3_i)\times 2^i & t=0\\ \sum_{i=0}^{w-1}(x3_i+x1_i\times x2_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=1\\ \sum_{i=0}^{w-1}(1-x1_i-x3_i+2\times x1_i\times x3_i+x1_i\times x2_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=2\\ \sum_{i=0}^{w-1}(1-x2_i-x3_i+2\times x2_i\times x3_i+x1_i\times x3_i-2\times x1_i\times x2_i\times x3_i)\times 2^i & t=3\end{cases} \quad (1.2)$$

$x1_i, x2_i, x3_i$ is i-th bit of x1, x2, x3. In system (1.2), it is known that G_i are cubic equations. The degree of the Arithmetic function that represent function $G(x1,x2,x3)$ is 3. And there are 7(resp. 5, 6) monomials in the Arithmetic function.

If function G is not data-depend function. It can look the equation (1.2) as cubic equations. It is hard to represented equation (1.2) with linear function. And there exists:

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

And c is constant.

If function G is data-depend function, the function G will be represented with Arithmetic function as follow:
$0\le t\le 3$, let t=(t_1, t_0).

$$\begin{aligned}&G(x1,x2,x3,t)=\\ &(1-t_1)\times(1-t_0)\times\sum_{i=0}^{w-1}(x1_i+x2_i+x3_i-2\times x1_i\times x2_i-2\times x1_i\times x3_i-2\times x2_i\times x3_i+\\ &\quad +4\times x1_i\times x2_i\times x3_i)\times 2^i\\ &+(1-t_1)\times t_0\times\sum_{i=0}^{w-1}(x1_i\times x2_i+x3_i-2\times x1_i\times x2_i\times x3_i)\times 2^i\\ &+t_1\times(1-t_0)\times\sum_{i=0}^{w-1}(1-x1_i-x3_i+2\times x1_i\times x3_i+x1_i\times x2_i-2\times x1_i\times x2_i\times x3_i)\times 2^i\\ &+t_1\times t_0\times\sum_{i=0}^{w-1}(1-x2_i-x3_i+2\times x2_i\times x3_i+x1_i\times x3_i-2\times x1_i\times x2_i\times x3_i)\times 2^i\end{aligned}$$

$$\begin{aligned}=\sum_{i=0}^{w-1}&(x1_i+x2_i+x3_i+t_1-2\times x1_i\times x2_i-2\times x1_i\times x3_i-2\times x2_i\times x3_i-x1_i\times t_0-\\ &-2\times x1_i\times t_1-2\times x2_i\times t_0-x2_i\times t_1-2\times x3_i\times t_1+4\times x1_i\times x2_i\times x3_i+\\ &+3\times x1_i\times x2_i\times t_0+3\times x1_i\times x2_i\times t_1+2\times x1_i\times x3_i\times t_0+4\times x1_i\times x3_i\times t_1+\\ &+2\times x2_i\times x3_i\times t_1+2\times x1_i\times t_0\times t_1-6\times x1_i\times x2_i\times x3_i\times t_0-6\times x1_i\times x2_i\times x3_i\times t_1-\\ &-4\times x1_i\times x2_i\times t_0\times t_1-3\times x1_i\times x3_i\times t_0\times t_1+6\times x1_i\times x2_i\times x3_i\times t_0\times t_1)\times 2^i\end{aligned} \quad (1.3)$$

t_1, t_0 is i-th bit of t. It is easy to known that the degree of the Arithmetic function that represent function $G(x1, x2, x3, t)$ is 5. And there are 24 monomials in the Arithmetic function. And there exists:

$$\frac{d^5(y)}{d(x1_i)d(x2_i)d(x3_i)d(t_0)d(t_1)} = 6 \times 2^i$$

Compare equation (1.2) and equation (1.3), if function G is data-depend function, the degree of the Arithmetic function will be higher, and ther are more monomials in the Arithmetic function. This make it is harder to analyses Dynamic SHA2.

6. Constitute Arithmetic functions to represent 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 2 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:

$$\begin{aligned} y &= ROTR^k(x) \\ &= \left|\frac{x}{2^k}\right| + (x - \left|\frac{x}{2^k}\right| \times 2^k) \times 2^{n-k} \\ &= x \times 2^{n-k} - \left|\frac{x}{2^k}\right| \times (2^n - 1) \end{aligned} \qquad (1.4)$$

If function $y = ROTR^k(x)$ is data-depend function, the k in equation (1.4) is variable, and equation (1.4) is exponential function. And equation (1.4) 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 SHA2, function R1 is data-depend function. And if use equation (1.4) represents function R1, 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 (1.4) will be complex exponential function. After several rounds, equation (1.4) will be iteration function with equation (1.4), it will be very huge and complex, and there exists no mathematical theory that reduces the size of equation (1.4). It is hard to analyses Dynamic SHA2 that includes function R1.

