Some Observations on Strengthening the SHA-2 Family

Somitra Kumar Sanadhya* and Palash Sarkar

Applied Statistics Unit,
Indian Statistical Institute,
203, B.T. Road, Kolkata,
India 700108.
somitra_r@isical.ac.in, palash@isical.ac.in

9^{th} May 2008

Abstract. In this work, we study several properties of the SHA-2 design which have been utilized in recent collision attacks against reduced SHA-2. We suggest small modifications to the SHA-2 design to thwart these attacks. The cost of SHA-2 evaluations does not change significantly due to our modifications but the new design provides resistance to the recent collision attacks.
Further, we describe an easy method of exhibiting non-randomness of the compression functions of the entire SHA family, that is SHA-0, SHA-1 and all the hash functions in SHA-2. Specifically, we show that given any IV_1 and any pair of messages M_1 and M_2, an IV_2 can be easily and deterministically constructed such that the relation $H(\mathsf{IV}_1, M_1) - \mathsf{IV}_1 = H(\mathsf{IV}_2, M_2) - \mathsf{IV}_2$ holds. For a truly random hash function H outputting a k-bit digest, such a relation should hold with probability 2^{-k}.
We introduce the general idea of "multiple feed-forward" in the context of construction of cryptographic hash functions. When used in SHA designs, this technique removes the non-randomness mentioned earlier. Perhaps more importantly, it provides increased resistance to the Chabaud-Joux type "perturbation-correction" collision attacks. The idea of feed-forward is taken further by introducing the idea of feed-forward across message blocks. This provides quantifiably better resistance to Joux type generic multi-collision attacks. For example, with our modification of SHA-256, finding 2^r messages which map to the same value will require $r \times 2^{384}$ invocations of the compression function.

1 Introduction

Following the attacks on SHA-0 [1] and SHA-1 [16], the attention of the cryptanalysis community has been directed to the SHA-2 family. Recent attacks against SHA-2 starting with [10], and followed by [14] and [6], have utilized certain previously unknown properties in the round function of SHA-2. These have led to 22-step attacks [11] against SHA-512 and 24-step attacks [6] against SHA-256. While none of these attacks threaten any of the security properties of the full SHA-2 hash functions, it is also true that the features of the compression function that have been exploited in the attack are undesirable. In view of the NIST call for potential SHA-3 proposals [3], it is of interest to take a closer look at the undesirable features and consider methods for eliminating them. One of the goals of this work is to carry out this task. The following four properties have been used in the step-reduced attacks.

1. The works in [4], [5], [8,9] and [12] show that 9-round local collisions obtained using XOR differentials hold with probability 2^{-39} or less. In contrast, the work of [10] shows that 9-round local collisions using additive differentials hold with probability 1/3 (and improved to probability 1 in [14]). This shows that the round function design is biased to resist XOR differentials.
2. SHA-2 design uses 8 registers a to h, where a and e registers are nonlinearly updated while the rest are simply copied. It turns out that there is a *very simple* relation by which the e-register value at Step i can be controlled using only the a-register values at Step i to $i-4$.

* This author is supported by the Ministry of Information Technology, Govt. of India.

3. The round update function for the a-register uses an invertible linear transformation Σ_0 while that of the e-register uses an invertible linear transformation Σ_1. Both Σ_0 and Σ_1 have 0 and -1 as fixed points. This considerably helps the attacks.
4. The technique of perturbation-correction [1] is used to build the attacks. A 9-step local collision is suitably placed between steps i and $i+8$ and it is ensured that all message word differences after Step $i+8$ are zero.

Based on the above four properties, we describe methods to overcome them. The linear maps Σ_0, Σ_1 are modified to affine maps Γ_0, Γ_1 to ensure that Γ_0, Γ_1 and $\Gamma_0 \oplus \Gamma_1$ do not have any fixed points (along with a few other properties). This takes care of the third point above.

By introducing simple changes to the update function of the a and e-registers, we improve the resistance to additive differentials. These changes also lead to cancelling out the simple relation between the a and e-registers. As a result, both the first and second points above are eliminated.

Recently, at the rump session of Eurocrypt '08, Yu and Wang showed the non-randomness of SHA-256 round function using a 33-step differential path. In contrast, we show that it is almost trivial to exhibit the non-randomness of any compression function $H(\mathsf{IV}, M)$ of the SHA family. As an example, we show that if M is the message "The round function of the SHA-2 family is random"; M' is the message "The round function of the SHA-2 family is not random"; IV_{std} is the IV specified in the standard, then it is easy to obtain an IV' such that $H(\mathsf{IV}_{std}, M) - \mathsf{IV}_{std} = H(\mathsf{IV}', M') - \mathsf{IV}'$. For SHA-256, we provide the corresponding IV'. If H was a true random function, then such a relation would hold with probability 2^{-256} for SHA-256 and 2^{-512} for SHA-512. In fact, if the update function of a stream cipher satisfied such a property, it would be considered broken.

We introduce a new hash function design construct called multiple feed-forward to eliminate the above easily exhibited non-randomness as also to provide additional resistance to the perturbation-correction technique mentioned above. The SHA-family design uses a single feed-forward where the IV is added to the output of composition of all the round functions. We suggest introducing several other feed-forward steps where the feed-forward is alternately provided using addition and XOR. A consequence is that if any 9-round local collision is placed within the first 16 steps, then there will be a step i within these 16 steps such that there will be a perturbation in the registers at Step i and this perturbation will necessarily extend to steps beyond the first 16 steps. Since message words beyond the first 16 steps are obtained using the message recursion, it will be very difficult to cancel out the effect of such cascaded perturbation. This significantly improves the resistance to perturbation-correction attacks. Such resistance is achieved at a marginal cost. The amortized cost of the new feed-forward steps is less than one t-bit operation (add/XOR) per step, where $t = 32$ for SHA-256 and $t = 64$ for SHA-512.

The idea of feed-forward is taken one step further. We suggest the idea of providing feed-forward across message blocks. The intuitive justification is that this provides an additional mechanism for allowing the processing of the current block to depend on earlier blocks. Concrete suggestions are given for the SHA-2 family. These improve the resistance of SHA-2 hash functions against generic multi-collision attacks introduced in [7]. For example, for SHA-256, finding 2^r messages which map to the same value requires $r \times 2^{128}$ invocations of the compression function. With our suggestion for modifying the SHA-256 hash function this increases to $r \times 2^{384}$ invocations of the compression function.

