Update on SHA-1*

Vincent Rijmen[1,2] and Elisabeth Oswald[1]

[1] IAIK, Graz University of Technology,
Inffeldgasse 16a, A-8010 Graz, Austria
{vincent.rijmen,elisabeth.oswald}@iaik.tugraz.at
[2] Cryptomathic A/S,
Jægergårdsgade 118, DK-8000 Århus C, Denmark

Abstract. We report on the experiments we performed in order to assess the security of SHA-1 against the attack by Chabaud and Joux [5]. We present some ideas for optimizations of the attack and some properties of the message expansion routine. Finally, we show that for a reduced version of SHA-1, with 53 rounds instead of 80, it is possible to find collisions in less than 2^{80} operations.

Keywords: hash functions, cryptanalysis

1 Introduction

In [5], Chabaud and Joux presented a method to find collisions for the original Secure Hash Standard (here denoted by SHA-0). We present here the results of our attempts to apply their attack to SHA-1, as well as some extensions to the approach described in [5]. For a good understanding of our results, it is recommended to study [5] very carefully. Space restrictions do not permit us to copy all the important details of the original attack.

In the case of SHA-0, the message expansion shows a certain weakness, which allows to reduce the search space for difference patterns to a size which makes exhaustive search possible. This weakness has been fixed in SHA-1 and consequently, it was necessary to design and implement more intelligent searching algorithms.

Furthermore, we investigated the use of alternative linear approximations for the non-linear functions. We also optimized the equation solving step, which allows to solve larger systems of equations. Finally, we analyzed in much detail the complexity of an attack on a version of SHA-1 reduced to 53 rounds.

The appendix lists some expanded message words with very low weight, which we can't use in an attack.

In parallel to our research, Biham and Chen [1] improved the complexity of the Chabaud-Joux attack on SHA-0. The same authors announce forthcoming results on SHA-1 in [2]. Saarinen describes attacks on block ciphers based on SHA-1 in [6].

* This research was supported financially by the A-SIT, Austria and by the BSI, Germany.

A.J. Menezes (Ed.): CT-RSA 2005, LNCS 3376, pp. 58–71, 2005.

2 SHA-0 and SHA-1

The SHA family of hash functions is described in [4]. Briefly, the hash functions consist of two phases: a message expansion and a state update function. These are explained in more detail in the following. SHA-0 and SHA-1 share the same state update, but SHA-0 has a simpler message expansion. Both SHA-0 and SHA-1 consist of 80 rounds. Because we will mainly study reduced versions here, we make the number of rounds variable, and denote it by R.

2.1 Message Expansion

In SHA-1, the message expansion is defined as follows. The input is a 512-bit message, denoted by a row vector m. The message is also represented by 16 32-bit words, denoted by M_t, with $t = 0, 1, \ldots, 15$.

In the message expansion, this input is expanded linearly into R 32-bit words W_t, also denoted as the $32R$-bit expanded message word w. The words W_t are defined as follows.

$$W_t = M_t, t = 0, \ldots 15 \tag{1}$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, t > 15 \tag{2}$$

The message expansion of SHA-0 is very similar, but uses:

$$W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ . \tag{3}$$

Consequently, a bit at a certain position i in one of the words of w only depends on the bits at corresponding positions in the words of m.

2.2 State Update Transformation

The state update transformation starts from a (fixed) initial state for 5 32-bit registers and updates them in R steps, using one word W_t in every step. Figure 1 illustrates one step of the state update function. The function f depends on the round number: rounds 1 to 20 use the *IF-function*, rounds 41 to 60 use the *MAJ-function*.

$$f_{\mathrm{if}}(X, Y, Z) = XY \oplus \overline{X}Z \tag{4}$$

$$f_{\mathrm{maj}}(X, Y, Z) = XY \oplus XZ \oplus YZ \tag{5}$$

The remaining rounds use 3-input XOR. A round constant K_t is added in every round. There are four different constants; one for rounds 1 to 20, one for rounds 21 to 40, one for rounds 41 to 60 and one for rounds 61 to 80. After the last application of the state update transformation, the initial register values are XOR-ed to the final values, and the result is outputted.

