Preimages for Reduced SHA-0 and SHA-1

Christophe De Cannière[1,2] and Christian Rechberger[3]

[1] Département d'Informatique École Normale Supérieure,
christophe.decanniere@ens.fr
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, and IBBT
[3] Graz University of Technology
Institute for Applied Information Processing and Communications (IAIK)
christian.rechberger@iaik.tugraz.at

Abstract. In this paper, we examine the resistance of the popular hash function SHA-1 and its predecessor SHA-0 against dedicated preimage attacks. In order to assess the security margin of these hash functions against these attacks, two new cryptanalytic techniques are developed:

– **Reversing the inversion problem**: the idea is to start with an impossible expanded message that would lead to the required digest, and then to correct this message until it becomes valid without destroying the preimage property.
– **P^{3}graphs**: an algorithm based on the theory of random graphs that allows the conversion of preimage attacks on the compression function to attacks on the hash function with less effort than traditional meet-in-the-middle approaches.

Combining these techniques, we obtain preimage-style shortcuts attacks for up to 45 steps of SHA-1, and up to 50 steps of SHA-0 (out of 80).

Keywords: hash function, cryptanalysis, preimages, SHA-0, SHA-1, directed random graph

1 Introduction

Until recently, most of the cryptanalytic research on popular dedicated hash functions has focused on collisions resistance, as can be seen from the successful attempts to violate the collision resistance property of MD4 [10], MD5 [32, 34], SHA-0 [6] and SHA-1 [13, 21, 33] using the basic ideas of differential cryptanalysis [2]. The community developed a wealth of fairly sophisticated tools that aid this type of analysis, including manual [33] and automated [7, 8, 20] methods to search and evaluate characteristics optimized for differential cryptanalysis of the used building blocks.

This wealth of results stands in stark contrast to what is known about the preimage and second preimage resistance of these hash functions. This is especially unsatisfying since most applications of hash functions actually rely more on preimage and second preimage resistance than on collision resistance.

Some of the main features of our results: All currently known generic preimage attacks require either impractically long first preimages [15], a first preimage lying in a very small subset of the set of all possible preimages [35], or a target digest constructed in a very special way [14].

In this work, we study the resistance of SHA-0 and SHA-1 against dedicated cryptanalytic attacks in settings where only relatively short preimages are allowed and a first preimage might not be available. An example of a very common use case of hash functions that relies on the resistance against these kind of attacks: hashed passwords. Especially SHA-1 is ubiquitously used, and will continue to be recommended by NIST even after 2010 outside the application of digital signatures [24], e.g., as RNG or KDF.

We exploit weak diffusion properties in the step transformation and in the message expansion to divide the effort to find a preimage, and consider only one or a small number of bits at a given time. In particular we present two new cryptanalytic tools. Firstly a compression function attack by means of correcting invalid messages, described in Sect. 3. Secondly, an algorithm based on the theory of random graphs that allows an efficient conversion of preimage attacks on the compression function to attacks on the hash function is presented in Sect. 4.

Later, in Sect. 5 we will discuss the results of combining these methods. This results in cryptanalytic shortcuts attacks for up to 50 step of SHA-0 (out of 80) and 45 steps of SHA-1. As a proof-of-concept we give a preimage for the 33-step SHA-0 compression function and also a second preimage of an ASCII text under the SHA-0 hash function reduced to 31 steps in Appendix B.

2 The SHA Family

In this paper, we will focus on the hash function SHA-1 and its predecessor SHA-0. The SHA-1 algorithm, designed by the US National Security Agency (NSA) and adopted as a standard in 1995, is widely used, and is representative for a large class of hash functions which started with MD4 and includes most algorithms in use today. In this section, we only briefly review a few features of the SHA design which are important for the techniques presented in this paper. For a complete description we refer to the specifications [25].

SHA-0 and SHA-1 consist of the iterative application of a compression function (denoted by f in Fig. 1), which transforms a 160-bit chaining variable h_{j-1} into h_j, based on a 512-bit message block m_j. At the core of the compression function lies a block cipher g which is used in Davies-Meyer mode (see Fig. 2). The block cipher itself consists of two parts: a message expansion and a state update transformation.

The purpose of the message expansion is to expand a single 512-bit input message block into eighty 32-bit words $W_0, \ldots, W_{79}$. This is done by splitting the message block into sixteen 32-bit words $M_0, \ldots, M_{15}$, which are then expanded linearly according to the following recursive rule:

$$W_i = \begin{cases} M_i & \text{for } 0 \le i < 16, \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll s & \text{for } 16 \le i < 80. \end{cases}$$

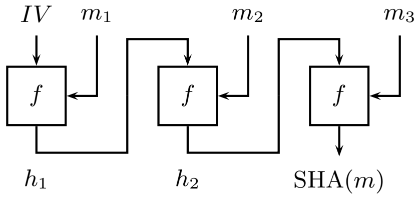


Fig. 1. An iterated hash function

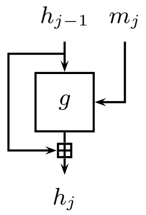


Fig. 2. The Davies-Meyer mode

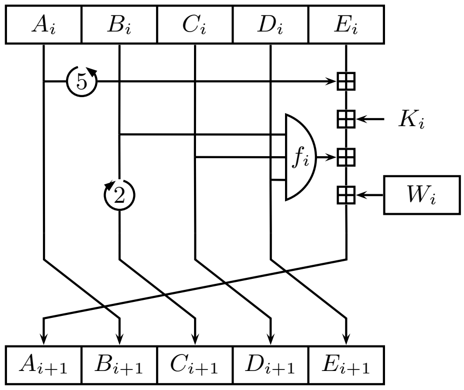


Fig. 3. A single state update step

The only difference between SHA-0 and SHA-1 lies in the rotation value s, which is 0 for SHA-0, and 1 for SHA-1.

The state update transformation takes as input a 160-bit chaining variable h_{j-1} which is used to initialize five 32-bit registers $A, B, \ldots, E$. These registers, referred to as state variables, are then iteratively updated in 80 steps, one of which is shown in Fig. 3. Note that the state update transformation can also be described recursively in terms of A_i only: after introducing $A_{-1} = B_0$, $A_{-2} = C_0 \lll 2$, $A_{-3} = D_0 \lll 2$, and $A_{-4} = E_0 \lll 2$, we can write:

$$A_{i+1} = (A_i \lll 5) + W_i + f(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \ggg 2) + (A_{i-4} \ggg 2) + K_i\,.$$

Because of this property, we will only consider the state variable A_i in the remainder of this paper.

3 Inverting the Compression Function

Before devising (second-) preimage attacks against the complete SHA function, we first focus on its compression function, and develop inverting methods which will be used as building blocks afterwards.

3.1 Possible Approaches

The recent successes in constructing collisions in SHA-0 and SHA-1 raise the natural question whether the differential techniques developed for collision attacks could also be used for constructing preimages. The question is especially pertinent in the case of second preimages, which are in fact just special types of collisions.

A first straightforward approach would consist in reusing the differential characteristics used in collision attacks by applying the corresponding message difference to the given message. If the characteristic is followed, then this will yield a second preimage. While this approach was applied to MD4 by Yu et al. [35], and to SHA-1 reduced to 53 steps by Rechberger and Rijmen [29, 30], it has some serious limitations when trying to find second preimages of reasonably short messages. The main problem is that, since the starting message is already fixed, the probability of the characteristic directly translates into the success probability of the attack (instead of determining the number of trials, as in collision attacks). This probability is further reduced by the fact that we lose the possibility to influence the difference propagation by fixing bits of the message to special values. In the case of MD4 and 53-step SHA-1, this results in attacks which only succeed with a probability of 2^{-56} and $2^{-151.5}$ respectively.