2 Overview of and Insights into Attacks Against SHA-2

Local collisions for linearized version of SHA-2 were studied by Gilbert and Handschuh [4] and by Sanadhya and Sarkar [12]. All these local collisions hold for the actual SHA-256 with probabilities of about 2^{-39} or less. Using these local collisions, Mendel et al. [8,9] and later Sanadhya and Sarkar [13]

obtained collisions for 18 step SHA-256. We call attacks using local collisions for the linearized version of SHA-2 as *linear attacks*.

Nikolić and Biryukov [10] presented a local collision which is valid for the actual SHA-256 function. The important point to note is that this local collision holds with probability of about 1/3. Using this local collision, the authors showed collisions for 21-step SHA-256 with probability about 2^{-19}. Using similar methods, Sanadhya and Sarkar [14] obtained another local collision which holds with probability 1. Extension of these attacks to 22-step collisions with probability one for both SHA-256 and SHA-512 have been given in [11]. Very recently, Indesteege et al. [6] have presented collisions for 23 and 24 step SHA-256 requiring computation effort of around 2^{18} and $2^{28.5}$ appropriately step-reduced SHA-256 hash calls. We call attacks using local collisions valid for the actual SHA-2 as *nonlinear attacks*.

We now discuss certain features of the SHA-2 design which facilitated collision attacks against reduced round versions discussed earlier.

Linear vs. Nonlinear attacks: The success probabilities of the linear attacks are very low in comparison to the nonlinear attacks. This is due to the huge difference in the success probability of the local collisions for these two types of attack. The successful collision attacks against SHA-0 [1] and SHA-1 [16] had utilized local collisions which are valid for the linearized versions of the corresponding hash functions. This indicates that the low success probability of the linear local collisions for SHA-2 could have been a design criterion. This is achieved in SHA-2 by using modular addition (modulo 2^{32} in SHA-256 and modulo 2^{64} in SHA-512) in several places in the round function. In fact, the only places where XOR addition is utilized in the SHA-2 design are in the design of the transformations Σ_0, Σ_1 and σ_0, σ_1.

Choice of the Transformations Σ_0 and Σ_1: Two transformations Σ_0 and Σ_1 are used in the round function of SHA-2. These transformations are given in Section A. We first note that all the four linear transformations are invertible.

Now consider the equations $\Sigma_0(x) = x$ and $\Sigma_1(x) = x$. Any solution to these equation will give a "fixed point" for the transformations Σ_0 and Σ_1. Since both these transformations use only XORs, we can equivalently look at the equations $(\Sigma_0 \oplus I_{32})(x) = 0$ and $(\Sigma_1 \oplus I_{32})(x) = 0$ for SHA-256, where I_{32} is the identity matrix of order 32. For SHA-512, the I_{32} needs to be replaced by I_{64}.

For SHA-256, $\Sigma_0 \oplus I_{32}$ has rank 31 but $\Sigma_1 \oplus I_{32}$ has rank 29. The null space of $\Sigma_0 \oplus I_{32}$ has basis {0xffffffff}, whereas the null space of $\Sigma_1 \oplus I_{32}$ has basis {0x99999999, 0xaaaaaaaa, 0xcccccccc}. For SHA-512, the ranks of both $\Sigma_0 \oplus I_{64}$ and $\Sigma_1 \oplus I_{64}$ are 63 and the null space has basis {0xffffffffffffffff}.

Fixed points of Σ_0 and Σ_1 for both SHA-256 and SHA-512 are shown in Table 1.

Table 1. Fixed points of Σ_0 and Σ_1 for SHA-256 and SHA-512.

Hash function	Transformation	Fixed Points
SHA-256	Σ_0	{0x00000000, 0xffffffff}
	Σ_1	{0x00000000, 0xffffffff, 0x33333333, 0x55555555,
		0x66666666, 0x99999999, 0xaaaaaaaa, 0xcccccccc}
SHA-512	Σ_0	{0x0000000000000000, 0xffffffffffffffff}
	Σ_1	{0x0000000000000000, 0xffffffffffffffff}

The analysis above shows that both Σ_0 and Σ_1 have common fixed points for all functions in the SHA-2 family. Moreover, the common fixed points have very simple structure as well: all the bits

are either zero or one when they are expressed as 32-bit (or 64-bit) quantities. The numeric value of these common fixed points is 0 and -1. This is the crucial issue which enables the high probability nonlinear local collisions utilized in recent attacks.

Choice of the Transformations σ_0 and σ_1: Two transformations σ_0 and σ_1 are used in the message expansion of SHA-2. They are given in Section A. Since both these transformations are GF(2) linear, they can be equivalently represented as matrices. We note that both σ_0 and σ_1 for SHA-256 as well as for SHA-512 are full rank matrices. Despite the fact that both these transformations have equal rank, their differential behaviour is not uniform. The transformation σ_1 is highly biased. In particular, for SHA-256, the expression $\sigma_1(x+1) - \sigma_1(x)$ takes only 6181 distinct values for all 2^{32} choices of x. In contrast, the differential range of σ_0 has much more uniform spread. This highly skewed behaviour of σ_1 was also noted in [6] recently.

Cross Dependence Equation: In the calculation of new register values at each step of the SHA-2 hash family, registers b, c and d are merely copies of register a values of previous steps. Registers e and a are also related since most of the terms in their computation are common. Thus, we note that e_i can be computed solely from the register a values as shown below.

$$\begin{aligned} e_i &= d_{i-1} + \Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i \\ &= d_{i-1} + a_i - \Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) \\ &= a_{i-4} + a_i - \Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, a_{i-2}, a_{i-3}). \end{aligned} \tag{1}$$

This relationship between these two register values, which we call the Cross Dependence Equation (CDE), implies that if the a register values for five consecutive steps are known then the e register for the last of these steps can be determined. This fact means that we can control the value of e_i from a_{i-4}, a register value which was computed 4 steps earlier. This fact can also be used to provide an alternate description of SHA-2 round function as in [6].

The SHA-0/1 design used the updation of only one register in the round function. In contrast, the SHA-2 designers chose to update two registers in each round. The CDE allows an attacker to get simple relations between a and e registers by ensuring suitable behaviour of f_{MAJ}. Note that it is rather easy to control the differential behavior of f_{MAJ} as utilized in [10] and other related works. The CDE, therefore, reduces the utility of two register updates in each round.

Local collision and message expansion: The idea of perturbation-correction from [1] is used to obtain a local collision. If a message difference (perturbation) is introduced at Step i, then it is possible to define subsequent message differences such that the perturbation is cancelled at Step $i+8$. This leads to a 9-step local collision. The idea of the NB attack and its extension for obtaining an r-round collision is the following. Choose a suitable i and place a local collision from Step i to $i+8$. Then ensure that $\delta W_j = 0$ for $j = i+9, \ldots, r-1$ leading to an r-round collision. This approach succeeds because a perturbation introduced at some step can be "quickly" cancelled within a few steps. Viewed another way, the introduced perturbation need not affect registers at Step j if j is somewhat far from i, i.e., the perturbation does not have long range effects.