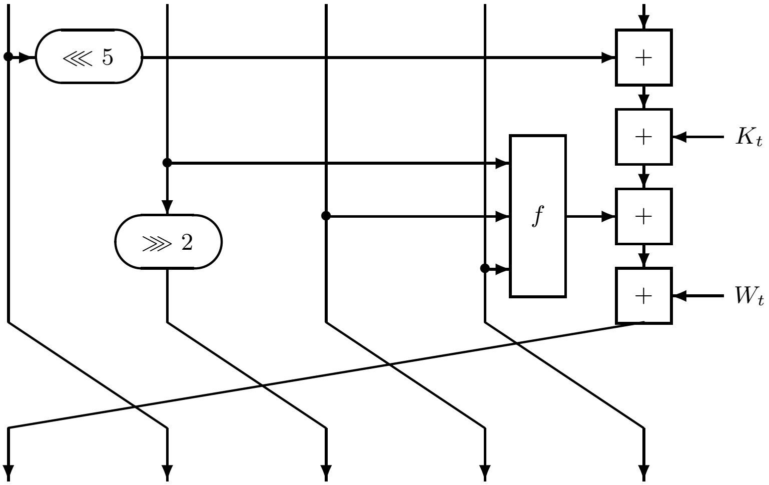

Fig. 1. One step of the state update function of SHA-1.

3 The Basic Attack Strategy

The attack on SHA-0 can be summarized as follows [5].

1. Firstly, a linear approximation of SHA-0 is constructed.
2. Secondly, collisions for the linear approximation are determined.
3. Thirdly, a collision for the real SHA-0 is searched among the collisions for the linear approximation.

We now discuss each of these steps and point out the differences between SHA-0 and SHA-1 relevant to the attack. The next sections also go into more detail for step 2 and step 3.

3.1 Determining a Linear Approximation

In this step, we replace all non-linear components by linear approximations. For our purposes, a linear function λ is a 'good' approximation for a non-linear function γ if the relation

$$\gamma(x \oplus \delta) \oplus \gamma(x) = \lambda(x \oplus \delta) \oplus \lambda(x) = \lambda(\delta) \tag{6}$$

holds for relatively many values of x and δ. The complexity of the third step of the attack is influenced by the quality of the approximation.

Both in SHA-0 and SHA-1, there are 3 non-linear components. Firstly, there is the addition modulo 2^{32}. This component is approximated by a bitwise exclusive-or of the inputs, i.e. the carry is ignored.

Next are the functions $f_{\mathrm{if}}(X, Y, Z)$ and $f_{\mathrm{maj}}(X, Y, Z)$. The functions operate bitwise, hence the approximation should also be a bitwise function. The authors of [5] approximate both functions by a bitwise exclusive-or of the 3 inputs. Since the f-function used in half of the rounds is exactly given by this exclusive-or, this approximation results in 80 iterations of the same round. On the other hand, the quality of this approximation seems sub-optimal. We discuss some ideas for improvements in Section 4.

3.2 Finding Collisions for the Linear Approximation

Finding *a* collision for a linear approximation of SHA-1 is not difficult. Whether two messages will produce a collision or not, doesn't depend on the value of the messages, but depends on the value of their difference only. Collision-producing values of the difference can be found as the solutions of an under-determined set of linear equations.

The difficulty lies in the additional constraints imposed by the third step of the attack. We want to find a collision that minimizes the work of the third step. The explanation of the third step will show that the most important requirement is that the weight of the difference should be small.

The problem of finding a collision-producing difference with small weight can be translated to the problem of finding a codeword with small weight in a linear code.

Since the message expansion is linear, there is a $512 \times 32R$ matrix E such that $w = mE$. The message expansion starts with a copy of the message, cf. (1). Hence, there is a $512 \times 32(R-16)$ matrix F such that E can be written as:

$$E_{512\times 32R} = \left[I_{512\times 512} \; F_{512\times 32(R-16)} \right] \; . \tag{7}$$

For the linearized state update transformation, we can construct a $32R \times 160$ matrix A that produces the output vector o from the expanded message w.

$$o = wA = mEA \tag{8}$$

A message word m corresponds to a collision-producing difference if and only if $o = 0$. Hence, the set of collision-producing differences is a linear code with check matrix $H_{160\times 512} = (EA)^t$. The dimension of the code is $k = 512 - 160 = 352$. The length of the code is $n = 512$. Later on, it becomes more useful to look at the expanded words w, which are code words of a code with $k = 352$, $n = 32R$ and check matrix

$$H'_{(32R-352)\times 32R} = \begin{bmatrix} A^t|_{160\times 32R} \\ F^t|_{(32R-512)\times 512} \; I_{(32R-512)\times(32R-512)} \end{bmatrix} \; . \tag{9}$$

In [5] an additional condition is imposed on the differences. The differences are constructed as the sum of a *perturbation word* w_p and 5 *correction words* $w_{c,i}, i = 1, 2, 3, 4, 5$. The authors require that w_p and the 5 $w_{c,i}$ are all codewords. It can be seen that this is a sufficient, but not a necessary condition for the sum to be a codeword as well.