Dynamic SHA2-224/256	$t0 = (((((a + b) \oplus c) + d) \oplus e) + f) \oplus g$ $t1 = (SHR^{17}(t0) \oplus t0) \wedge (2^{17} - 1)$ $t2 = (SHR^{10}(t1) \oplus t1) \wedge (2^{10} - 1)$ $k = (SHR^{5}(t2) \oplus t2) \wedge 31$
Dynamic SHA2-384/512	$t0 = (((((a + b) \oplus c) + d) \oplus e) + f) \oplus g$ $t1 = (SHR^{36}(t0) \oplus t0) \wedge (2^{36} - 1)$ $t2 = (SHR^{18}(t1) \oplus t1) \wedge (2^{18} - 1)$ $t3 = (SHR^{12}(t2) \oplus t2) \wedge (2^{12} - 1)$ $k = (SHR^{6}(t3) \oplus t3) \wedge 63$

Table B.2. function K for Dynamic SHA2

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

Appendix 3: Function G and Function R and Function R1

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

1, Function G:

Function y=G(x1, x2, x3, t) operates on tree words x1,x2,x3 and an integer t, $0 \le t \le 3$. Function G use the integer t select a logical function from f_0, f_1, f_2, f_3. And y, x1, x2, x3 are w-bit word. So the bit-length of (x1,x2,x3,t) is $3 \times w+2$, the bit-length of y is w.

To a given value y'=G(x1,x2,x3,t), there is $2^{2\times w+2}$ 4-tuple (y',x1,x2,t). To a given 4-tuple (y',x1',x2', t'). There is the relation:

$$x4' = \begin{cases} x1' \oplus x2' \oplus y' & t=0 \\ (x1' \wedge x2') \oplus y' & t=1 \\ (\neg(x1' \vee y')) \vee (x1' \wedge (x2' \oplus y')) & t=2 \\ (\neg(x1' \vee (x2' \oplus y'))) \vee (x1' \wedge \neg y') & t=3 \end{cases}$$

To given 4-tuple (y',x1',x2',t'), it can compute the value for x3'. So there are $2^{2\times w+2}$ 4-tuple (x1,x2,x3,t) have the same value y'. x1, x2, x3, t are random and uncorrelated variable, there is:

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

$$p(y) = \sum_{i1=0}^{2^w-1} \sum_{i2=0}^{2^w-1} \sum_{i3=0}^{2^w-1} \sum_{i4=0}^{3} p(y \mid (x1_{i1}, x2_{i2}, x3_{i3}, t_{i4})) \times p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3}) \times p(t_{i4})$$

$$p(y) = p(x1_{i1}) \times p(x2_{i2}) \times p(x3_{i3}) \times p(t_{i4}) \times \sum_{i1=0}^{2^w-1} \sum_{i2=0}^{2^w-1} \sum_{i3=0}^{2^w-1} \sum_{i4=0}^{3} p(y \mid (x1_{i1}, x2_{i2}, x3_{i3}, t_{i4}))$$

$$p(y) = 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-2} \times 2^{2\times w+2} = 2^{-w}$$

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

2, Function R:

Function y=R(x1,x2,x3,x4,x5,x6,x7,x8,t) operates on eight words x1,x2,x3,x4,x5,x6, x7, x8 and an integer t. To a given value y'=R(x1,x2,x3,x4,x5,x6,x7,x8,t), there is $2^{7\times w} \times w$ 9-tuple (y',x1',x2',x3',x4',x5',x6',x7',t'). To a given 9-tuple (y',x1',x2',x3',x4', x5', x6', x7',t'). There is the relation:

$$x8 = ((((((x1' \oplus x2') + x3') \oplus x4') + x5') \oplus x6') + x7') \oplus ROTR^{w-t'}(y')$$

To given 9-tuple (y',x1',x2',x3',x4',x5',x6',x7',t'), it can compute the value for x8, So there are $2^{7\times w}\times w$ 9-tuple (y',x1',x2',x3',x4',x5',x6',x7',t') have the same value y'. x1,x2,x3,x4,x5,x6,x7,x8,t are random and uncorrelated variable, there is:

$$p(x1) = p(x2) = p(x3) = p(x4) = p(x5) = p(x6) = p(x7) = p(x8) = 2^{-w}$$
$$p(t) = w^{-1}$$
$$p(y) = \sum_{i1=0}^{2^w-1}\sum_{i2=0}^{2^w-1}\cdots\sum_{i8=0}^{2^w-1}\sum_{i9=0}^{w-1} p(y \mid (x1_{i1}, x2_{i2}, ..., x8_{i8}, t_{i9})) \times p(x1) \times ... \times p(x8) \times p(t)$$
$$p(y) = 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-w} \times 2^{-w} \times w^{-1} \times 2^{7\times w} \times w = 2^{-w}$$