A second approach, which was recently proposed by Leurent in [17], relies on the existence of special messages which can simultaneously be combined with a large number of different characteristics, resulting in a large set of related messages. The idea is to compute the hash value of such a special message, and then apply the appropriate differences in order to steer this value towards the target value. Similar strategies have previously been used in practical second preimage attacks on SMASH by Lamberger et al. [16], and more recently in preimage attacks on GOST by Mendel et al. [18, 19]. In the case of MD4, this approach does not require a first preimage to start with, and results in a preimage attack against full MD4 with a complexity of 2^{100}.

It is not clear, however, how these ideas could efficiently be applied to hash functions such as SHA-0 or SHA-1, which, while still being vulnerable, show much more resistance to differential cryptanalysis than MD4. In the next sections, we will therefore study a completely different approach, which, as will be seen, has little in common with the techniques used in collision attacks.

3.2 Turning the Function Around

The problem we are trying to solve in this section is the following: given a 160-bit target value h_1, and a 160-bit chaining input h_0, find a 512-bit message input m_0 such that $f(h_0, m_0) = h_1$, or equivalently that $g(h_0, m_0) = h_1 - h_0$. Since the size of the message is much larger than the size of the output, we expect this equation to have a very large number of solutions. The difficulty in determining the 512 unknown input bits, however, lies in the fact that each of the 160 bit-conditions imposed at the output, depends in a complicated way on all 512 input bits.

The main observation on which the inversion method proposed in this paper is based, is that we can obtain a larger, but considerably less interconnected system of equations by expressing the problem in terms of internal state variables, rather than in terms of message words. That is, instead of trying to tweak a message in the hope to be able to control its effect on the output after being expanded and fed through several iterations of the state update transformation, we will start from state variables which already produce the correct output, and modify them

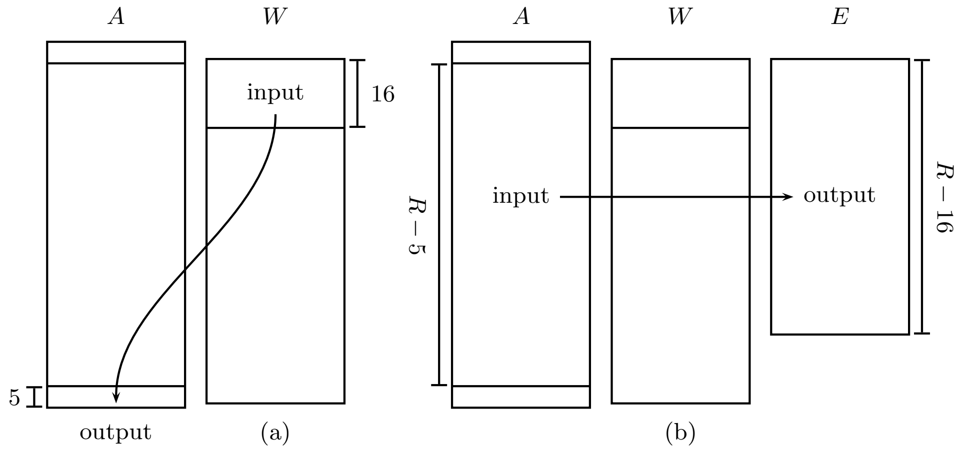


Fig. 4. Two equivalent descriptions of the inversion problem for a compression function reduced to R rounds

in such a way that the expanded message words, which can easily be derived from them, satisfy the linear recursion of the message expansion.

The idea is illustrated in Fig. 4. Instead of considering the function which maps $M_0 \dots M_{15}$ to $A_{76} \dots A_{80}$ as in Fig. 4(a), we will first fix $A_{76} \dots A_{80}$ to the target value determined by $h_1 - h_0$, and then analyze the function in Fig. 4(b) which maps $A_1 \dots A_{75}$ to error words $E_0 \dots E_{64}$, where

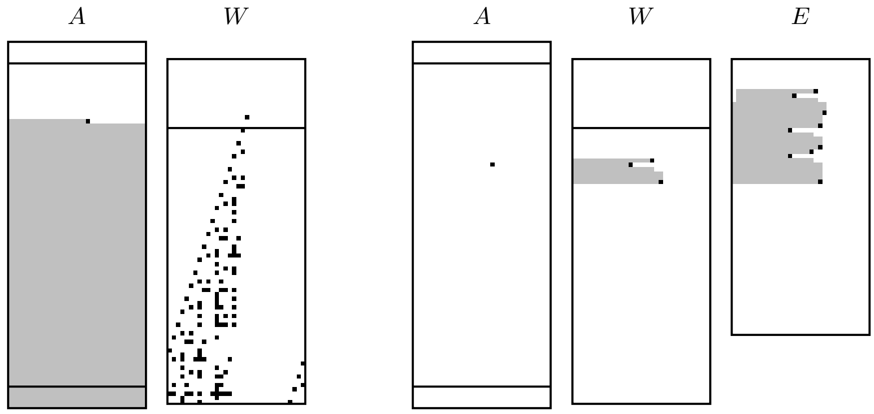
$$\begin{aligned} E_i &= W_i \oplus W_{i+2} \oplus W_{i+8} \oplus W_{i+13} \oplus (W_{i+16} \ggg s), \quad \text{and} \\ W_i &= A_{i+1} - (A_i \lll 5) - f(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \ggg 2) - (A_{i-4} \ggg 2) - K_i \,. \end{aligned}$$

Clearly, finding an input which maps to $h_1 - h_0$ in Fig. 4(a) is equivalent to the problem of finding an input which maps to zero in Fig. 4(b).

The potential advantages of this alternative approach are clearly seen when analyzing how flipping a single bit in the input affects the output in both cases. In the first case, illustrated in Fig. 5(a), a single flip in the message quickly propagates through both the expanded message and the state, resulting in a completely uncontrollable pattern of changes at the output. In the second case, however, a bit-flip in the state propagates to the output in a very predictable way, as shown in Fig. 5(b). A change in the state affects at most 6 consecutive expanded messages words, and at most 22 words of the output. More importantly, depending on the position of the flipped bit in the state word, it will leave the least significant bits of all W_i and E_i untouched. The downside is that both the input and the output of the function to invert are considerably larger.

3.3 Fixing Problems Column by Column

Let us now analyze in a little bit more detail how state bits affect the output words in our new function. In order to simplify the analysis, we will for now assume that we deal with a variant of SHA-0 reduced to R rounds.



(a) (b)

Fig. 5. Bits affected by a single bit flip at the input (SHA-1). Black bits are guaranteed to flip; gray bits may be flipped; white bits are unaffected

Suppose that we restrict ourselves to the first $j+1$ bits of each expanded message word W_i (denoted by $W_i^{j\cdots 0}$), and that we keep all state bits constant except for those at bit position $j+2$ (referred to as a_i^{j+2}). In this case, we can derive a simple relation (by collecting all constant parts into a $j+1$-bit word $C_i^{j\cdots 0}$ and a 1-bit variable c_i^j), which holds as long as $0 \leq j < 25$:

$$W_i^{j\cdots 0} = C_i^{j\cdots 0} - (f(c_i^j, a_{i-2}^{j+2}, a_{i-3}^{j+2}) \ll j) - (a_{i-4}^{j+2} \ll j)\,. \tag{1}$$

The interesting property of this relation is that the effect of the state bits a_i^{j+2} is confined to the most significant bit of $W_i^{j\cdots 0}$. Furthermore, this effect is linear in all rounds where f_{XOR} or f_{IF} is used. Since the words E_i in SHA-0 are just a bitwise XOR of expanded message words W_i, this property holds for those words as well.