This restriction of the search space corresponds to adding 160 rows to the check matrix of the code. It allows to zoom in quickly on the optimal solutions in the case of SHA-0. In the case of SHA-1, our experiments indicate that this restriction on the search space leads to suboptimal results (cf. Table 4). Hence, we can't apply the perturbation-correction technique. A small side-effect is that the weights we quote in this paper, are always weights for the full codeword, which makes it difficult to compare them to the results in [5], where the authors quote the weights of the perturbation words only.

3.3 Finding the Collisions for the Real SHA-1

We now have a difference that produces collisions in the linear approximation of SHA-1. In the non-linear components of SHA-1, the propagation of the differences may be the same as in the linear case, or it may be different, depending on the value of the message bits.

In order to find a collision, we want to find the values of the inputs such that the difference propagation in SHA-1 corresponds to the difference propagation in the linear approximation. This condition results in the equations that we have to solve in order to find the collision.

For example, consider the addition of a register A_t and a word W_t of the expanded message. Let the result be denoted by B_t. Assume now that in both inputs, the differences are equal to zero except for one bit, at position i. It is clear that in the linear approximation of the addition, the result has difference zero. In B_t, the result of the real addition, the difference will be zero for the bits with positions $0, 1, \ldots, i$. The difference at positions above i will be zero if and only if the carry into position $i + 1$ has difference zero. Writing down the equations for the carry results in the following requirement on the value of the operands: the bits at position i in A_t and W_t should be of opposite value.

If we look at the equations generated during the attack, we typically get groups of several equations involving the same state bit(s). It is then often possible to rework some of the equations and obtain linear equations involving bits of the expanded message words only. Linear equations in the expanded message words can easily be translated into conditions on the message words. They can easily be solved.

Besides the additions, also the approximations of f_{if} and f_{maj} result in conditions. These conditions can be derived from Table 1. If the functions are approximated by 3-input XOR, then the first 3 and the 7th equation have to be copied from the table (output difference equal to 1 is desired), while the 4th, 5th and 6th have to be inverted (output difference equal to 0 is desired). The conditions result in equations involving bits from one, two or three registers. Some special attention should go to the 4th equation for f_{if}, respectively the 7th equation for f_{maj}. For instance, when f_{if} is approximated by 3-input XOR, this input difference is problematic. It was this observation that led us to the idea of considering other linear approximations. This is discussed in Section 4.

Table 1. Conditions to have output difference equal to 1.

$f_{\text{if}}(x,y,z)$	
δ	equation
001	$x = 0$
010	$x = 1$
100	$y + z = 1$
011	always
101	$x + y + z = 1$
110	$x + y + z = 0$
111	$y + z = 0$

$f_{\text{maj}}(x,y,z)$	
δ	equation
001	$x + y = 1$
010	$x + z = 1$
100	$y + z = 1$
011	$y + z = 0$
101	$x + z = 0$
110	$x + y = 0$
111	always

4 Other Linear Approximations

4.1 Motivation

Approximation of both f_{if} and f_{maj} by 3-input XOR has as main advantage that the resulting approximation for SHA-1 has 80 identical rounds. There appear to be also a couple of disadvantages related to this choice:

1. For $\delta = 011$, the output bit of the linear approximation flips with probability 0, but the output bit of f_{if} flips with probability 1. Hence there is no input pair that produces the same behavior in f_{if} and the approximation.
2. 3-Input XOR has good diffusion properties (avalanche effect). Other linear functions on the 3 inputs have worse diffusion properties.

The first property complicates the search for a suitable collision in the linear approximation of SHA-1, because the situation where $\delta = 011$, has to be avoided. The restriction of the search space corresponds to the addition of non-linear conditions. The second property makes it more difficult to prevent a difference from expanding; more equations have to be added. Therefore we considered also other linear approximations, which produce different equations, that can also be generated by copying and inverting equations from Table 1.

