Differential Collisions: an Explanation for SHA-1

Florent Chabaud   Antoine Joux

Centre d'Électronique de l'Armement

CASSI/SCY/EC

F-35998 Rennes Armées

France

chabaud@celar.fr   joux@celar.fr

April 29, 1998

Abstract

In this paper, we present a method for finding collisions in hashing functions which is apparented to linear and differential cryptanalysis of block ciphers. We apply this method on the SHA-0 function and we obtain a theoretical attack on the compression function with complexity 2^{71}, which is thus better than the birthday paradox attack. In the case of SHA-1, this method is unable to find collisions faster than the birthday paradox. This is a strong evidence that the transition to version 1 indeed raised the level of security of SHA.

Differential Collisions: an Explanation for SHA-1

April 29, 1998

Abstract

In this paper, we present a method for finding collisions in hashing functions which is apparented to linear and differential cryptanalysis of block ciphers. We apply this method on the SHA-0 function and we obtain a theoretical attack on the compression function with complexity 2^{71}, which is thus better than the birthday paradox attack. In the case of SHA-1, this method is unable to find collisions faster than the birthday paradox. This is a strong evidence that the transition to version 1 indeed raised the level of security of SHA.

1 Description of SHA

1.1 Historical overview

The secure hash standard (SHS) [SHA-0] was issued by the National Institute of Standards and Technology in 1993. It was largely inspired from Rivest's proposition MD4 [5]. However, a certain number of basic blocks of this function were different from MD4 ones, but no explanation was given for the choices. Two years later, an addendum was made to the standard, slightly altering the function [SHA-1]. This change was claimed to correct a technical weakness in SHA but no justification was given. Yet, the rumour was that a collision attack better than the birthday paradox had been found by NSA.

Independantly, several attacks on the original MD4 function, and its MD5 improvement [6] have been published [2, 4]. However, these attacks couldn't be applied to the Secure Hash Algorithm (SHA) because of the expansion used.

1.2 Generic architecture

The basic architecture of SHA is described in figure 1. The expansion box is a linear mapping from $\mathbb{F}_2^{512}$ to $\mathbb{F}_2^{2560}$. This linear mapping is the only

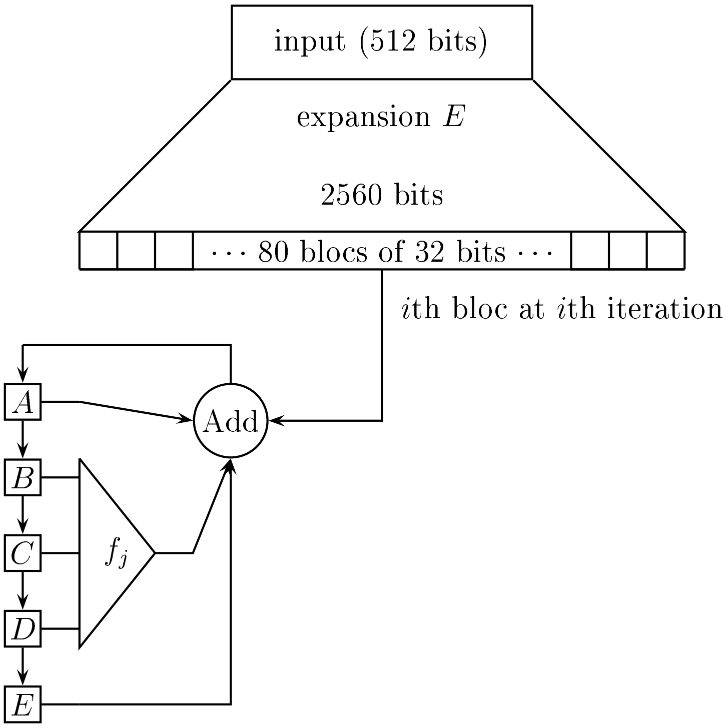


Figure 1: `SHA` architecture (rotations omitted)

Round	Function	bit definition
0	IF	$(x \wedge y) \vee (\not x \wedge z)$
1	XOR	$(x \oplus y \oplus z)$
2	MAJ	$(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
3	XOR	$(x \oplus y \oplus z)$

Table 1: `SHA` function $f_i(x, y, z)$

difference between first and second version of `SHA`. We will denote E_0 the initial expansion of [SHA-0], and E_1 the modified expansion in [SHA-1].

The function f_i is set according to the table 1, where a round counts for 20 steps of `SHA`. Symbol $\vee$ stands for the bitwise *or* function, symbol $\wedge$ stands for *and*, and $\oplus$ is the *exclusive-or* function.