We can now use this observation to gradually fix the bits of E_i to zero, column by column. We start by determining $a_1^2 \ldots a_{R-5}^2$ such that the least significant bits of all $R-16$ output words E_i are zero. Since we have $R-5$ degrees of freedom and only need to satisfy $R-16$ conditions, we expect to find 2^{11} different solutions. Thanks to the special structure of the equations, these solutions can be found recursively with a computation effort which is linear in the number of rounds R. Next, we use $a_1^3 \ldots a_{R-5}^3$ (which, as indicated by (1), will not affect the least significant bits) to correct the second least significant bits. We proceed this way as long as (1) holds, and eventually we will only be left with non-zero bits in the 7 most significant bits of the $R-16$ output words.

In order to eliminate the remaining non-zero bits, we could just repeat the previous procedure with different solutions for the state bits, until these non-zero bits disappear by themselves. This would require in the order of $2^{7\cdot(R-16)}$ trials. In the next section, we will show how this number can be reduced.

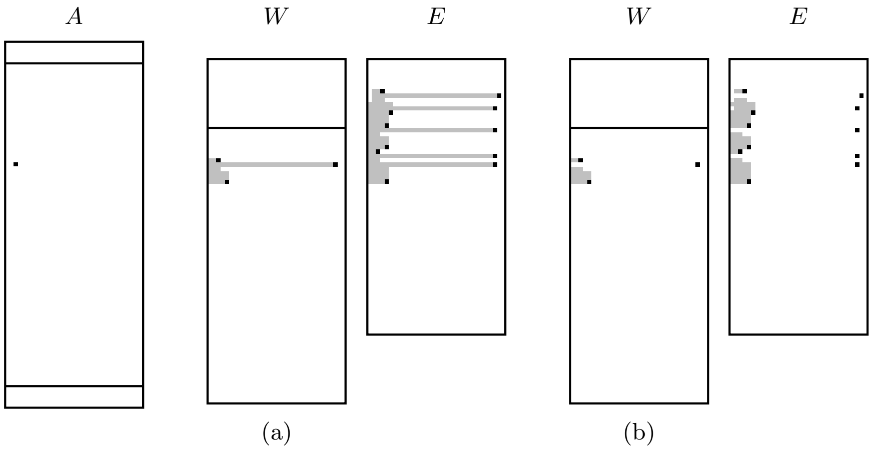


Fig. 6. Flipping state bit 29 with (a) and without (b) carries (SHA-1)

3.4 Preventing Carries

A natural way to improve the previous attack is to try to extent the property found in (1) to the case $j \geq 25$. The problem however is that the equation gets an extra term for $25 \leq j < 30$:

$$W_i^{j\cdots 0} = C_i^{j\cdots 0} - (a_i^{j+2} \ll j-25) - (f(c_i^j, a_{i-2}^{j+2}, a_{i-3}^{j+2}) \ll j) - (a_{i-4}^{j+2} \ll j)\,.$$

Hence, when trying to fix the output bits in column j, we have to make sure that this extra term at position $j-25$ does not reintroduce errors in the previously fixed columns. In order to do so, we will first try to confine the potential trouble caused by this term to a single column by preventing the propagation of carries to other columns (the idea is shown in Fig. 6). This can easily be achieved by noting that the 5 most significant bits of A_i, which we are currently trying to determine, affect the least significant part of W_i through the equation $W_i = X_i - (A_i \lll 5)$, where

$$X_i = A_{i+1} - f(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \ggg 2) - (A_{i-4} \ggg 2) - K_i\,.$$

If we now choose the 7 least significant columns of the state in beforehand in such a way that there are no zeros in the 5 least significant bits of X_i, then no carries (borrows) will appear later on when the 5 most significant bits of A_i are modified. Once these 7 columns have been determined, we start correcting the output columns for $5 \leq j < 25$ in exactly the same way as explained in the previous section.

When we arrive at $j \geq 25$, we will try to use the state bits at position $j+2$ to simultaneously correct columns j and $j-25$ of the output. This time, we have $R-5$ degrees of freedom to satisfy $2 \times (R-16)$ conditions, and hence we will still have to rely on chance for $R-27$ of these conditions. In total, we will leave $5 \times (R-27)$ uncorrected output bits in columns 25–29 and $2 \times (R-16)$ in columns 30–31. As a consequence, we will need to perform 2^c trials with $c = 2 \cdot (R-16) + 5 \cdot (R-27)$ in order for all non-zero bits to be eliminated.

3.5 Relaxing the Problem: Partial-Pseudo-Preimages

In the previous section, we had to leave a number of output bits uncorrected because of a lack of degrees of freedom in the state bits in columns 27–31. One way to create up to 10 additional degrees of freedom in each of these 5 columns is to allow the attacker to modify bits $a^j_{-4} \dots a^j_0$ and/or $a^j_{R-4} \dots a^j_R$ as well. In this case, the input and the output of the compression function will only partially match h_0 and h_1, and we call this a partial-pseudo-preimage. It is easy to see that each additional degree of freedom will reduce the cost by a factor two, i.e., if we allow $b_1 \le 25$ input bits and $b_2 \le 25$ output bits to deviate from their original target, then the computation effort of finding a partial-pseudo-preimage will be given by

$$2^c\,, \quad \text{where} \quad c = 2 \cdot (R - 16) + 5 \cdot (R - 27) - (b_1 + b_2)\,.$$

3.6 Application to SHA-1

The techniques explained for SHA-0 can be applied to SHA-1 in a relatively straightforward way. The only difference is that affected bits in W_i, with $i \ge 16$, will not only propagate to the corresponding columns in the error words, but also to the columns shifted by one position to the right. In order to compensate for this, it suffices to consider different state bits when correcting the columns, i.e., instead of using $a^{j+2}_1 \dots a^{j+2}_{R-5}$ to correct column j (and $j-25$ if $j \ge 25$), we will now use the state bits $a^{j+2}_1 \dots a^{j+2}_{11}$ and $a^{j+3}_{12} \dots a^{j+3}_{R-5}$. This works fine as long as $j < 29$. The bits $a^{j+3}_{12} \dots a^{j+3}_{R-5}$ cannot be used anymore when $j = 29$, though. Since we lose $R - 16$ degrees of freedom for fixing the last pair of columns (columns 29 and 4), the computational effort increases to:

$$2^c\,, \quad \text{where} \quad c = 3 \cdot (R - 16) + 5 \cdot (R - 27) - (b_1 + b_2)\,.$$

In addition to this, and for the same reason, we can now only fully exploit 20 additional degrees of freedom at the output, i.e., $b_2 \le 20$. We still have $b_1 \le 25$, though.

4 Preimages from Partial-Pseudo-Preimages – P^3graphs

For the discussion in this section, let's assume we are given a method to produce partial-pseudo-preimages that is faster than a method to find preimages directly.

We first discuss a number of well understood methods in Sect. 4.1 that transform such attacks on the compression function into a preimage attack on the hash function by means of meet-in-the-middle and tree building techniques. Next, in Sect. 4.2 we discuss a new method using so-called P^3graphs, that makes it possible to exploit the existence of such weaker attacks more directly.