2.1 Illustrations

We now illustrate the role of some of the features pointed above in recent reduced round attacks against SHA-2. Later, we suggest modifications to the SHA-2 design to take care of these weaknesses.

Role of fixed points of Σ_0, Σ_1: We provide a brief overview of some steps of the recent attack against reduced SHA-256 from [10].

The essential idea is to use two different messages such that both produce the same register value after some steps of the SHA-256 function. Let the two messages (after message expansion) be denoted by $\{W_0, W_1, \ldots W_{63}\}$ and $\{W'_0, W'_1, \ldots W'_{63}\}$. The register values corresponding to the two messages after running through r steps be denoted by $\{a_r, b_r, \ldots h_r\}$ and $\{a'_r, b'_r, \ldots h'_r\}$ respectively. Let the modular difference of two registers be denoted by $\delta p = p' - p \pmod{2^{32}}$, where p could be any of the registers used in SHA-2.

If two message words W_i and $W'_i = W_i + 1 \pmod{2^{32}}$ are run through a step of SHA-256, then the resulting register values will satisfy the following relation: $\delta a_i = \delta e_i = 1$ and $\delta b_i = \delta c_i = \delta d_i = \delta f_i = \delta g_i = \delta h_i = 0$. That is, $a'_i = a_i + 1$, $e'_i = e_i + 1$ and $b'_i = b_i$, $c'_i = c_i$, $d'_i = d_i$, $f'_i = f_i$, $g'_i = g_i$, $h'_i = h_i$.

For the next step, it is possible to choose δW_{i+1} such that $\delta a_{i+1} = 0$ and $\delta e_{i+1} = -1$. To highlight the SHA-2 design issues on which this step hinges, we briefly explain it below.

Using (3) with the values of the registers at the ith step, we get:

$$\begin{aligned}\delta a_{i+1} &= \Sigma_0(a_i + 1) - \Sigma_0(a_i) + f_{MAJ}(a_i + 1, b_i, c_i) - f_{MAJ}(a_i, b_i, c_i) + \Sigma_1(e_i + 1) - \Sigma_1(e_i) \\ &\quad + f_{IF}(e_i + 1, f_i, g_i) - f_{IF}(e_i, f_i, g_i) + \delta W_{i+1}; \\ \delta e_{i+1} &= \Sigma_1(e_i + 1) - \Sigma_1(e_i) + f_{IF}(e_i + 1, f_i, g_i) - f_{IF}(e_i, f_i, g_i) + \delta W_{i+1}.\end{aligned}$$

The f_{MAJ} terms can be cancelled by ensuring $b_i = c_i$, i.e. $a_{i-1} = a_{i-2}$. The $\delta\Sigma_0$ and $\delta\Sigma_1$ terms can be simplified by ensuring that $a_i = e_i = -1$. These choices of a_i and e_i imply that $a'_i = e'_i = 0$. Both 0 and -1 are fixed points of both Σ_0 and Σ_1, therefore $\Sigma_0(a_i + 1) - \Sigma_0(a_i) = \Sigma_1(e_i + 1) - \Sigma_1(e_i) = 1$. With these simplifications, we get:

$$\begin{aligned}\delta a_{i+1} &= 1 + 0 + 1 + f_{IF}(0, f_i, g_i) - f_{IF}(-1, f_i, g_i) + \delta W_{i+1} \\ &= 2 + g_i - f_i + \delta W_{i+1}; \\ \delta e_{i+1} &= 1 + f_{IF}(0, f_i, g_i) - f_{IF}(-1, f_i, g_i) + \delta W_{i+1} \\ &= 1 + g_i - f_i + \delta W_{i+1}.\end{aligned}$$

Now choosing $\delta W_{i+1} = -2 - g_i + f_i$, we get the desired register differences $\delta a_{i+1} = 0$ and $\delta e_{i+1} = -1$.

Similarly, further steps of the attack in [10] can be obtained. For complete details of the attack, refer to [10].

Role of Cross Dependence Equation: In [11], an algorithm is developed for attacking 22-step SHA-2 hash family deterministically. The essential idea is to set e register values by fixing the a register values of previous 5 steps suitably.

Linear vs Nonlinear attacks: The previous works on attacking reduced round SHA-256 using linear attacks [8,9], [13] could only succeed in obtaining 18-step collisions. In comparison, the attack of Nikolić and Biryukov [10], and later works [11] and [6] based on it, succeeded in obtaining 22 and 24-step collisions respectively. The success probability of these non-linear attacks is much higher than the earlier linear attacks. This shows that the SHA-2 design can resist XOR based differential attacks much more than modular addition based differential attacks.

Handling message expansion: The attack of Nikolić and Biryukov [10] used a single local collision and chose this local collisions in such a way that it produces no difference in the message words from the end of the local collision to Step 20. This way they could obtain 20-step collisions for SHA-256 with the same probability as the success probability of a single local collision. Using similar ideas

they extended their 20-step attack to 21-steps. Later, using the same technique, works [11] and [6] extended this attack up to 22 and 24-steps respectively. The central issue of these attacks is that the differing register values during the progress of the local collision do not have any role to play in the later steps.

3 Exhibiting Non-randomness of the SHA Family

We first discuss the design of SHA-0/1 and SHA-2 hash functions in a generalized form. Using this general design, we later show non-randomness of the compression functions of the entire SHA family.

3.1 Structure of the SHA Family of Hash Functions

The internal state S consists of n t-bit words, while a message block M consists of m t-bit words. The compression function $H(S, M)$ maps $(m + n)$ t-bit words to n t-bit words. The form of the compression function $H(S, M)$ is $H(S, M) = G(S, M) + S$, where $G(S, M)$ produces as output n t-bit words. The addition of $G(S, M)$ and S is the component-wise addition of n t-bit words modulo 2^t. This addition is usually called "feed-forward".

The function $G(S, M)$ has a complex nonlinear structure and consists of successive applications of r functions $G_0, \ldots, G_{r-1}$ to S. Each G_i takes as input a state S consisting of n t-bit words and another t-bit word W and produces as output n t-bit words. By an extension of notation, for $i > j$, we use $G_{i,j}$ to denote the output obtained by the successive application of $G_i \ldots, G_j$. So, $G_{i,j}$ will take as input a state S consisting of n t-bit words and $(j - i)$ t-bit words and produce as output n t-bit words.