2 Local propagation of perturbation

2.1 Weakened `SHA`

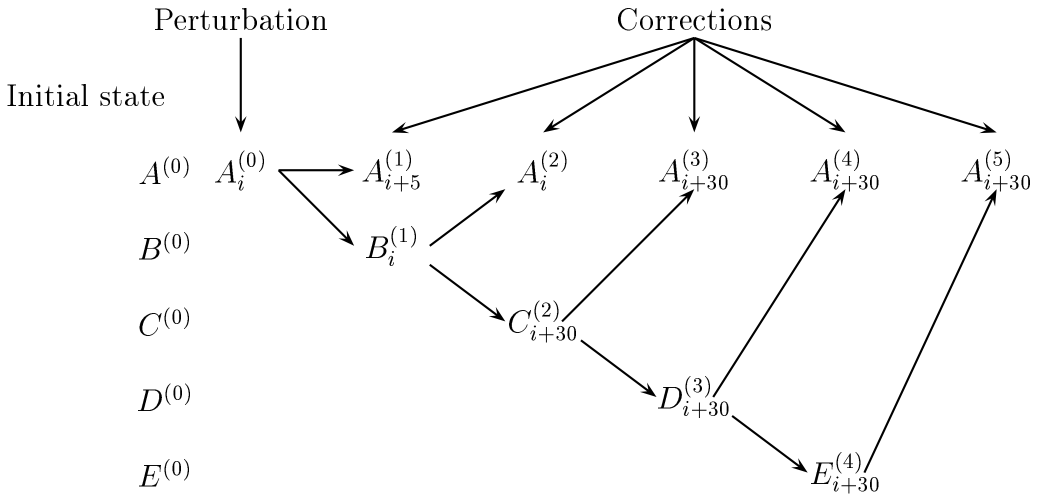
2.1.1 Considering the architecture

We will first describe the propagation of a perturbation provided by the overall architecture of `SHA`. We denote by `SHI1`[1] the compression function deduced from `SHA-0` by keeping all rotations on blocks, but replacing the *Add* function by a *Xor* one. We also consider that the function f_j is always a *Xor* function.

We denote as usual by W_i the ith bloc of the expansion ($0 \leq i < 80$), and the 32 bits of this bloc are numbered $W_{i,0}, \ldots, W_{i,31}$. We assume these blocs to have values determined by an output of the expansion, and we now try to introduce a single bit error on bit 1 (we will explain this choice later on) of the ith bloc. However, we want this perturbation to be fully controlled, and to disappear as soon as possible. This can be done in `SHI1` by introducing five other perturbations on some bits of the $i+1$th through $i+5$th iterations (see figure 2).

As all the functions of `SHI1` are linear, this means that a differential mask with 6 bits can be applied to the output of the expansion box without change

[1] `SHI1` is a french pun involving cats and dogs.



Subscripts point the perturbated bit of the state.

Superscripts indicates the number of iterations elapsed since perturbation.

Figure 2: `SHI1` propagation of perturbation

on the final state of the function. However, this cannot be done directly, but only through the expansion. As this expansion is linear, that means that one has to find a word of this expansion that can be built using our elementary masks of 6 bits (1 perturbation bit plus 5 correction bits). Such a word will give us an input mask that can be xored without modification of the `SHI1` hash value.

As the expansion E_0 do not interleave bits, we can split the expansion in 32 identical boxes e_0 expanding 16 bits to 80 bits. Besides, E_0 is invariant by translation, so that we can in fact deduce the 5 correction masks from the perturbative mask. The constraints are that no pertubation may appear after the 75th round, and that no correction may be implicitely induced by sliding before the first pertubation. Hence, a word of e_0 can be used as perturbative mask if it ends with 5 zeroes and is preceded by 5 zeroes (these 5 bits can easily be computed by applying the linear expansion backward). The five associated perturbative masks are then deduced by translation of round, and rotation of bits. They start with respectively 1, 2, 3, 4 and 5 zeroes.

2.1.2 Introducing non linear functions

We now introduce `SHI2`, another model of `SHA` where we replace the *Add* function by a *Xor* as we did in `SHI1`, but where we keep the non-linear functions f_j. It can easily be seen that in some cases the f_j behaves like a *Xor*. Thus, the previous attack may work. The question that arises is "What is the probability of success?" The precise answer to this question is a little tricky, and we therefore only sketch the method of computation.