Table 2 lists for both f-functions and the 7 possible linear functions with 3 inputs the probability that the output bit flips for the given input difference. By definition, the output flip probability for a linear function is either 0 or 1; it can be computed by simply evaluating $f(\delta)$.

The quality of the approximations differs only for the values $\delta = 011$ and $\delta = 111$. For these values of δ, the output bit of f_{if}, respectively f_{maj}, flips with probability 1. The first disadvantage explained above can be avoided by selecting a linear approximation that does not have output bit flip probability equal to 0 when the non-linear function has output flip probability equal to 1. The second criterion for the quality of an approximation is the diffusion: an approximation with bad diffusion is more likely to result in low-weight collision-producing differences.

For the function f_{if}, the approximations y, z, $x \oplus y$ and $x \oplus z$ appear to be better choices. For the function f_{maj}, the approximations x, y and z have equally good flip probabilities as $x \oplus y \oplus z$, and less avalanche.

Table 2. The probability that the output bit changes value when the input bits are changed according to the input difference, for the two f-functions and for all the linear approximations.

δ	output flip probability								
	f_{if}	f_{maj}	linear functions						
	$xy \oplus \overline{x}z$	$xy \oplus xz \oplus yz$	x	y	z	$x \oplus y$	$x \oplus z$	$y \oplus z$	$x \oplus y \oplus z$
000	0	0	0	0	0	0	0	0	0
001	1/2	1/2	0	0	1	0	1	1	1
010	1/2	1/2	0	1	0	1	0	1	1
011	1	1/2	0	1	1	1	1	0	0
100	1/2	1/2	1	0	0	1	1	0	1
101	1/2	1/2	1	0	1	1	0	1	0
110	1/2	1/2	1	1	0	0	1	1	0
111	1/2	1	1	1	1	0	0	0	1

4.2 Results

Despite the expected improvements in the search for low-weight codewords, the results obtained with alternative linear approximations turn out to be inferior. We tried out replacing the approximations by any of the other linear functions with the same or better flip probabilities. For versions with more than 25 rounds, we never obtained better results than with the original approximation.

We can think of two possible explanations. The use of alternative approximations for the Boolean functions results in an approximation for SHA-1 that has two different round transformations (at least), since the 40 rounds using the 3-input XOR clearly can't be approximated by another linear function. Hence we obtain a linear code with less regularity.

As we explain in Section 5, we use heuristic algorithms to search for low-weight codewords. A first explanation would be that the decrease in regularity causes an increase in the minimum distance of the code. But this almost implies that the round transformation of SHA-1 would show some kind of weakness, which results in a lower minimum distance of the corresponding linear code. An alternative explanation is that the decrease in regularity makes the heuristic search algorithms perform worse: the low-weight words are still there, but we can't find them. In that case, perhaps better search algorithms can be found.

5 Searching for Low-Weight Codewords

There is no fast, deterministic algorithm known that can find low-weight codewords in arbitrary linear codes. Different approaches are possible:

1. Exhaustive search.
2. Apply heuristic techniques that can be used to find low-weight codewords in random linear codes, e.g. [3].
3. Exploit the structure of the code and obtain an analytical solution.

In the case of SHA-0, it is possible to define a restricted search space that is very likely to contain the best codewords. Since the restricted search space has dimension 2^{16}, exhaustive search is possible. For SHA-1, it seems impossible to define a search space small enough to allow an exhaustive approach.

It seems that the dimensions we are dealing with here, are still out of reach for the algorithms discussed in [3]. Secondly, as follows from Section 4, we are clearly not in the situation of a purely random code.

The best strategy seems to combine the second and the third approach. For instance, we know that the codewords are produced by an LFSR. If $w = (W_0, W_1, \ldots)$ is a codeword, then also $(\mathrm{rot}(W_0), \mathrm{rot}(W_1), \ldots)$ is a codeword. More specific knowledge of the LFSR allowed us to define other strategies resulting in words with a weight probably very close to the minimal weight for $R < 50$. For larger values of R, the problem is still open.