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

Function R1

Function y=R1(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 $2^{w-\log_2^w}$ $(w-\log_2^w)-\mathrm{tuple}(t0_{\log_2^w},\dots,t0_{w-1})$. To a given $(w-\log_2^w)-tuple$ $(t0'_{\log_2^w},\dots,t0'_{w-1})$, it can compute the $\log_2^{w}$ – tuple for $(t0_0,\dots,t0_{\log_2^w-1})$. 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:

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

If x1, x2, x3, x4, x5, x6, x7, x8 are random and uncorrelated words, function R1 will produce random word and $p(y) = 2^{-w}$.

Appendix 4: Some thing about Dynamic SHA2

1. Why Dynamic SHA2 use function G , R and function R1

The reason Dynamic SHA2 use function G, R and function R1 is:

1. When the variables are random and uncorrelated, function G , R and R1 will produce random output. This makes the last hash values has close probability.
2. Function G, R and R1 are data-depend function, it is hard to describe data-depend function with linear function, and it is hard to analyze data-depend function with differential analysis. The arithmetic function that describe function G, R and R1 is up to 5, 261(resp. 518), 256(resp.512). And the ANFs that describe function G, R and R1 has up to 32, $2^{8\times w+\log_2^{w}}$, $2^{8\times w}$ monomials.

2. It is hard analysis Dynamic SHA2 with linear function and differential analysis

To analyze the relationship between message and hash value, it need the unchangeable formulas that represent hash function. And when message is changed, the calculation will be different.

The ANFs that describe function R, R1 has up to $2^{8\times w+\log_2^{w}}$ $2^{8\times w}$ monomials.

The degree of the arithmetic function that describe function R, R1 is up to 261(resp.518), 256(resp.512). Or it needs construction exponential function to describe function R, R1 and G.

So it is hard analysis Dynamic SHA2 with linear function and differential analysis.

3. Avalanche of Dynamic SHA2.

After the first iterative part, all bits in message have been mixed. The second iterative part includes function R1. It is easy to know that one bit different in working variables a, b, c, d, e, f, g will lead to different ROTR operation been done. And after the second iterative part, every bit in working variables that before the second iterative part will affect all bits in working variables that after the second iterative part.

Appendix 5: Spreading of Dynamic SHA

To simplification, Let:

1.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.

2. 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 function COMP called.

3. $H_i(\text{hv}(-1), \text{MW1}, \text{MW2}) = \text{hv}(i) \quad 1 \le i \le 15$

4. Message word and working variables are b-bit words.

From the definition of Dynamic SHA2, it is easy know that function COMP had been called sixteen times, when function COMP is called, MW1 or MW2 will be mixed. So it can describe Dynamic SHA2 as follow:

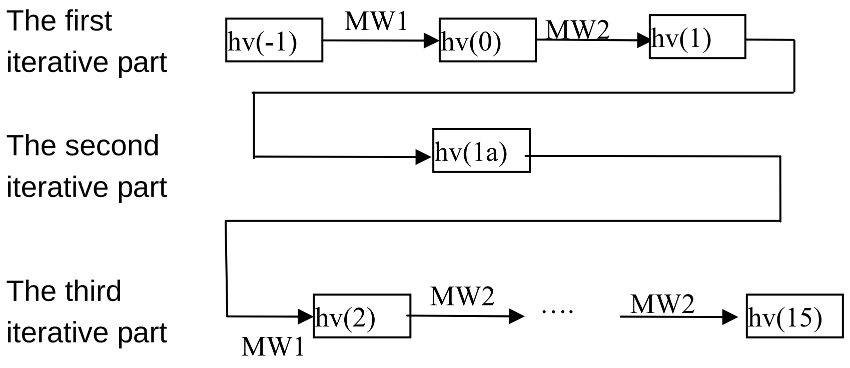

Table E.1 data processing of Dynamic SHA2

At first there are two theorems:

Theorem 3:

To function $COMP(a,b,c,d,e,f,g,h,w0,w1,w2,w3,w4,w5,w6,w7,t)$*, there is:*

1.MW=(W0,W1,W2,W3,W4,W5,W6,W7), where W0,…,W7 are words that mixed.