A message block M consists of t-bit words $M_0, \ldots, M_{m-1}$. A recurrence relation $\mathcal{R}$ expands M to r t-bit words $W_0, \ldots, W_{r-1}$ in the following manner: $W_i = M_i$, for $0 \leq i \leq m - 1$; and $W_i = \mathcal{R}(W_{i-m}, \ldots, W_{i-1})$ for $m \leq i \leq r - 1$.

The value of $G(S, M)$ is defined as follows: $S^{(0)} = G_0(S, W_0)$; and for $0 < i \leq r - 1$, $S^{(i)} = G_i(S_{(i-1)}, W_i)$. The output of $G(S, M)$ is defined to be $S^{(r-1)}$. By our earlier notation, $G(S, M) = G_{0,r-1}(S, W_0, \ldots, W_{r-1})$.

3.2 Non-Randomness of SHA Family

Using the notation defined above, we now exhibit non-randomness of the SHA family compression functions. We use the following fact about the SHA family.

Fact 1. Each G_i is efficiently invertible, i.e., given n t-bit words S and a t-bit word W, it is easy to obtain n t-bit words T, such that $S = G_i(T, W)$. (Further, this T is unique; though we do not require the uniqueness in our analysis.)

Theorem 1. *Given any two message blocks M, M'; any IV; and any $T = (T_0, \ldots, T_{n-1})$ where the T_is are t-bit words, it is possible to deterministically construct an IV', such that $H(\mathsf{IV}, M) + T = H(\mathsf{IV}', M') - \mathsf{IV}'$. The computation cost is approximately two invocations of H.*

Proof. Given IV and M, compute $H(\mathsf{IV}, M)$ and let $S = T + H(\mathsf{IV}, M)$. From M', use the message recursion $\mathcal{R}$ to obtain r t-bit words $W'_0, \ldots, W'_{r-1}$.

Set $(S^{(r-1)})' = S$. Now use Fact 1 in the following manner. Given $(S^{(r-1)})'$ and W'_{r-1} invert G_{r-1} to obtain $(S^{(r-2)})'$ such that $(S^{(r-1)})' = G_{r-1}((S^{(r-2)})', W'_{r-1})$. This inverts G for one step. Now use Fact 1 repeatedly to obtain $(S^{(0)})', \ldots, (S^{(r-2)})', (S^{(r-1)})'$ such that $(S^{(i)})' = G_i((S^{(i-1)})', W'_i)$ for $1 \leq i \leq r - 1$. Finally, use Fact 1 once more to obtain IV' such that $(S^{(0)})' = G_0(\mathsf{IV}', W'_0)$. This is our required IV'.

We now have $G(\mathsf{IV}', M') = S = T + H(\mathsf{IV}, M)$. Recall that $H(\mathsf{IV}', M') = G(\mathsf{IV}', M') + \mathsf{IV}'$ from which the first part of the result follows.

One invocation of H is required to compute $H(\mathsf{IV}, M)$. The cost of carrying out the other part of the computation is to expand the message once and to invert each of the G_is. For the SHA family, inverting a G_i is as efficient as computing it in the forward direction. Hence, the computation cost is one invocation of G which is approximately equal to one invocation of H. □

By choosing T appropriately, we obtain different variations.

1. If $T = (0^{32}, \ldots, 0^{32})$, then $H(\mathsf{IV}, M) = H(\mathsf{IV}', M') - \mathsf{IV}'$.
2. If $T = -H(\mathsf{IV}, M)$, then $H(\mathsf{IV}', M') = \mathsf{IV}'$. In other words, this means that given any message M' it is possible to efficiently obtain with probability one, an IV' such that $H(\mathsf{IV}', M') = \mathsf{IV}'$.
3. If $T = -\mathsf{IV}$, then $H(\mathsf{IV}, M) - H(\mathsf{IV}', M') = \mathsf{IV} - \mathsf{IV}'$.

Note. In contrast to Point 2 above, if H were a random function, then given M, the probability of obtaining an IV such that $H(\mathsf{IV}, M) = \mathsf{IV}$ would be $1/2^{nt}$. This shows the nonrandom behaviour of the SHA-family compression functions. Similarly, for the other two points also, if we consider $H()$ to be a random function, then the probability that these hold is $1/2^{nt}$.

Yu-Wang non-randomness: At the rump session of Eurocrypt 2008, Yu and Wang showed the non-randomness of the compression function up to 39-step SHA-256 using a 33-step differential path. From our result it can be seen that if the intention is only to show non-random behaviour of the compression function, then this can be easily done. Of course, a differential path maybe more useful since it has potential applications to finding collisions.

In Table 2, we show an example message pair and an IV pair for SHA-256 such that $H(\mathsf{IV}_{std}, M_1) - \mathsf{IV}_{std} = H(\mathsf{IV}', M_2) - \mathsf{IV}'$. The two messages and the first IV are specified a-priori and IV' is obtained deterministically. Similar examples can be constructed for SHA-512 and SHA-0/1 as well.

Table 2. Example showing the non-randomness of the round function of SHA-256. The two messages M_1 and M_2 with the initial register values IV_1 and IV_2 satisfy the relation $H(M_1, \mathsf{IV}_1) - \mathsf{IV}_1 = H(M_2, \mathsf{IV}_2) - \mathsf{IV}_2$. In this example, IV_1 is the standard IV for SHA-256. Given any M_1, M_2 and IV_1, the IV_2 can be computed deterministically.

M_1	The round function of the SHA-2 family is random.
IV_1	6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
H_1	99767f0a 3659a88f 96c8b0bd 566bb6d7 df45fb3f 8daf752b 594ad309 858ba623
$H_1 - \mathsf{IV}_1$	2f6c98a3 7af1fa0a 5a59bd4b b11bc19d 8e37a8c0 f2aa0c9f 39c6f95e 29aad90a
M_2	The round function of the SHA-2 family is not random.
IV_2	d44068db e5bac254 bfa4cc74 48edbcce b5eb7040 9b83ebae e7c6d942 452ea6c9
H_2	03ad017e 60acbc5e 19fe89bf fa097e6b 44231900 8e2df84d 218dd2a0 6ed97fd3
$H_2 - \mathsf{IV}_2$	2f6c98a3 7af1fa0a 5a59bd4b b11bc19d 8e37a8c0 f2aa0c9f 39c6f95e 29aad90a

3.3 Comments on the exhibited Non-randomness

1. The property that we have shown does not affect any of the conventional security notions of a hash function. Namely, it does not affect collision resistance, pre-image resistance or pseudo versions of both these properties.