In order to compute the probability we need to make a detailed analysis of the If and Maj functions. Since these functions work in parallel on 32 bits, we need only study what happens on a single bit. Assuming that we study the behaviour of the transition from $f(b_k, c_k, d_k)$ to $f(b'_k, c'_k, d'_k)$, by looking at the rotations and at our pertubation model one can see that different cases can occur:

1. There is no change at all in the inputs. In that case the output does not change and f behaves as Xor.

2. There is a single difference in the entry in position b. In that case, f behaves as a Xor, if and only if the output of f changes.

3. There is a single difference in the entry in position c or d. In that case, f behaves as a Xor, if and only if the output of f changes.

4. There are two differences in the entry in position c and d. In that case, f behaves as a Xor, if and only if the output of f does not change.

We can now look at the three last cases for the Maj and If function. For the Maj function, cases 2 and 3 behave identically, the change in the output occurs if and only if the two bits of input that do not change are opposite. This occur with probability 1/2. In case 4, the output does not change if and only if the two bits c_k and d_k change in opposite directions. We can ensure this by forcing a few conditions on the input bit of `SHI2` after the expansion. Thus, the probability of good behaviour in case 4 for the Maj function is 1.

For the If function, in case 2 the output changes if and only if bits c_k and d_k are opposite. This occurs with probability 1/2. In case 3, the output changes if and only if bit b_k points on the changing bit (i.e. $b_k = 1$ if c_k changes and $b_k = 0$ if d_k changes), this occurs with probability 1/2. In case 4, the ouput will always change, so the probability of good behaviour is 0. This implies, that we need to choose a pertubation pattern with no two adjacent pertubations in the If rounds.

Under all our constraints, we were able to find a pattern with a global probability of success $1/2^{21}$:

```
00000 00010100101000001111
      10011110011000111111
      10110111100001100010
      10101110100101000000
```

Evaluating the probability of success is a quite tricky, because the 16 first round must not be included in this evaluation. The reason for this is that a good collision path for the 15 first round stay good even is the input word of the 16th round changes. However, if something wrong happens in any of the subsequent rounds, there is no choice but to backtrack to round 16, since all the following input words are outputs of the expansion. Yet, when the probability of success become very small it is necessary every once in while to backtrack to round 1, because round 16 offers only a limited number of choices (2^{32}).

In the case of `SHI2`, the collision search is very fast and can be performed in just a half minute. Here is a sample collision:

```
6bb5101e 239ca976 6e07d2f2 4d419345
272bd97b ec7aa3ee 76ae3b7a 927990d2
0440db86 b0d2360b a1767279 4650fb90
9b8ba39c 0b5bfbc5 003ff5b7 5058904c
```

and

```
6bb5101e 239ca976 6e07d2f2 4d419347
272bd93b ec7aa3ee f6ae3b3a 127990d0
0440db84 30d2364b 21767279 c650fbd0
1b8ba39e 0b5bfbc5 803ff5b7 d058904c
```

both gives

```
1423379a 8d424727 156d7f93 cb8c56b5 56eee67b
```

after the 80 rounds of the SHI2 function.

2.1.3 Introducing addition

We now build a SHI3 model that uses the *Add* function of SHA but sticks to *Xor* in place of the non-linear functions of SHA. The propagation of the perturbation is now slightly different, because the first perturbation can introduce more than one bit perturbations if a carry occurs.

If the pertubation stays on one bit, then in order to correct it, the correction bits should change in the opposite direction of the pertubation bit (A change from 0 to 1 should be corrected by a change from 0 to 1 and vice-versa). If this condition holds, then correction of error bits in a and e will occur with probability 1. However, error bits in b, c or d may cause some difficulty. The reason for this is that the *Xor* function may reverse the direction of the change (with probality 1/2). In that case, the overall probability of success of a simple pertubation pattern is 1/16, this is too small for the collision search to have a reasonable hope of success. However, we may use a new trick to improve this probability. This (simple) trick uses the fact that the high-order bit of the *Add* function behaves as a *Xor* since the eventual carry is lost. Thus, if some corrections can happen on bit 31 it will significantly improve the attack. This explains our above choice to insert pertubation on bit 1, because in that case, after the left ROL-30 rotation, the change will finally occur on bit 31 of c, d and e. Thus, two conditions only remain (one for the insertion of the pertubation and one for the correction of b), and the probability of success of a simple pertubation pattern becomes 1/4.