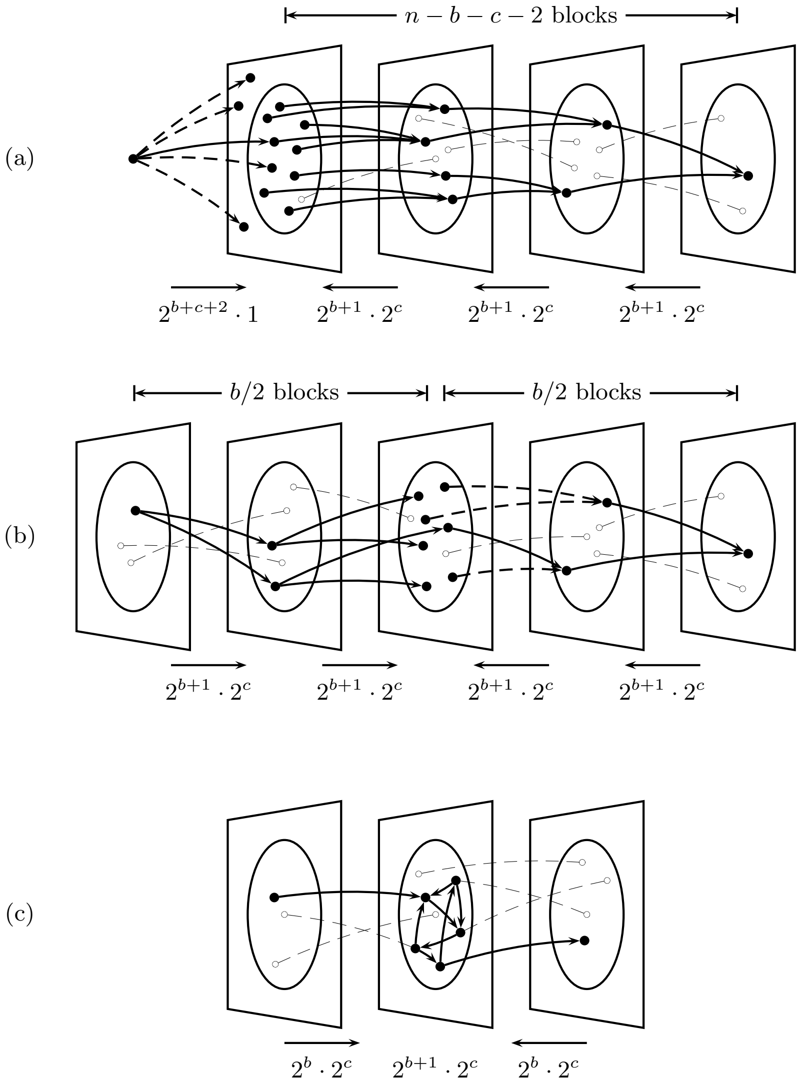


Fig. 7. Three different ways to build preimages from partial pseudo-preimages

4.1 Meet-in-the-middle and Tree Based Methods

Inverting a Davis-Meyer compression function is the problem of finding a pair (h, m) such that $g(h, m) + h$ equals a given digest d. It was shown that no black-box attack can give a preimage faster than essentially 2^n [3, 27]. Inverting a Merkle-Damgård hash function is the problem of, given an initial chaining input h_0, finding an (almost arbitrarily large) number x of message blocks $m_0 \ldots m_x$ such that h_x equals a given digest d.

In the following, we assume that a part of the chaining input (say $n - b_1$ out of the n bits) can be chosen by the attacker, or in other words: the attacker can control all but b_1 bits of the chaining input (always the same $n - b_1$ bits). Let's further assume that a partial preimage attack on the compression function (of cost 2^c) has the property that a preimage can be found where all but b_2 out of n bits match the targeted digest d (again always the same $n - b_2$ bits). In addition to the parameters b_1 and b_2 introduced in Sect. 3.5, we will denote the number of bit positions of the chaining variable which can be controlled both from the input and from the output by $n - b$. All the following methods yield a preimage of the hash function for any given digest d

- **Meet-in-the-middle approach 1.** A basic unbalanced meet-in-the-middle approach that does not take advantage of the b bits that overlap has runtime $2^{(n+c)/2+1}$ and memory costs of $2^{(n-c)/2}$. The balanced case appeared already in [9], memoryless variants appear to have been first proposed in [23, 28].
- **Meet-in-the-middle approach 2.** By using the fact that both in forwards, and backwards direction, only b bits need to meet, the runtime requirement improves to $2^{b/2+c_1} + 2^{b/2+c_2}$, where c_1 denotes the cost of a partial-preimage attack (the forward part, if no compression function attack is available, a brute force attack with this property has cost 2^{n-b}), and c_2 denotes the cost of the pseudo-preimage attack (this is equivalent to calling the partial-pseudo-preimage attack 2^b times at the cost of 2^{b+c}). The total runtime is hence $2^{3b/2+c+1}$, the memory requirement is 2^b.
- **Layered Tree method due to Leurent, see Fig. 7(a)**. In [17] the following tree method was proposed. Starting from the target hash d, produce two different pseudo preimages with cost 2^{b+c+1}. As a next step, produce four different pseudo preimages with the same cost that target both new target chaining values. This process is continued for $n - b - c - 2$ blocks and needs about $2^{n-b-c-1}$ of storage. For a fixed length preimage, only the last layer of the tree can be used for random trials in the forward direction, amounting to 2^{b+c+2} trials. Variants with a different branch number, or with less restrictions on the way the tree grows are thinkable [17].
- **Alternative Backward-Forward Tree method, see Fig. 7(b)**. Similar to the approach above, one could let the tree grow in the backward direction for $b/2$ blocks, regardless of the time complexity of the compression function attack. In the forward direction we rely on using the partial-pseudo-preimage on the compression function of cost 2^c again, now having to call it 2^b times to have a partial-preimage. Using this, the tree grows in the forwards direction in exactly the same manner as in the backwards direction. Because of the

birthday effect, both trees have at least one connection with high probability. The total runtime is $b \cdot 2^{b+c+1}$, the memory requirement is $2^{b/2}$.

– **Tree method due to Mendel and Rijmen, see Fig. 7(c)**. In [22] a tree-based method was proposed that has the same runtime and memory requirements as the new graph-based method we are about to introduce in the following section.

4.2 A Graph Based Approach

The meet-in-the-middle method discussed above requires the generation of many partial-preimages for the first part of the preimage and many pseudo-preimages for the second part of the preimage. The new method based on random directed graphs we are about to introduce allows to reduce the number of partial-preimages needed at the beginning and pseudo-preimages needed at the end to 1, at the cost of a number of partial-pseudo-preimages (each 2^c) in between. Hence the name $\mathsf{P}^3\mathsf{graph}$ method, see also Fig. 7(c). We first outline the proposed method, and give time and memory complexities. Afterwards we discuss and compare it with other methods.

Edges of P^3graph: Using a partial pseudo preimage algorithm, generate 2^{b+1} tuples $(h_{(i)}, m_{(i)})$, at cost 2^{b+c+1}. All these tuples, which map $h_{(i)}$ to $f(h_{(i)}, m_{(i)})$, can be seen as the 2^{b+1} edges of a directed graph consisting of 2^b nodes. As explained in Appendix A, we expect the majority of those nodes to be part of a large densely interconnected component.