2. If the compression function was defined to be $G(\mathsf{IV}, M)$ instead of $H(\mathsf{IV}, M) = G(\mathsf{IV}, M) + \mathsf{IV}$, then our technique would have exhibited pseudo-collisions for the compression function. In other words, resistance against pseudo-collisions is obtained due to the feed-forward, i.e., the addition of IV, and not by the nonlinear complexities of the round function of G. We do not know whether this is an intended property or a design oversight.
3. A hash function may be used in many situations and in many ways which a designer may not even anticipate. Our result provides a caution against such indiscriminate use of the SHA family.
4. One common use of a hash function is to produce a "random looking string" or to "kill off algebraic structure or patterns" by hashing a string having more structure. Our result show that the SHA-family compression functions do not behave as "randomly" as one might expect.

Does our technique apply to any other hash function? If Fact 1 holds for the hash function in question, then the technique indeed applies. In particular, it is applicable for SHA-0, SHA-1 and the entire SHA-2 family. However, there are designs for which the technique will not work. Examples are RIPEMD-160 and Rumba20. In the former, there are two parallel strands of computations which are then combined at the end. For our technique to work, it would have to work in a synchronized manner on both the strands. This does not seem to be simple. For Rumba20, the output is the XOR of four invocations of Salsa20. Again, it is not clear how to apply the present technique to this design.

The problem with the SHA design. Given M and the final output S of G, it is easy to invert G to obtain an IV, such that $G(\mathsf{IV}, M) = S$. In other words, for every fixed M, $G(\ .\ , M)$ is an easily invertible function. We do not see any intuitive reason why a compression function should satisfy such a property or even why such a feature should be considered desirable. As mentioned above, such easy invertibility is not a feature of RIPEMD-160 or Rumba20. It may not be present in other hash functions also; we have not looked at all the known designs.

4 Improving the SHA-2 design

4.1 Improving Affine Transformations in the Update Function

We suggest that the linear functions Σ_0, Σ_1 be replaced by affine functions Γ_0 and Γ_1 respectively such that the following conditions are satisfied.

1. For all $x \in \{0,1\}^t$, $\Gamma_i(x) \neq x$ or $\overline{x}$.
2. For no $x \in \{0,1\}^t$, $\Gamma_0(x) \neq \Gamma_1(x)$ or $\overline{\Gamma_1(x)}$.

The first property is similar to one of the design criteria for the AES S-box [2].

Achieving the above properties requires a bit of linear algebra. Suppose, we define $\Gamma_i(x) = \Sigma_i(x) + \mathbf{b}_i$, where $\mathbf{b}_0, \mathbf{b}_1$ are to be chosen such that the above two conditions are satisfied. Additionally, we would like both $\mathbf{b}_0$ and $\mathbf{b}_1$ as well as $\mathbf{b}_0 \oplus \mathbf{b}_1$ are all either balanced or nearly balanced.

Consider the first point for SHA-256: $\Gamma_i(x) = x$ for some x implies $(\Sigma_i \oplus I_{32})x = \mathbf{b}_i$ and $\Gamma_i(x) = \overline{x}$ for some x implies $\Gamma_i(x) = (\Sigma_i \oplus I_{32})x = \overline{\mathbf{b}_i}$. In other words, the first point holds if both $\mathbf{b}_i$ and $\overline{\mathbf{b}_i}$ are not in the column space of $(\Sigma_i \oplus I_{32})$. In a similar manner, it can be shown that the second point holds if both $\mathbf{b}_0 \oplus \mathbf{b}_1$ and $\overline{\mathbf{b}_0 \oplus \mathbf{b}_1}$ are not in the column space of $(\Sigma_0 \oplus \Sigma_1)$.

We computed many choices of $\mathbf{b}_0$ and $\mathbf{b}_1$ satisfying these constraints. Examples for SHA-256 are

$$\mathbf{b}_0 = \texttt{0xdcb2344c} \text{ and } \mathbf{b}_1 = \texttt{0x9b097671}.$$

For SHA-512, examples are

$$\mathbf{b}_0 = \texttt{0x1762e66a04d6be32} \text{ and } \mathbf{b}_1 = \texttt{0x12135c7549e2fcdd}.$$

4.2 Mix of + and $\oplus$

For the compression function of SHA-2, it is possible to obtain a 9-round local collision using modular differentials with probability one while using XOR differentials the probability of 9-round local collision is at most 2^{-39}. This shows that the round function design is heavily biased towards protection against XOR differentials. A better mix of + and $\oplus$ can be obtained by using + to add W_i to obtain a_i and using $\oplus$ to XOR W_i to obtain e_i. This will mean that if we work with XOR differentials, then it will be difficult to analyze the XOR differentials of the a-register; on the other hand, if we work with modular differentials, then it will be difficult to analyze the modular differentials of the e-register.

The second property of the compression function is that it is easy to fix $a_{i-4}, \ldots, a_i$ (using words $W_{i-4}, \ldots, W_i$) to simple values and ensure that e_i is also fixed to a simple value. This is because of the CDE (1). An example of the simplification possible is having $a_{i-1} = a_{i-2} = 0$. In this case, $e_i = a_i + a_{i-4}$, which is a rather simple relation.

To remove the above two issues, we suggest the update functions to be the following.

$$
\begin{aligned}
a_i &= h_{i-1} \oplus (\Gamma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) + \Gamma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + K_i + W_i)\\
b_i &= a_{i-1}\\
c_i &= b_{i-1}\\
d_i &= c_{i-1}\\
e_i &= (d_{i-1} + \Gamma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i) \oplus W_i\\
f_i &= e_{i-1}\\
g_i &= f_{i-1}\\
h_i &= g_{i-1}.
\end{aligned}
$$

Note that the term $T_{i-1} = \Gamma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + K_i$ is present in the computation of both a_i and e_i. This common sub-expression need to be computed only once for each step. In the SHA-2 compression function, the term $\Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i$ is common to both a_i and e_i. Our new definition and the SHA-2 definition have similar computational complexity. Using this, we have

$$
\begin{aligned}
T_{i-1} &= (a_i \oplus h_{i-1}) - (\Gamma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) - W_i)\\
&= (e_i \oplus W_i) - (d_{i-1} + h_{i-1}).
\end{aligned}
$$

Consequently,

$$
e_i = W_i \oplus ((a_i \oplus h_{i-1}) + (d_{i-1} + h_{i-1}) - (\Gamma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) - W_i)) \tag{2}
$$

The dependences on both W_i and h_{i-1} do not directly cancel out. As a result, it is not possible to express e_i solely in terms of a_{i-4} to a_i. Going back to the previous example, if $a_{i-1} = a_{i-2} = 0$ (note $b_{i-1} = a_{i-2}$) then $e_i = W_i \oplus ((a_i \oplus h_{i-1}) + (d_{i-1} + h_{i-1} - W_i - \mathbf{b}))$ which has two alternations of + and $\oplus$ and is a more complicated relation compared to the simple $e_i = a_i + a_{i-4}$ that is obtained for the SHA-2 compression function.