As in the SHI2 case, evaluating the overall probality of success is tricky, but it is roughly $\frac{1}{2^w}$, where w is the number of pertubation bits after round 16 of SHI3 . The best pattern we could find is:

```
00000 01000010100100011110
      01011000001110000000
      00001100000011011000
      00011000101101100000
```

and its associated probability of success is $1/2^{45}$, which leads a feasible attack, given enough computing power.

2.2 True SHA case

In the true SHA case, we must take into account considerations seen both in SHI2 and SHI3 . We must also see whether the direction of a change in case 2 can be reversed by the If and Maj functions (we do not care about the direction of the change in cases 3 and 4 because they occur on bit 31). It can easily be seen that in the Maj case the direction of a change is never reversed, so the good case occur with probability 1. In the If case, the direction of the change may be reversed, and an additionnal 1/2 probability must be taken into account.

We were able to find several patterns with probability $1/2^{71}$ or sightly less. Here are some such patterns:

```
00000 00010000000100100000
      00100001101101111110
      11010010000101010010
      10100010111001100000 c=71

00000 00100010000000101111
      01100011100000010100
      01000100100100111011
      00110000111110000000 c=72
```

In the middle of the second 20-rounds block of the pattern with probability $1/2^{72}$ we find a block of 5 zeroes (in fact 6 but 5 is sufficient for our purpose).

This allows us to stop the attack after the block with a partial collision after 35 rounds of SHA. Here is such a partial collision:

```
78fb1285 77a2dc84 4035a90b b61f0b39
4a4d1c83 186e8429 74326988 7f220f79
a08e7920 16a3e469 2ed4213d 4a75b904
38bef788 2274a40c 4c14e934 cee12cec
```

and

```
78fb1285 77a2dc84 4035a909 b61f0b79
4a4d1c81 986e8429 f432698a ff220f39
a08e7922 96a3e469 aed4213d ca75b904
38bef788 2274a40c 4c14e936 cee12cac
```

both yields after 35 rounds of SHA-0

```
7b907fb9 d050108b 88d6e6d6 5c70d4a3 7e06a692
```

3 SHA-1 case

In the SHA-1 case, the bits are interleaved and therefore it is no more possible to split the expansion. However, the invariance by translation is still true. Hence, it is still feasible to deduce the 5 corrective masks from the perturbative one. Finding the perturbative mask can be done using coding theory tools[3], because the mask can be considered as a low-weight codeword of the extension. Performing such a search on E_1 leads to some very short codewords as compared to the dimensions of the code. However, with very high probability, no codeword of weight less than 200 exists in E_1, that satisfies the constraints (see 2.1.1), whereas there exists 28 weighted codewords in E_0.

Our attack will therefore be totally unefficient on SHA-1.

4 Conclusion

We have developped a new kind of attack on SHA functions that yields better results than the classical birthday-paradox attack on SHA-0. This attack is apparented to the well known differential cryptanalysis [1] in that it looks for some kind of characteristic masks that can be added to input word with non trivial probability of unchanging the output of the compression function. The expansion of `SHA-1` seems to be designed to counter this kind of attack, which should increase the level of confidence in this standard.

References

[1] E. Biham, and A. Shamir. Cryptanalysis of the Full 16-Round DES, *CRYPTO'92* LNCS 740, pp 487–496, 1993.

[2] B. den Boer, and A. Bosselaers. Collisions for the compression function of MD5, *EUROCRYPT'93* LNCS 773, pp 293–304, 1994.

[3] A. Canteaut, and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to primitive narrow-sense BCH codes of length 511, *IEEE Trans. Inform. Theory*, IT-44(1), pp 367–378, Jan. 1998.

[4] H. Dobbertin. Cryptanalysis of MD4, *Fast Software Encryption* LNCS 1039, pp 53–69, 1996.

[5] R. Rivest. The MD4 Message-Digest Algorithm, *Network Working Group Request for Comments: 1321*, April 1992. `http://theory.lcs.mit.edu/ rivest/Rivest-MD5.txt`

[6] R. Rivest. The MD5 Message-Digest Algorithm, *Network Working Group Request for Comments: 1321*, April 1992. `http://theory.lcs.mit.edu/ rivest/Rivest-MD5.txt`

[SHA-0] Secure Hash Standard. *Federal Information Processing Standard Publication # 180*, U.S. Department of Commerce, National Institute of Standards and Technology, 1993.

[SHA-1] Secure Hash Standard. *Federal Information Processing Standard Publication # 180-1*, U.S. Department of Commerce, National Institute of Standards and Technology, 1995 (addendum to [SHA-0]).