2. hva=(a0, b0, 0c, d0, e0, 0f, g0, h0). Where a0, b0, c0, d0,e0, f0, g0, h0 are working variables that before call function COMP.

3. hvb=(a1, b1, c1, d1, e1, f1, g1, h1). Where a1, b1, c1, d1,e1, f1, g1, h1 are working variables that after call function COMP.

working variables are b-bit word. hva, MW are random and uncorrelated.

Then there exist:

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

(2),p(hvb|MW)= $2^{-8\times b}$

(3),p(hvb|hva)= $2^{-8\times b}$

Proof.

The integer t in function COMP is decided by which round function COMP be. So the integer t can be look as constant. And we can use function $hvb = F(hva, MW)$ describe function COMP. And we have $F : \{0,1\}^{16\times b} \to \{0,1\}^{8\times b}$. hva, MW are random and uncorrelated. So there exist p(hva)= $2^{-8\times b}$ and p(MW)= $2^{-8\times b}$

There are $2^{8\times b}$ MW. To a given MW', there exist:

To a given hva', from the definition of F, there is only a hvb that make $hvb = F(hva', MW')$.

And to a given hvb', it can backward function F, and there is only a hva that make $hvb' = F(hva, MW')$. So there exist:

$$p(hvb) = \sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hvb \mid (hva(i1), MW(i2))) \times p(hva(i1)) \times p(MW(i2))$$

$$p(hvb) = p(hva) \times p(MW) \times \sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hvb \mid (hva(i1), MW(i2)))$$

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

$$p(hvb \mid MW) = \sum_{i1=0}^{2^b-1} p((hvb \mid MW) \mid hva(i1)) \times p(hva(i1))$$

$$p(hvb \mid MW) = p(hva) \times \sum_{i1=0}^{2^b-1} p((hvb \mid MW) \mid hva(i1))$$

$$p(hvb \mid MW) = 2^{-8\times b}$$

Dynamic SHA2-224/256	$w0 = c1 - a0$ $w5 = d1 - G(b1, a0, b0, t \wedge 3)$ $w6 = f1 - ROTR^{32-(SHR^{10} w0) \wedge 31}(d0)$ $w7 = h1 - ROTR^{32-(SHR^{5} w0) \wedge 31}(f0)$ $w3 = ROTR^{32-(SHR^{20} w0) \wedge 31}(g1) - e0$ $w2 = ROTR^{32-(SHR^{25} w0) \wedge 31}(e1) - G(a0, b0, c0, SHR^{30} w0)$ $w1 = b1 - R(a0, b0, c0, d0, e0, f0, g0, h0, w0 \wedge 31)$ $d' = ROTR^{32-(SHR^{25} w0) \wedge 31}(e1)$ $e' = f1 - w6$ $f' = e0 + w3$ $g' = ROTR^{32-((SHR^{5} w0) \wedge 31)}(e)$ $w4 = a1 - R(b1, a0, b0, d', e', f', g', g0, (SHR^{15}(w0)) \wedge 31)$
Dynamic SHA2-384/512	$w0 = c1 - a0$ $a' = ROTR^{64-((SHR^{54} w0) \wedge 63)} b1$ $c' = ROTR^{64-((SHR^{24} w0) \wedge 63)} b0$ $w5 = d1 - G(a', a0, c', t \wedge 3)$ $w6 = f1 - ROTR^{128-((SHR^{42} w0) \wedge 63)-((SHR^{18} w0) \wedge 63)}(d0)$ $w7 = h1 - ROTR^{64-(SHR^{6} w0) \wedge 63}(f0)$ $w3 = ROTR^{128-((SHR^{36} w0) \wedge 63)-((SHR^{12} w0) \wedge 63)}(g1) - e0$ $w2 = ROTR^{64-(SHR^{48} w0) \wedge 31}(e1) - G(a0, b0, c0, SHR^{62} w0)$ $w1 = ROTR^{64-(SHR^{54} w0) \wedge 31}(b1) - R(a0, b0, c0, d0, e0, f0, g0, h0, w0 \wedge 63)$ $d' = ROTR^{64-(SHR^{48} w0) \wedge 63}(e1)$ $e' = ROTR^{64-(SHR^{18} w0) \wedge 63}(d0)$ $f' = ROTR^{64-(SHR^{36} w0) \wedge 63}(g1)$ $g' = ROTR^{64-((SHR^{6} w0) \wedge 63)}(f0)$ $w4 = a1 - R(a', a0, c', d', e', f', g', g0, (SHR^{15}(w0)) \wedge 31)$

Table E.2. Relationship of hva, hvb

(3)

To given hva’, there exist:

To a given hvb’, there is the relationship as table E.2, It is easy to compute the value for MW that make $hvb' = F(hva', MW)$. So there exist:

$$p(hvb \mid hva) = \sum_{i=0}^{2^b-1} p((hvb \mid hva) \mid MW(i)) \times p(MW(i)) = 2^{-8\times b} = 2^{-8\times b} \qquad \square$$

By theorem 3, to function COMP, it is easy to know that:

To a given hva', mix different message words MW, the hvb will be different.