5.1 Our Search Algorithm

Our heuristic search algorithm is based on an observation that resulted from experiments on SHA-1 versions with $R \leq 25$. First, we introduce the following notation. Let the bitwise OR operation be denoted by $\vee$, then we define the following shorthand notation.

$$W_\vee = \bigvee_{i=0}^{R-1} W_i \tag{10}$$

Observation 1. *For codewords $w = (W_0, W_1, \ldots, W_{R-1})$ with low Hamming weight, the Hamming weight of $W_\vee$ is low. In other words: codewords with low Hamming weight have the property that the non-zero bits usually occur at the same positions in all the words W_i.*

The observation was derived from experiments, but we believe it is also in agreement with intuition. Differences introduced in the state have to be compensated for and eventually canceled. This requires that the differences in the expanded message words occur in 'bands'. Algorithm 1 uses the observation to perform an accelerated search. The results obtained with the algorithm are shown in Table 3. Further restriction of the search space is possible by using Observation 2.

Observation 2. *The non-zero bits in $W_\vee$ occur at consecutive positions, or 'almost' consecutive positions.*

By 'occur in almost consecutive positions,' we mean that there are at most two runs of ones, separated by a run of one or two zeroes. Motivated by this observation, we remove the inner for-loop of Algorithm 1, which results in an important speedup. Algorithm 2 uses a parameter u, which denotes the sum of the lengths of the runs of ones and the one or two zeroes in between.

By starting with the large values of u and moving to the lower ones, we save on the operations needed to compute the new check matrix of the code and its rank. The algorithm starts an exhaustive search when the dimension of the code

Table 3. Minimal weights of collision-producing codewords for reduced versions of SHA-1. Produced with Algorithm 1.

R	$\mathrm{Hwt}(W_\vee)$	$\mathrm{Hwt}(w)$
20	3	18
25	6	34
30	6	38
31	6	38
32	6	38
33	6	38
34	6	38
35	8	76
36	8	76
37	8	80
38	8	100
39	8	112
40	10	128

gets below a parameter D. Algorithm 2 was executed for various numbers of rounds. In the cases where there were several values for u that resulted in a code with less than 2^D codewords, it was always the case that the codeword with the lowest weight was among those with the smallest u. The results are presented in Table 4.

In order to optimize the complexity of the attack, the first 15 message words are pre-computed such that the conditions in the first 15 rounds are satisfied. Hence, the weight in the first 15 rounds is not relevant. Therefore, the results in Table 4 do not take into account the first 15 rounds (neither in $\mathrm{Hwt}(w)$, nor in u). The results indicate that a shortcut attack finding a collision is feasible for versions reduced to 35–40 rounds. In Section 6, we examine the complexity of the attack for a version reduced to 53 rounds.

5.2 Observations on the Message Expansion

We applied Algorithm 2 also to the naked message expansion: dropping the condition to have a collision and simply searching for low-weight expanded message words. The results are given in Table 5. Since the words are obtained with a heuristic algorithm, it is not proven that these are the best words. Indeed, in Appendix A, we give three 80-rounds expanded message words with Hamming weight 44 (including the first 15 rounds), obtained by other means. Nevertheless, the values in the table give some indication about the 'penalty' in additional weight coming from the requirement to produce a collision.

For all the words listed in this paper, it can be observed that the rounds that contribute the most to the total weight, are situated at the beginning and at the end. Furthermore, the rounds with the lowest weight aren't situated exactly in the middle, but slightly more towards the end of the word. This can be explained by the fact that the diffusion of the message expansion goes slower in the backwards direction.

Algorithm 1

```
Input: $H$ /* check matrix of the code */
       $n$ /* length of the code */

For $h = 1$ to 32 do
  For all values of $W_\vee$ with Hamming weight $h$ do
    Copy $H$ to $H_e$
    Extend $H_e$ with $R \times (32 - h)$ rows corresponding to the conditions on
      the bits of $W_0, W_1, \ldots$ at the positions where $W_\vee = 0$
    If rank$(H_e) < n$ then
      Perform an exhaustive search for low-weight codewords
      Output the word with lowest weight and exit
```

Algorithm 2

```
Input: $H$ /* check matrix of the code */
       $n$ /* length of the code */
       $D$ /* exhaustive search space bound */

For $u = 32$ to 1 do
  Extend $H$ with $R$ rows corresponding to the conditions on
    the bits of $W_0, W_1, \ldots$ at position $h - 1$
  If $n -$ rank$(H) < D$ then
    Perform an exhaustive search for low-weight codewords
    Output the word with lowest weight
```