5 Multiple Feed Forward: A New Design Construct

We have noted that resistance against pseudo-collisions of the compression function is provided by the feed-forward. This has led us to explore the effect of feed-forward further. We introduce the idea of multiple feed-forward, i.e., feed-forward at several places. For one thing, this prevents the trivial exhibition of the non-randomness behaviour mentioned earlier. Second, it provides an additional layer of resistance against the perturbation-correction technique. These issues are discussed later.

The computation starts with a C (the initial value of which is IV) and suppose that the output of step i is $S^{(i)}$. The SHA-2 compression function employs a single feed-forward, i.e., the output of H is $C + S^{(N-1)}$ where $N = 64$ for SHA-256 and $N = 80$ for SHA-512.

Let $t_1 = 7$, $t_2 = 14$ and $s = 16$. We introduce new feed-forwards at steps $t_1 + s \times k_1$ and $t_2 + s \times k_2$, where k_1, k_2 are positive integers. Let $S^{(-1)} = C$. We use two sets of temporary registers T_1 and T_2, i.e., each T_i consists of $n = 8$ t-bit words, where $t = 32$ for SHA-256 and $t = 64$ for SHA-512.

The algorithm for the compression function is shown in Figure 2. In the figure assume for the moment that in the input $T_1 = T_2 = 0^{nt}$ and the output does not contain T_1 and T_2. The general description allows the introduction of feed-forward across message blocks as explained in the next section.

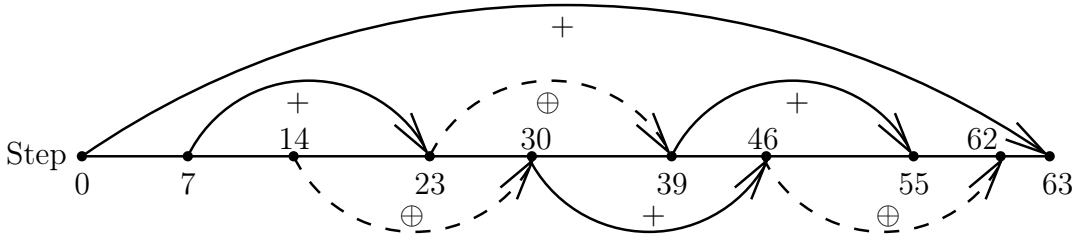
The following points are to be noted.

1. The feed-forward with the C after N steps remains unchanged.
2. There are two feed-forward threads starting at t_1 and t_2. Both threads alternate the type of feed-forward between $+$ and $\oplus$. The first line starts with $+$, while the second line starts with $\oplus$.
3. The choice of s ensures that the sets $\{t_1 + s \times k_1 : k_1 \geq 1\}$ and $\{t_2 + s \times k_2 : k_2 \geq 1\}$ are disjoint. So, at no step will two feed-forward threads starting at t_1 and t_2 meet.
4. New feed-forward is not introduced at Step $(N-1)$. Again, the choices of t_1, t_2 and s ensure this.
5. The amortized cost of the new feed-forwards is less than one t-bit operation (addition or XOR) per step, where t is 32 for SHA-256 and 64 for SHA-512.

The choice of t_1 and t_2 ensures that if we take any interval of length 9 in the range 0 to 15, then at least one of t_1 or t_2 will lie properly within the interval. In an efficient implementation with loop unrolling, the conditional statements will not be required. The temporary registers T_1 and T_2 will simply be added or XORed to the internal registers at the appropriate step.

For SHA-256, the line starting at $t_1 = 7$ introduces feed-forwards at Steps 23, 39 and 55 and the line starting at $t_2 = 14$ introduces feed-forwards at Steps 30, 46 and 62. Additive feed-forward is used at Steps 23 and 55 due to the first line and at Step 46 due to the second line. XOR feed-forward is used at Step 39 due to the first line and at Steps 30 and 62 due to the second line. There are a total of 6 feed-forward steps requiring a total of $6 \times 8 = 48$ 32-bit add/XOR operations. Amortized over the 64 steps, the cost is 3/4th 32-bit operation per step. The feed-forwards for SHA-256 are illustrated in Figure 1.

Fig. 1. Illustration of multiple feed-forward for SHA-256. There are two threads of feed-forwards, based on XOR and modular addition. The original additive feed-forward from Step 0 to Step 63 is also retained.



For SHA-512, the feed-forwards of the first line occur at Steps 23, 39, 55 and 71 with additive feed-forwards at Steps 23 and 55 and XOR feed-forwards at Steps 39 and 71. The second line of feed-forwards is introduced at Steps 30, 46, 62 and 78, with XOR at Steps 30 and 62 and additive

at Steps 46 and 78. Amortized over the 80 steps, the cost is 4/5th 64-bit operation (add/XOR) per step.

The idea of multiple feed-forward is generic and can be used with other hash designs. Depending on the number of registers, the number of free words and the total number of steps N, the value of t_1, t_2 and s has to be chosen appropriately so as to ensure the properties described above.

Non-random behaviour: The first thing to note is that this makes the function G difficult to invert and removes the trivially non-random behaviour discussed earlier. Similar effect can be achieved using two parallel threads with separate recursions for message expansion. (This has been done in RIPEMD-160.) However, when executed sequentially, the time required would be doubled (unless the number of rounds is reduced). In comparison, the increase in computation cost is only marginal in the case of multiple feed-forward – less than one t-bit operation per step. Also, the change to the compression function design is minimal.

Resistance to perturbation-correction attacks: The basic idea of such an attack has been discussed earlier. By our choice of t_1 and t_2, at least one of these will be properly contained in any 9-step local collision. Consequently, the registers at the corresponding step will have a difference. Due to feed-forwards, this difference will be pushed out into the steps where the message words are defined through message recursion. Since these words cannot be directly controlled, it will be difficult to cancel out the introduced perturbation at these steps. The overall effect will be that the local collision based perturbation-correction attack will become substantially more difficult to apply.

6 Feed-Forward Across Message Blocks

The idea of feed-forward can be extended to different message blocks. Let us go back to the construction in Section 5. After the first message block has been processed, three quantities are produced, the chaining variable C and the quantities T_1 and T_2. But, T_1 and T_2 are not used further. These two quantities are "lightweight" digests of the first message block. We suggest that these be used in the processing of the second message block.