Mix given message words MW', if the hva is different, the hvb will be different.

Theorem 4. *In Dynamic SHA2, there exist:*

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

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

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

$j = 1,\ldots., 15$

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-1),MW)=hv(i), MW is mixed words MW1 or MW2.

To a given hv(i)' $i = 1,\ldots., 15$, 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 $\mathrm{H}_{\mathrm{i}}(\mathrm{hv(-1)}, \mathrm{MW1}', \mathrm{MW2}') = \mathrm{hv(i)}'$.

So there exist:

$$p(hv(i)) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i) \mid (hv(-1)_{i0}, MW1_{i1}, MW2_{i2})) \times p(hv(-1)_{i0}, MW1_{i1}, MW2_{i2})$$
$$p(hv(i)) = p(hv(-1), MW1, MW2) \times \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1}\sum_{i2=0}^{2^b-1} p(hv(i) \mid (hv(-1)_{i0}, MW1_{i1}, MW2_{i2}))$$
$$p(hv(i)) = 2^{-8\times b} \times 2^{-8\times b} \times 2^{-8\times b} \times 2^{8\times b} \times 2^{8\times b} = 2^{-8\times b}$$
$$p(hv(i) \mid MW1) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i) \mid MW1) \mid (hv(-1)_{i0}, MW2_{i1})) \times p(hv(-1)_{i0}, MW2_{i1})$$
$$p(hv(i) \mid MW1) = p(hv(-1), MW2) \times \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i) \mid MW1) \mid (hv(-1)_{i0}, MW2_{i1}))$$
$$p(hv(i) \mid MW1) = 2^{-8\times b} \times 2^{-8\times b} \times 2^{8\times b} = 2^{-8\times b}$$
$$p(hv(i) \mid MW2) = \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i) \mid MW2) \mid (hv(-1)_{i0}, MW1_{i1})) \times p(hv(-1)_{i0}, MW1_{i1})$$
$$p(hv(i) \mid MW2) = p(hv(-1), MW1) \times \sum_{i0=0}^{2^b-1}\sum_{i1=0}^{2^b-1} p((hv(i) \mid MW2) \mid (hv(-1)_{i0}, MW1_{i1}))$$
$$p(hv(i) \mid MW2) = 2^{-8\times b} \times 2^{-8\times b} \times 2^{8\times b} = 2^{-8\times b}$$

□

Theorem 5. *In Dynamic SHA2, to a given hv(-1), there exist:*

p(hv(2)| (hv(-1),MW1))= $2^{-8\times b}$

Proof. To simplification, Let F(hv(i-1),MW)=hv(i), MW is mixed words MW1 or MW2. Let F2(hv(1))= hv(1a) is the second iterative part.

Let F1(hv(1))=hv(1a).

To a given 3-tuple (hv(2)’,hv(-1)’,MW1’). By theorem 3, there exist a 2-tuple (hv(0),hv(1a)) that make F(hv(-1)’,MW1’)=hv(0) and F(hv(1a),MW1’)=hv(2)’.

To a given hv(1a)’, frome the definition of the second iterative part , there exist a hv(1) that make F2(hv(1))=hv(1a)’.

To a given 2-tuple (hv(0)’,hv(1)’) . By theorem 3, there exist a MW2 that make F(hv(0)’,MW2)=hv(1)’.

So there exist:

$$p(hv(2) \mid (hv(-1), MW1)) = \sum_{i=0}^{2^b-1} p((hv(1) \mid (hv(-1), MW1)) \mid MW2_i) \times p(MW2_i)$$
$$p(hv(2) \mid (hv(-1), MW1)) = p(MW2) \times \sum_{i=0}^{2^b-1} p((hv(1) \mid (hv(-1), MW1)) \mid MW2_i)$$
$$p(hv(2) \mid (hv(-1), MW1)) = 2^{-8\times b} \times 1 = 2^{-8\times b}$$

□

By theorem 4 and 5, it is 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 first the second iterative part, the bits in message have been mixed, the mixed bits and working variables value are not uncorrelated, it is hard to analyze the probability of hash value. To get better property of spreading, Dynamic SHA2 adopt ways as follow:

1. When the variable of function COMP is random value. Function COMP will produce random value.