Table 4. Hamming weights of the codewords with smallest weights, for reduced versions of SHA-1. Produced with Algorithm 2. The weight of the first 15 rounds is not taken into account. The fourth and the fifth column list the results obtained when the search space is restricted to the *perturbation-correction* codewords used with success on SHA-0 in [5]. The second and the third column give the results for an unrestricted search space.

	full search space		restricted space	
R	u	Hwt(w)	u	Hwt(w)
35	6	35	16	127
40	8	67	17	178
45	8	81	17	215
50	8	83	18	258
51	8	86		
53	8	95		
54	10	145		
55	10	157		
60	12	167		
65	12	226		
70	12	276		
75	13	278		
80	14	333	21	552

Table 5. Weight of low-weight codewords for the message expansion of SHA-1. The weight includes the weight of the first 15 rounds. Produced with Algorithm 2.

R	u	Hwt(m)
50	3	20
60	3	31
70	4	41
80	5	51

The diffusion of the message expansion (2) is determined by the feedback polynomial and the rotation. If we visualize the expanded message as a $32 \times R$ rectangle, then the 'influence region' of a bit occurring at position i in round t can be visualized as two triangles meeting at the point (t, i). This is illustrated in Figure 2. In the forward direction, the upper line of the triangle has a slope of 3 rounds per bit. The lower line has a slope of 16 rounds per bit. In the backwards direction, the influence region is bounded by the horizontal line and a line with a slope of 16 rounds per bit. This region expands much slower.

Consequently, a good strategy to find a low-weight codeword is to place a word with weight 1 somewhere in the middle, add 7 or 8 zero words before and after, and compute the other rounds backwards and forwards. Since the diffusion backwards goes slower, it is better to compute more rounds backwards than forwards.

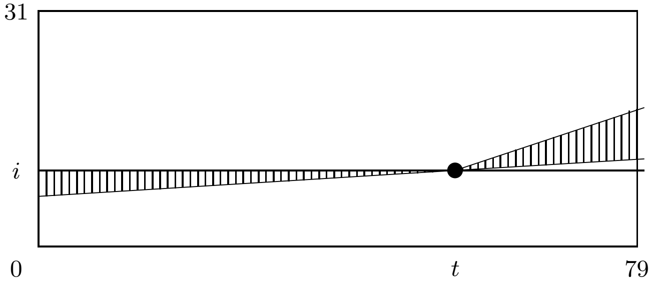

Fig. 2. Diffusion in the SHA-1 message expansion. A bit at position i in round t influences only bits in the shaded triangles.

6 Experiments with a 53-Round Characteristic

Table 4 shows that the weight of our best 54-round codeword is significantly larger than the weight of our best 53-round codeword. Therefore, we decided to use a version of SHA-1 reduced to 53 rounds to apply the third step of the attack: generating (and possibly solving) the equations. The codeword we used, is listed in Table 6.

During the generation of equations, a new problem became apparent. In the original attack on SHA-0, the modular additions have input differences in 0, 1

Table 6. Low-weight codeword for SHA-1 reduced to 53 rounds.

00000000	80000030	00000020
00000000	20000001	80000001
00000000	C0000012	C0000002
40000000	60000041	40000040
00000008	40000032	40000002
40000002	20000003	80000002
90000040	C0000042	80000040
50000011	E0000042	80000002
10000068	E0000002	80000000
E0000002	00000002	80000000
F0000022	00000040	80000000
70000051	80000001	00000000
10000010	80000060	00000000
60000041	80000001	
C0000022	40000042	
80000003	C0000040	
E0000052	40000042	
C0000040	00000000	
E0000052	80000040	
20000003	00000003	

or 2 of the input words. However, our codeword results also in situations where 3 input words have a non-zero difference at the same position. In this situation, a carry to the next position can't be avoided. Hence, there is no input that can behave the same in the linear approximation and in the real SHA-1. An exception to this rule is of course formed by the most significant bit position, where the carry simply overflows. It turns out that we are lucky enough that all the situations with 3 non-zero input differences occur at the same bit position, hence we can choose a rotated version of the codeword where all cases happen at the most significant bit position.

In order to achieve the best results, we decided to avoid the 'IF-rounds' of SHA-1, by defining the start of our reduced version at round 21. Doing this, we get a total of 166 equations. From these equations, we can isolate 62 linear equations in bits of the message words only, leaving 104 equations. 33 equations apply to the first 15 rounds, and can be solved explicitly during the pre-computation phase, leaving 71 equations for the main step. Using the most naive methods for solving non-linear Boolean equations, these equations can be solved with a complexity that is close to but below that of 2^{71} hash function evaluations. Hence, in principle it is possible to find collisions for the reduced version, faster than by the birthday paradox.