First message block, forward direction: Using the partial preimage generation method, generate a single tuple (h_0, m_0) that hits this component. The expected work is in the order of 2^{b+c}.

Last message block, backward direction: Also here, generate a single tuple (h_x, m_x) such that $f(h_x, m_x) = \mathsf{d}$ and that h_x falls into the interconnected part of the graph. The expected work is again in the order of: 2^{b+c}.

Connection: What remains to be found is a connection (a path) between these nodes (the entry node and the exit node) in the graph. Given the number of edges in the graph, such a path is very likely to exist, as we discuss in detail in Appendix A. Total expected work: $2^{b+c+1} + 2^{b+c} + 2^{b+c} = 2^{b+c+2}$

On exploiting precomputation. The computations for constructing the first message block and the $\mathsf{P}^3\mathsf{graph}$ do not need to be repeated when attacking a different digest. The effort for every additional preimage attack is only 2^{b+c}.

4.3 Discussion

There are a number of useful and distinctive properties of the $\mathsf{P}^3\mathsf{graph}$ method. Firstly, the graph approach does not impose any structure on the connections of partial-pseudo-preimages, which is an intuitive explanation of the efficiency again compared to the L-Tree and the BF-Tree methods. Secondly, the $\mathsf{P}^3\mathsf{graph}$ is friendlier towards precomputation: Whereas the full $\mathsf{P}^3\mathsf{graph}$ (potentially in such a way that the IV of the hash function is one of the nodes) can be precomputed, it is not possible to precompute the backwards tree for the L-Tree and the BF-Tree method. Another advantage of the $\mathsf{P}^3\mathsf{graph}$ method over all other known methods is that paths (and hence preimages) of almost any length have high probability to exist. There is no upper limit, the lower bound is discussed in Appendix A. This property will be useful when dealing with the padding in a preimage attack on the hash function (see Sect 5.1).

One drawback of the $\mathsf{P}^3\mathsf{graph}$ method can be the higher memory requirements. Storage requirements for all the edges is exponential in the number of bits b that can not be controlled. Hence the runtime gain of the $\mathsf{P}^3\mathsf{graph}$ method is useful in practice if the compression function attack allows to choose a reasonable small b. The $\mathsf{P}^3\mathsf{graph}$ method allows time/memory tradeoffs that resemble e.g., the BF-Tree method. Space constraints do not allow us to discuss them here. In Table 1 we summarize and compare the meet in the middle approach with the $\mathsf{P}^3\mathsf{graph}$ method.

5 Putting Everything Together

We have now set the state to talk about the security margin of the SHA-0 and SHA-1 hash function against the new cryptanalytic methods. We do this by combining the compression function attack from Sect. 3 and the $\mathsf{P}^3\mathsf{graph}$ method from Sect. 4.

5.1 Padding

So far, we neglected the fact that in a preimage attack on SHA-0 and SHA-1, the padding fixes a part of the input message of the last message block. Hence, without being able to cope with such a restriction, our attack would only be a second preimage attack, but not a preimage attack. We discuss here several possibilities to produce a correctly padded last message block without a first preimage.

- **Restrict the degrees of freedom in the compression function attack:** In order to fix a particular value for the message length, at least the last

Table 1. Comparison of the meet-in-the-middle approach, various tree approaches, and the $\mathsf{P}^3\mathsf{graph}$ method. All numbers are exponents of base 2.

	MITM2	L(ayered)-Tree	BF-Tree	MR-Tree	$\mathsf{P}^3\mathsf{graph}$
total work	$3b/2+c+1$	$b+c+1+log_2(n-b-c)$	$b+log_2(b)+c+1$	$b+c+2$	$b+c+2$
total mem.	b or less	$n-b-c-1$	$b/2$	$b+1$	$b+1$
onl. work	$b+c$	-	-	$b+c$	$b+c$
offl. work	$2b+c$	-	-	$b+c+log_2(3)$	$b+c+log_2(3)$
memory	b	-	-	$b+1$	$b+1$
flexible len.	no	no	no	no	yes

65 bits of the last message block need to be fixed. Among them are 25 bits whose freedom is needed in the compression function attack (for both SHA-0 and SHA-1), hence fixing them results (without further optimizations) in a slowdown of the compression function attack by up to a factor of 2^{25}. In detail, these bits are M_{14}^{0}, $M_{15}^{0\ldots4,24\ldots31}$ and $M_{16}^{0\ldots4,24\ldots31}$.

- **Expandable messages:** By making sure that every message length can be constructed after the compression function attacks have been performed, almost no additional degrees of freedom need to be spent for a correct padding. Using any of the following methods will hence return preimages of uncontrollable length. The only two property that the compression function attack needs to have, are as follows. Firstly, to make sure that the end of the message (before the length encoding, i.e., the LSB of M_{13}) is a '1'. Secondly, make sure that the length is a exact multiple of the block length, i.e., fix the last nine bits of M_{15} to '110111111' (447). In total ten bits need to be fixed for this, which will result (without further optimizations) in a slowdown of the compression function attack by a factor 2^6. In detail, the six crucial bits are M_{14}^{0} and $M_{16}^{0\ldots4}$. Possibilities to construct expandable messages are as follows.
  - **Multicollisions:** As soon as the compression function attack has a complexity slightly above the birthday bound ($2^{n/2+log_2(n)}$), the multicollision idea [12] can be used to construct expandable messages [15] without being the bottleneck.
  - **Flexibility of the P^3graph method (cycles)**: In the random directed graph used in the P^3graph method of Sect. 4.2, we expect to have many cycles, also on the path between entry- and exit node. As detailed in Appendix A, we hence expect to find paths of any length longer than some lower bound that connect any entry- and exit node with high probability.

5.2 Summary of Attacks

From Sect. 3 we learn that $b_1 = b_2 = 25$ is a straight-forward choice for the case of SHA-0. Since the method allows us to pick the same bit positions, we also have $b = 25$. Since $b_2 \leq 20$ for SHA-1, we will have to restrict ourselves to $b = 20$ in this case. Note that for seriously reduced SHA-0 and SHA-1, less degrees of freedom are of use in the compression function attack, and hence b can be smaller. A quick check in Table 1 will convince the reader that memory requirements will not be a problem in the practical implementation of such an attack, even with the most time efficient P^3graph method.

In order to illustrate our results we consider SHA-0 and SHA-1 reduced to concrete numbers of steps, and give attack complexities in Figure 8. We combine the attacks on the compression function as given in Sect. 3 with the different generic ways of turning them into a preimage attack as outlined in Sect. 4.2. In our implementation of this attack the memory requirements are negligible. Additionally, we also give attack complexities in Table 2. For both SHA-0 and SHA-1, the number of steps for which we list results are chosen as follows. To compare (lack of) resistance against the new attack of similarly reduced

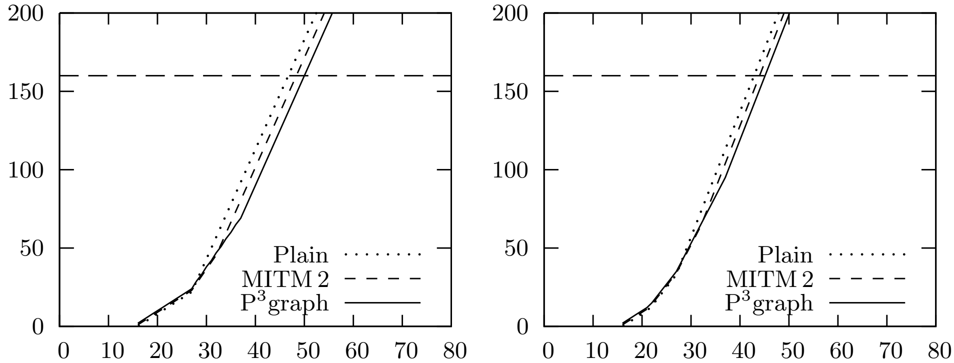


Fig. 8. Complexities of second preimage attacks against reduced SHA-0 (left) and SHA-1 (right). The line 'Plain' refers to a direct preimage attack using only a single block. The line 'MITM 2' refers to a meet-in-the-middle approach where partial-preimages in the forward direction are combined with pseudo-preimages in the backwards direction. The line '$\mathsf{P^3graph}$' refers to the new graph based method.

primitives, we pick 32 steps in all cases. Additionally, we give results for the highest number of steps for which the attack would be better than the birthday bound and an actual brute force attack, respectively. Our approach takes less than 2^{160} hash evaluations for SHA-0 reduced to up to 50 steps and for SHA-1 reduced to up to 45 steps. Note that inverting the hash function also implies the ability to construct a fixed point.