The processing of the second block starts with C as the IV and by the construction of Section 5, the output of Step 7 is taken to be new T_1 and the output of Step 14 is taken to be the new T_2. We modify this as follows. The new T_1 is the XOR of the old T_1 and the output of Step 7; and the new T_2 is the sum of the old T_2 and the output of Step 14. The rest of the two feed-forward threads are as before. Figure 2 shows the description.

6.1 Merkle-Damgård (MD) Iteration

The modification can still be considered to be within the MD framework. Let $M^{(0)}, \ldots, M^{(\ell)}$ be the message blocks (including padding with length). Let $C^{(-1)} = \mathsf{IV}$ and $T_1^{(-1)} = T_2^{(-1)} = 0^{nt}$. There are ℓ invocations of the compression function, where the $(i+1)$st invocation of the compression function takes $(C^{(i)}, T_1^{(i)}, T_2^{(i)}, M^{(i+1)})$ as input and produces a $(C^{(i+1)}, T_1^{(i+1)}, T_2^{(i+1)})$ as output. Here $C^{(i)}, C^{(i+1)}$ are the chaining variables; and $T_1^{(i)}, T_2^{(i)}, T_1^{(i+1)}, T_2^{(i+1)}$ are the feed-forward quantities.

Viewed in this way, it is easy to prove as usual by backward induction that a collision for the hash function leads to a collision for the compression function. Finding a pseudo collision for the compression function defined in this manner may be easier to find than the compression function where there is no feed-forward across message blocks. This is because one may choose T_1 and T_2 to suitable values. However, the hashing starts with $C^{(-1)} = \mathsf{IV}$ and $T_1^{(-1)} = T_2^{(-1)} = 0^{nt}$, which fixes the initial choice of T_1 and T_2. Hence, finding actual collisions for the new hash function is no easier than finding collisions without feed-forward across message blocks.

Fig. 2. Modified compression function with two feed-forward threads. Here $t_1 = 7$, $t_2 = 14$ and $s = 16$; G_i is the step function.

Input: C, T_1, T_2, $W_0, \ldots, W_{N-1}$.
1. for $i = 0, \ldots, 15$ do
2. $\quad S^{(i)} \leftarrow G_i(S^{(i-1)}, W_i)$;
3. $\quad$ if $(i = 7)$ then $T_1 \leftarrow T_1 \oplus S^{(i)}$; if $(i = 14)$ then $T_2 \leftarrow T_2 + S^{(i)}$;
4. end for;
5. for $i = 16, \ldots, N-1$ do
6. $\quad S^{(i)} = G_i(S^{(i-1)}, W_i)$;
7. $\quad$ if $((i - t_1) \bmod s = 0)$ then $S^{(i)} \leftarrow S^{(i)} + T_1$; $T_1 = S^{(i)}$;
8. $\quad$ if $((i - t_1) \bmod 2s = 0)$ then $S^{(i)} \leftarrow S^{(i)} \oplus T_1$; $T_1 = S^{(i)}$;
9. $\quad$ if $((i - t_2) \bmod s = 0)$ then $S^{(i)} \leftarrow S^{(i)} \oplus T_2$; $T_2 = S^{(i)}$;
10. $\quad$ if $((i - t_2) \bmod 2s = 0)$ then $S^{(i)} \leftarrow S^{(i)} + T_2$; $T_2 = S^{(i)}$;
11. end for;
12. output $(S^{(N-1)} + C, T_1, T_2)$.

6.2 Multi-Collision Attacks

Let us now consider the effect of multi-collision attacks [7]. This is a generic technique which applies to the MD type construction. Using r invocations of generic collision finding algorithm on the compression function, it is possible to construct 2^r messages which map to the same value. Suppose M_1, M_1' are two distinct message blocks which (starting from IV) map to the same value C_1; and M_2, M_2' are two distinct message blocks which starting from C_1 map to the same value C_2. Then the four message (M_1, M_2), (M_1, M_2'), (M_1', M_2) and (M_1', M_2') map to the same value C_2. Since the output of the compression function consists of nt bits, the generic algorithm will require $2 \times 2^{nt/2}$ invocations of the compression function to find a collision.

Suppose we apply this to the new construction. The output of the compression function is the chaining variable C as well as T_1 and T_2. Then we need distinct M_1, M_1' which starting from IV and $T_1 = T_2 = 0^{nt}$ map to same value $C^{(1)}, T_1^{(1)}, T_2^{(1)}$; and distinct M_2, M_2' which starting from $C^{(1)}, T_1^{(1)}, T_2^{(1)}$ map to the same value $C^{(2)}, T_1^{(2)}, T_2^{(2)}$. Then as before, we will have four messages which map to the same value $C^{(2)}, T_1^{(2)}, T_2^{(2)}$. The advantage here is that the generic collision finding algorithm now needs to be applied to a compression function whose output is $3nt$ bits. As a result, the generic algorithm will require $2 \times 2^{3nt/2}$ invocations of the new compression function to find a collision. This is better than $2 \times 2^{nt/2}$ invocations required in the usual case. The number $2^{3nt/2}$ of invocations can be increased by using more suitable feed-forward threads.

On the other hand, suppose that M_1, M_1' are such that starting from IV and $T_1 = T_2 = 0^{nt}$, they produce the same value for $C^{(1)}$ but not necessarily the same value for $T_1^{(1)}, T_2^{(1)}$, i.e., $(T_1^{(1)}, T_2^{(1)}) \neq ((T_1^{(1)})', (T_2^{(1)})')$. Further, suppose that M_2, M_2' are such that M_2 starting from $C^{(1)}$ and $T_1^{(1)}, T_2^{(1)}$ produces the same $C^{(2)}$ that M_2' starting from $C^{(1)}$ and $(T_1^{(1)})', (T_2^{(1)})'$ produces. This results in single two-block collision between (M_1, M_1') and (M_2, M_2'). But, now (M_1, M_2) does not produce the same value as (M_1, M_2'). More generally no two of the four possible messages produce the same value. This is due to the difference in the intermediate feed-forward values, i.e., $(T_1^{(1)}, T_2^{(1)}) \neq ((T_1^{(1)})', (T_2^{(1)})')$.

To summarize, extending feed-forward across message blocks quantifiably increases the resistance to generic multi-collision attacks. Using two feed-forward threads, the cost of finding 2^r multi-collisions is $r \times 2^{3nt/2}$ invocations of the compression function. Without feed-forward, the requirement is $r \times 2^{nt/2}$ invocations. More generally, using κ suitable feed-forward threads will make the cost of finding 2^r multi-collisions equal to $r \times 2^{(\kappa+1)nt/2}$ invocations of the compression function.