7 Conclusions

In this paper we presented the results from our attempts to extend the Chabaud-Joux attack to SHA-1. The application to SHA-1 results in several complications, which were not obvious from the start. We proposed several strategies for optimization of the attack and examined their effectiveness.

As a result, we have described a theoretical shortcut attack on a version of SHA-1 reduced to 53 rounds. The shortcut attack becomes feasible for SHA-1 reduced to 35–40 rounds. It is also clear that we are still far from even a theoretical attack on the full SHA-1.

Furthermore, we presented several observations that came out of our experiments. We hope that they might be of use for other cryptographers trying to break SHA-1.

Acknowledgements

The authors wish to thank Carlos Cajal for assistance with the programming, and Antoon Bosselaers for helpful discussions.

References

1. Eli Biham, Rafi Chen, "Near-Collisions of SHA-0," *Advances in Cryptology – Crypto '04, LNCS*, M. Franklin, Ed., Springer-Verlag, to appear.
2. Eli Biham, Rafi Chen, "Near-Collisions of SHA-0," *Cryptology ePrint Archive, Report 2004/146*, 2004, version of June 22, 2004, http://eprint.iacr.org/.
3. Anne Canteaut, Florent Chabaud, "A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511," *IEEE Transactions on Information Theory*, Vol. 44, No. 1, January 1998.
4. *Federal Information Processing Standard 180-2, Secure Hash Standard*, August 1, 2002.
5. Florent Chabaud, Antoine Joux, "Differential Collisions in SHA-0," *Advances in Cryptology – Crypto '98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 56–71.
6. Markku-Juhani O. Saarinen, "Cryptanalysis of Block Ciphers Based on SHA-1 and MD5," *Fast Software Encryption 2003, LNCS 2887*, T. Johansson, Ed., Springer-Verlag, 2003, pp. 36–44.

A Some Low-Weight Codewords

Below is a codeword for 80 rounds, with weight 51. It is the word corresponding to the entry in Table 5. The weight includes the weight of the first 15 rounds. The ordering is from top to bottom, and then from left to right.

10000000	40000000	40000000	00000000
20000000	40000000	20000000	40000000
00000000	40000000	00000000	00000000
30000000	00000000	00000000	00000000
40000000	00000000	40000000	00000000
41000000	20000000	60000000	00000000
40000000	00000000	00000000	00000000
40000000	50000000	40000000	00000000
10000000	40000000	40000000	00000000
40000000	50000000	00000000	00000000
00000000	00000000	00000000	00000000
30000000	40000000	40000000	00000000
00000000	00000000	00000000	00000000
21000000	60000000	00000000	00000000
40000000	00000000	00000000	00000000
40000000	40000000	40000000	00000000
50000000	40000000	00000000	00000000
20000000	20000000	40000000	80000000
00000000	00000000	00000000	00000000
30000000	40000000	40000000	00000000

The absolutely smallest weight we found for the 80-round message expansion, is 44. We found 3 such codewords, given below. The 3 codewords have a large amount of W_i in common. The first codeword starts top left and ends at the bottom of the fourth column, the second codeword is shifted over 4 W_i's, the third one over a further two. The words have $W_\vee$ = C0000FF, C0001FF, C0003FF and Hwt$(W_\vee) = 10, 11, 12$.

80000000	80000000	00000001	00000002	00000050
00000000	00000000	00000000	00000000	00000100
C0000000	00000001	00000001	00000000	00000010
00000001	00000001	00000000	00000004	00000000
00000001	80000000	00000001	00000000	00000210
00000001	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000008	
00000000	00000001	00000000	00000000	
80000000	80000001	00000000	00000004	
00000000	00000000	00000000	00000010	
40000001	00000001	00000000	00000000	
00000001	00000001	00000000	00000000	
40000001	00000000	00000000	00000020	
00000000	00000000	00000000	00000000	
00000001	00000001	00000000	00000014	
00000000	00000000	00000000	00000040	
80000001	00000000	00000000	0000000C	
00000000	00000000	00000000	00000000	
00000001	00000001	00000000	00000080	
00000001	00000000	00000000	00000010	