6 Conclusions and Outlook

The first method to construct preimages for SHA-0 and SHA-1 reduced to a nontrivial number of steps (up to 50 out of 80) is presented. The impossible message approach we proposed exploits weak diffusion properties in the step transformation and in the message expansion, which allows to divide the work and consider only one or a small number of column at a given time. Both, the impossible message approach, and the $\mathsf{P^3graph}$ we introduced to efficiently transform attacks on the compression function to attacks on the hash function, are rather generic and await to be applied to other settings and hash functions as well.

Our results shed some light on the security margin offered by SHA-0 and SHA-1 when only preimage attacks are of a concern. However, several aspects of this work suggest that the security margin might be smaller. Let's compare the result of this work on cryptanalytic preimage attacks to the situation of collision search attacks in 2004 and early 2005:

– **Step-reduced variants:** Work on SHA-1 resulted in theoretical collision attacks for up to 58 steps [1, 31]. Our preimage attacks cover slightly less steps but are on a comparable magnitude.

Table 2. Exemplification of new preimage attacks on reduced SHA-0 (left table) and SHA-1 (right table). Efforts are expressed in terms of time complexity; memory and communication costs can be considered negligible. For ideal building blocks, all these attacks would require a 2^{160} effort. For simplification, the small constant factor between the numbers given here and a naive brute force search is neglected. We give the total runtime for attacking the first target digest; attacks on subsequent targets will be faster.

type of attack	building block	steps	b	effort with new attack	building block	steps	b	effort with new attack
inv. compression f.	SHA-0	32	25	2^{32}	SHA-1	32	20	2^{53}
inv. compression f.	SHA-0	38	25	2^{74}	SHA-1	35	20	2^{77}
inv. compression f.	SHA-0	50	25	2^{158}	SHA-1	45	20	2^{157}
2nd preimage of hash	SHA-0	32	12	2^{47}	SHA-1	32	10	2^{65}
2nd preimage of hash	SHA-0	38	25	2^{76}	SHA-1	34	14	2^{77}
2nd preimage of hash	SHA-0	49	25	2^{163}	SHA-1	45	20	2^{159}
preimage of hash	SHA-0	37	25	2^{75}	SHA-1	34	17	2^{80}
preimage of hash	SHA-0	49	25	2^{159}	SHA-1	44	20	2^{157}

- **Degrees of freedom:** Whereas in the most recent collision search attacks on SHA-1 the availability of degrees of freedom is the limiting factor for further improvements, this was of no concern in earlier work. The fact that not all degrees of freedom are used in our new preimage attacks suggests that further improvements are possible.
- **Sensitivity for different choices of rotation constants:** The state update transformation of SHA-0 and SHA-1 uses the fixed set of rotation constants $(5, -2)$. A study of the effect of different choices of rotation constants on earlier collision search strategies [26] concluded that already a slightly different choice would impact the performance significantly, although in a complex way. In our attack, we observe a similar situation: The attack complexity directly depends on the used rotation constants and would be lower or higher, depending on the actual choice. The most recent collision search attacks on SHA-1 do not show such a strong dependency on the choice of rotation constants. Again, this suggests that further improvements on the preimage attack presented in this paper is an interesting open problem.

Acknowledgements

The authors wish to thank Florian Mendel, Adi Shamir, Yiqun Lisa Yin and the anonymous reviewers for their useful comments. The work in this paper has been supported in part by the Fund for Scientific Research (FWO), the Chaire France Telecom pour la sécurité des réseaux, the Secure Information Technology Center-Austria (A-SIT), by the Austrian Science Fund (FWF), project P19863, and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.
2. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In A. Menezes and S. A. Vanstone, editors, *CRYPTO*, volume 537 of *LNCS*, pages 2–21. Springer, 1990.
3. J. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In M. Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 320–335. Springer, 2002.
4. B. Bollobás. *Random Graphs*. Academic Press, 1985.
5. B. Bollobás. *Modern Graph Theory*. Springer, 1998.
6. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.
7. C. De Cannière, F. Mendel, and C. Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 56–73. Springer, 2007.
8. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
9. W. Diffie and M. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.
10. H. Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.
11. P. Erdös and A. Rènyi. On random graphs. *Publicationes Mathematicae 6*, pages 290–297, 1959.
12. A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
13. A. Joux and T. Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In A. Menezes, editor, *CRYPTO*, volume 4622 of *LNCS*, pages 244–263. Springer, 2007.
14. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *LNCS*, pages 183–200. Springer, 2006.
15. J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.
16. M. Lamberger, N. Pramstaller, C. Rechberger, and V. Rijmen. Second Preimages for SMASH. In M. Abe, editor, *CT-RSA*, volume 4377 of *LNCS*, pages 101–111. Springer, 2007.
17. G. Leurent. MD4 is Not One-Way. In K. Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, March 26-28, 2007, to appear*, LNCS. Springer, 2008.
18. F. Mendel, N. Pramstaller, and C. Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In K. Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 224–234. Springer, 2008.
19. F. Mendel, N. Pramstaller, C. Rechberger, M. Kontac, and J. Szmidt. Cryptanalysis of the GOST Hash Function. In D. Wagner, editor, *Proceedings of CRYPTO 2008, to appear*, LNCS. Springer, 2008.

20. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. The Impact of Carries on the Complexity of Collision Attacks on SHA-1. In M. J. B. Robshaw, editor, *FSE*, volume 4047 of *LNCS*, pages 278–292. Springer, 2006.
21. F. Mendel, C. Rechberger, and V. Rijmen. Update on SHA-1. Rump Session of CRYPTO 2007, 2007.
22. F. Mendel and V. Rijmen. Weaknesses in the HAS-V Compression Function. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 335–345. Springer, 2007.
23. H. Morita, K. Ohta, and S. Miyaguchi. A Switching Closure Test to Analyze Cryptosystems. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *LNCS*, pages 183–193. Springer, 1991.
24. National Institute of Standards and Technology. NIST's Policy on Hash Functions, 2006. Available online at `http://csrc.nist.gov/groups/ST/hash/policy.html`.
25. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
26. N. Pramstaller, C. Rechberger, and V. Rijmen. Impact of Rotations in SHA-1 and Related Hash Functions. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *LNCS*, pages 261–275. Springer, 2005.
27. B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.
28. J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search. New Results and Applications to DES. In G. Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 408–413. Springer, 1989.
29. C. Rechberger and V. Rijmen. On Authentication with HMAC and Non-random Properties. In S. Dietrich and R. Dhamija, editors, *Financial Cryptography*, volume 4886 of *LNCS*, pages 119–133. Springer, 2007.
30. C. Rechberger and V. Rijmen. New Results on NMAC/HMAC when Instantiated with Popular Hash Functions. *Journal of Universal Computer Science (JUCS), Special Issue on Cryptography in Computer System Security*, 14(3):347–376, 2008.
31. V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.
32. M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *LNCS*, pages 1–22. Springer, 2007.
33. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
34. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
35. H. Yu, G. Wang, G. Zhang, and X. Wang. The Second-Preimage Attack on MD4. In Y. Desmedt, H. Wang, Y. Mu, and Y. Li, editors, *CANS*, volume 3810 of *LNCS*, pages 1–12. Springer, 2005.