7 Conclusions

In this work, we studied several properties of the SHA-2 design which were used in recent collision attacks against reduced SHA-2. We have suggested modifications to the SHA-2 design so as to make these attacks inapplicable. The modified SHA-2 design is almost as efficient as the original one. We showed a property of the round functions of the SHA family which causes non-randomness in the entire SHA family.

We provided a generic construction of multiple feed-forward. This can be used to strengthen a design against perturbation-correction collision finding attacks. Further, the idea of feed-forward over several message blocks is suggested and shown to provide quantifiable better resistance to multi-collision attacks.

Hopefully, this work will lead to better understanding of the SHA designs and lead to improved construction of next generation hash functions.

References

1. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO 1998, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.
2. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
3. Federal Register Vol. 72, No. 212. *Announcing Request for Candidate Algorithm Nominations for a new Cryptographic Hash Algorithm (SHA-3) Family*. U.S. Department of Commerce, National Institute of Standards and Technology(NIST), November 2, 2007. Available at `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
4. Henri Gilbert and Helena Handschuh. Security Analysis of SHA-256 and Sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2003.
5. Philip Hawkes, Michael Paddon, and Gregory G. Rose. On Corrective Patterns for the SHA-2 Family. *Cryptology eprint Archive*, August 2004. Available at `http://eprint.iacr.org/2004/207`.
6. Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Christian Rechberger. Collisions and other Non-Random Properties for Step-Reduced SHA-256. *Cryptology eprint Archive*, April 2008. Available at `http://eprint.iacr.org/2008/131`.
7. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
8. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of Step-Reduced SHA-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2006.
9. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of Step-Reduced SHA-256. *Cryptology eprint Archive*, March 2008. Available at `http://eprint.iacr.org/2008/130`.
10. Ivica Nikolić and Alex Biryukov. Collisions for Step-Reduced SHA-256. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, March 26-28, 2008*, volume Pre-proceedings version of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.
11. Somitra Kumar Sanadhya and Palash Sarkar. Collision attacks against 22-step SHA-512. Communicated.
12. Somitra Kumar Sanadhya and Palash Sarkar. New Local Collisions for the SHA-2 Hash Family. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2007.
13. Somitra Kumar Sanadhya and Palash Sarkar. Attacking Reduced Round SHA-256. In Steven Bellovin and Rosario Gennaro, editors, *Applied Cryptography and Network Security - ACNS 2008, 6th International Conference, New York, NY, June 03-06, 2008, Proceedings*, volume 5037 of *Lecture Notes in Computer Science*. Springer, 2008.

14. Somitra Kumar Sanadhya and Palash Sarkar. Non-Linear Reduced Round Attacks Against SHA-2 Hash family. In Yi Mu and Willy Susilo, editors, *Information Security and Privacy - ACISP 2008, The 13th Australasian Conference, Wollongong, Australia, 7-9 July 2008, Proceedings*, volume To appear of *Lecture Notes in Computer Science*. Springer, 2008.
15. Secure Hash Standard. *Federal Information Processing Standard Publication 180-2.* U.S. Department of Commerce, National Institute of Standards and Technology(NIST), 2002. Available at `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`.
16. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

A The SHA-2 Hash Family

The newest members of SHA family of hash functions were standardized by US NIST in 2002 [15]. There are 2 differently designed functions in this standard: the SHA-256 and SHA-512. In addition, the standard also specifies their truncated versions as SHA-224 and SHA-384. The number in the name of the hash function refers to the length of message digest produced by that function. Next we describe SHA-256 and SHA-512 in detail.

Eight registers are used in the evaluation of SHA-2. The initial values in the registers are specified by an $8 \times n$ bit IV, n=32 for SHA-256 and 64 for SHA-512. In Step i, the 8 registers are updated from (a_{i-1}, b_{i-1}, c_{i-1}, d_{i-1}, e_{i-1}, f_{i-1}, g_{i-1}, h_{i-1}) to (a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i) according to the following equation:

$$\left.\begin{aligned} a_i &= \Sigma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) + \Sigma_1(e_{i-1}) \\ &\quad + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i \\ b_i &= a_{i-1} \\ c_i &= b_{i-1} \\ d_i &= c_{i-1} \\ e_i &= d_{i-1} + \Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) \\ &\quad + h_{i-1} + K_i + W_i \\ f_i &= e_{i-1} \\ g_i &= f_{i-1} \\ h_i &= g_{i-1} \end{aligned}\right\} \tag{3}$$

The f_{IF} and the f_{MAJ} are three variable boolean functions defined as:

$$\begin{aligned} f_{IF}(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z), \\ f_{MAJ}(x, y, z) &= (x \wedge y) \oplus (y \wedge z) \oplus (z \wedge x). \end{aligned}$$

For SHA-256, the functions Σ_0 and Σ_1 are defined as:

$$\begin{aligned} \Sigma_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x), \\ \Sigma_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x). \end{aligned}$$

For SHA-512, the corresponding functions are:

$$\begin{aligned} \Sigma_0(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x), \\ \Sigma_1(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x). \end{aligned}$$

Round i uses a t-bit word W_i which is derived from the message and a constant word K_i. There are 64 steps in SHA-256 and 80 in SHA-512. The hash function operates on a 512-bit (resp. 1024-bit)

message specified as 16 words of 32 (resp. 64) bits for SHA-256 (resp. SHA-512). Given the message words m_0, m_1, $\ldots m_{15}$, the W_i 's are computed using the Equation:

$$W_i = \begin{cases} m_i & \text{for } 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \leq i \leq 63 \text{ (or 80)} \end{cases} \tag{4}$$

For SHA-256, the functions σ_0 and σ_1 are defined as:

$$\begin{aligned} \sigma_0(x) &= ROTR^7(x) \ \oplus ROTR^{18}(x) \oplus SHR^3(x), \\ \sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x). \end{aligned}$$

And for SHA-512, they are defined as:

$$\begin{aligned} \sigma_0(x) &= ROTR^1(x) \ \oplus ROTR^8(x) \ \oplus SHR^7(x), \\ \sigma_1(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x). \end{aligned}$$

The $\mathsf{IV} = (a_{-1}, b_{-1}, c_{-1}, d_{-1}, e_{-1}, f_{-1}, g_{-1}, h_{-1})$ is defined differently for SHA-224, SHA-256, SHA-384 and SHA-512. For details, see [15].

The output hash value of a one block (512-bit for SHA-256 and 1024-bit for SHA-512) message is obtained by chaining the IV with the register values at the end of the final round. A similar strategy is used for multi-block messages, where the IV for next block is taken as the hash output of the previous block.