A Some Useful Properties of Random Graphs

In this appendix, we briefly review some properties of random graphs which are relevant to the graph based approach proposed in Sect. 4.2. For a more rigorous and comprehensive treatment of random graph theory we refer to [4, 11] and [5, Chapt. VII.5].

A.1 Following Edges in a Random Directed Graph

Let G be a large directed graph consisting of n nodes and $m = c \cdot n$ randomly selected edges. On average, each node has c outgoing edges, and we denote the probability that a given ordered pair of nodes is connected by an edge by:

$$p_c = \frac{m}{n^2} = \frac{c}{n} \,.$$

Let us now study what happens when we start from an arbitrary node a and construct sets of nodes $S_0, S_1, S_2, \ldots$ where $S_0 = \{a\}$, and S_i contains all nodes that can be reached from a in exactly i hops. If we eventually end up with an empty set, the initial node a is called a "dying" node. In the opposite case, a is said to "explode". Clearly, if there exists an edge from a to b, and b is an exploding node, then a must be exploding as well. Conversely, a node a can only die if none of the n nodes in the graph are both connected to a and exploding. Hence, the probability p_e that a node explodes must satisfy:

$$1 - p_e = (1 - p_c \cdot p_e)^n \approx e^{-c \cdot p_e} \,.$$

From this expression we can deduce that p_e must necessarily be 0 as long as $c \leq 1$. However, when $c > 1$, the equation $1 - x = e^{-c \cdot x}$ does have a non-zero (and positive) solution, which we will refer to as $\gamma(c)$.[4]

Assuming that the sets S_i reach some moderately large size (i.e., a does not die), we can write a simple recursive relation between the expected sizes $E(|S_i|)$ of successive sets by computing the probability that an arbitrary node is connected to at least one node of S_i:

$$E(|S_{i+1}|) = n \cdot \left[1 - (1 - p_c)^{E(|S_i|)}\right] \approx n \cdot \left[1 - e^{-c \cdot E(|S_i|)/n}\right] . \tag{2}$$

Note that we can apply the same reasoning to obtain an almost identical recursive relation between successive values of $E(|S_0 \cup S_1 \cdots S_i|)$. By filling in $i = \infty$, we find that the expected size of the sets converges to:

$$E(|S_\infty|) \approx E(|S_0 \cup S_1 \cdots S_\infty|) \approx n \cdot \gamma(c) \,.$$

A.2 Connecting Two Given Nodes

In the previous section, we argued that a node a explodes with probability $p_e = \gamma(c)$, and that a fraction $\gamma(c)$ of all nodes can be reached from it if it does. Similarly, if a dies, it can be shown that only a negligible fraction of nodes will be reached. The probability p_p that two given nodes a and b are connected by a path is hence:

$$p_p = \gamma(c)^2 \,.$$

In the context of the attack proposed in this paper, we are interested in the expected number of random edges $\overline{m}$ that need to be added to a graph in order

[4] One can show that $\gamma(c) = 1 + W(-c \cdot e^{-c})/c$, where $W(x)$ is Lambert's W function.

to find a path between two given nodes a and b. Suppose our current graph has $m > n$ edges. In that case we know that with probability $1-\gamma(m/n)^2$ there will be no path between a and b, in which case we will need at least one more edge. Repeating this reasoning, we find;

$$\overline{m} \approx n + \sum_{m=n}^{n^2} \left[1 - \gamma(m/n)^2\right] .$$

We can approximate this sum by an integral, and after a few changes of variables, we eventually obtain:

$$\begin{aligned} \overline{m} &\approx n + n \cdot \int_1^\infty \left[1 - \gamma(c)^2\right] dc \\ &= n + n \cdot \int_0^1 \left(1 - \gamma^2\right) \cdot \frac{dc}{d\gamma}\, d\gamma \\ &= 2 \cdot n . \end{aligned}$$

This result, which states that, in order to connect two given nodes, we need on average twice as many edges as nodes (i.e., $c = 2$), is the main property used in Sect. 4.2.

A.3 Path Lengths

If we want to apply our graph based attack to a hash function which includes the message length in the padding block, then we not only need to make sure that there exists a path between two given nodes; we would also like to know in advance how long this path will be.

In order to estimate how many nodes can be reached for a fixed path length, we need to solve the recursive relation of (2). A closed form solution probably does not exist, but we can find a very good approximation:

$$E(|S_i|) \approx n \cdot \gamma \cdot \frac{\left[\alpha^{2\cdot(i-\delta)} + 1\right]^\beta}{\alpha^{(i-\delta)} + 1} ,$$

where $\alpha = c \cdot (1-\gamma)$, $\alpha^{2\cdot\beta-1} = c$, and $n \cdot \gamma \cdot c^{-\delta} = 1$. For $c = 2$, we find that $\gamma = 0.80$, $\alpha = 0.41$, $\beta = 0.12$, and

$$\delta = \frac{1}{\log_2 c} \cdot (\log_2 n + \log_2 \gamma) = \log_2 n - 0.33 .$$

We can now compute the minimal path length l for which we expect that S_l includes all reachable nodes (i.e., $S_l = S_\infty$). By solving the inequality $E(|S_\infty|) - E(|S_l|) < 1$, we obtain:

$$l > \left[\frac{1}{\log_2 \alpha} - \frac{1}{\log_2 c}\right] \cdot (\log_2 n + \log_2 \gamma) = 1.77 \cdot \log_2 n - 0.58 .$$

In other words, if we are given a random graph with n nodes and $2 \cdot n$ edges, and if this graph connects two arbitrary nodes a and b (this occurs with probability $\gamma^2 = 0.63$), then we expect to find paths from a to b of length l for any l exceeding $1.77 \cdot \log_2 n$.

B Proof-of-concept Examples

As a proof-of-concept, we give examples of an implementation of the described methods. We chose two examples. The first is a preimage for the 33-step SHA-0 compression function. The second is also a second preimage of a (roughly) 1KB ASCII text for the 31-step SHA-0 hash function, using the $\mathsf{P}^3\mathsf{graph}$ method.

B.1 A preimage for the 33-step compression function of SHA-0

As a proof-of-concept, we consider the compression function of SHA-0 reduced to 33 steps. In Figure 9 we give a preimage for the all-1 output. $A_{-4} \ldots A_0$ and $W_0 \ldots W_{15}$ represent the input to the compression function. Computing $A_{-4} + A_{29} \ldots A_0 + A_{33}$ results in the all-1 output.

i	A_i	W_i
-4:	00110111111111111111111111111100	
-3:	11010111111111111111111111111100	
-2:	00100111111111111111111111111100	
-1:	00100111111111111111111111111111	
0:	10110111111111111111111111111111	10100111011111011000111010001001
1:	00100010000000000000100000010110	01100111100011001010011000011011
2:	11000010000100000001000100111011 0	01010000100000010111100010000111
3:	11100000100001000010000000111101 10	01000000110000101100010010110001 1
4:	00110101000000000000000101100100	10110010111111010101011101011001
5:	01000100000000000000000000001100	10100010011110010111101001010111
6:	10110110000000000000000000111010	11011111101101110110011001001001
7:	01100111000000000000000000001110	00001111111110110111010000110011
8:	00011100000000000000000000011000	10000111001111011000001011111100
9:	10100100000000000000000000000000	01000000111111011000011010001011
10:	11100111000000000000000001000001	10011100101111010111111010000011
11:	10100010100000000000000001101001	10101101000000111111101001001011
12:	00010010010001101000000100100001	01011101010110010110110100111101
13:	00110001001011000000101011111110	00011011111010010011001011011001
14:	00101110011011010000110001001000	00000011001110111111110010011010
15:	11101101100111111111111110010000	11100001000001101011110110000010
16:	10100101000000101100100101011010	01101011001010000100011000101011
	...	...
28:	01110111001110111011010101110100	11111100111010011110011000110001
29:	11001000000000000000000000000011	01110110101111001110011000100110
30:	00101000000000000000000000000011	11011100010011000000000000111010
31:	11011000000000000000000000000011	10000010111111000100100010100100
32:	11011000000000000000000000000000	11011011010101110010011011100100
33:	01001000000000000000000000000000	

Fig. 9. A preimage of the all-1 output for the 33-step SHA-0 compression function

```
0000000: 416c 6963 6520 7761 7320 6265 6769 6e6e  Alice was beginn
0000010: 696e 6720 746f 2067 6574 2076 6572 7920  ing to get very
0000020: 7469 7265 6420 6f66 2073 6974 7469 6e67  tired of sitting
0000030: 2062 7920 6865 7220 7369 7374 6572 206f   by her sister o
0000040: 6e20 7468 6520 6261 6e6b 2c20 616e 6420  n the bank, and
0000050: 6f66 2068 6176 696e 6720 6e6f 7468 696e  of having nothin
0000060: 6720 746f 2064 6f3a 206f 6e63 6520 6f72  g to do: once or
0000070: 2074 7769 6365 2073 6865 2068 6164 2070   twice she had p
0000080: 6565 7065 6420 696e 746f 2074 6865 2062  eeped into the b
0000090: 6f6f 6b20 6865 7220 7369 7374 6572 2077  ook her sister w
00000a0: 6173 2072 6561 6469 6e67 2c20 6275 7420  as reading, but
00000b0: 6974 2068 6164 206e 6f20 7069 6374 7572  it had no pictur
00000c0: 6573 206f 7220 636f 6e76 6572 7361 7469  es or conversati
00000d0: 6f6e 7320 696e 2069 742c 2060 616e 6420  ons in it, ‘and
00000e0: 7768 6174 2069 7320 7468 6520 7573 6520  what is the use
00000f0: 6f66 2061 2062 6f6f 6b2c 2720 7468 6f75  of a book,’ thou
0000100: 6768 7420 416c 6963 6520 6077 6974 686f  ght Alice ‘witho
0000110: 7574 2070 6963 7475 7265 7320 6f72 2063  ut pictures or c
0000120: 6f6e 7665 7273 6174 696f 6e3f 2720 536f  onversation?’ So
0000130: 2073 6865 2077 6173 2063 6f6e 7369 6465   she was conside
0000140: 7269 6e67 2069 6e20 6865 7220 6f77 6e20  ring in her own
0000150: 6d69 6e64 2028 6173 2077 656c 6c20 6173  mind (as well as
0000160: 2073 6865 2063 6f75 6c64 2c20 666f 7220   she could, for
0000170: 7468 6520 686f 7420 6461 7920 6d61 6465  the hot day made
0000180: 2068 6572 2066 6565 6c20 7665 7279 2073   her feel very s
0000190: 6c65 6570 7920 616e 6420 7374 7570 6964  leepy and stupid
00001a0: 292c 2077 6865 7468 6572 2074 6865 2070  ), whether the p
00001b0: 6c65 6173 7572 6520 6f66 206d 616b 696e  leasure of makin
00001c0: 6720 6120 6461 6973 792d 6368 6169 6e20  g a daisy-chain
00001d0: 776f 756c 6420 6265 2077 6f72 7468 2074  would be worth t
00001e0: 6865 2074 726f 7562 6c65 206f 6620 6765  he trouble of ge
00001f0: 7474 696e 6720 7570 2061 6e64 2070 6963  tting up and pic
0000200: 6b69 6e67 2074 6865 2064 6169 7369 6573  king the daisies
0000210: 2c20 7768 656e 2073 7564 6465 6e6c 7920  , when suddenly
0000220: 6120 5768 6974 6520 5261 6262 6974 2077  a White Rabbit w
0000230: 6974 6820 7069 6e6b 2065 7965 7320 7261  ith pink eyes ra
0000240: 6e20 636c 6f73 6520 6279 2068 6572 2e20  n close by her.
0000250: 5468 6572 6520 7761 7320 6e6f 7468 696e  There was nothin
0000260: 6720 736f 2056 4552 5920 7265 6d61 726b  g so VERY remark
0000270: 6162 6c65 2069 6e20 7468 6174 3b20 6e6f  able in that; no
0000280: 7220 6469 6420 416c 6963 6520 7468 696e  r did Alice thin
0000290: 6b20 6974 2073 6f20 5645 5259 206d 7563  k it so VERY muc
00002a0: 6820 6f75 7420 6f66 2074 6865 2077 6179  h out of the way
00002b0: 2074 6f20 6865 6172 2074 6865 2052 6162   to hear the Rab
00002c0: 6269 7420 7361 7920 746f 2069 7473 656c  bit say to itsel
00002d0: 662c 2060 4f68 2064 6561 7221 204f 6820  f, ‘Oh dear! Oh
00002e0: 6465 6172 2120 4920 7368 616c 6c20 6265  dear! I shall be
00002f0: 206c 6174 6521 2720 2877 6865 6e20 7368   late!’ (when sh
```

Fig. 10. 31-round SHA-0: original message (part 1)

```
0000300: 6520 7468 6f75 6768 7420 6974 206f 7665  e thought it ove
0000310: 7220 6166 7465 7277 6172 6473 2c20 6974  r afterwards, it
0000320: 206f 6363 7572 7265 6420 746f 2068 6572   occurred to her
0000330: 2074 6861 7420 7368 6520 6f75 6768 7420   that she ought
0000340: 746f 2068 6176 6520 776f 6e64 6572 6564  to have wondered
0000350: 2061 7420 7468 6973 2c20 6275 7420 6174   at this, but at
0000360: 2074 6865 2074 696d 6520 6974 2061 6c6c   the time it all
0000370: 2073 6565 6d65 6420 7175 6974 6520 6e61   seemed quite na
0000380: 7475 7261 6c29 3b20 6275 7420 7768 656e  tural); but when
0000390: 2074 6865 2052 6162 6269 7420 6163 7475   the Rabbit actu
00003a0: 616c 6c79 2054 4f4f 4b20 4120 5741 5443  ally TOOK A WATC
00003b0: 4820 4f55 5420 4f46 2049 5453 2057 4149  H OUT OF ITS WAI
00003c0: 5354 434f 4154 2d50 4f43 4b45 542c 2061  STCOAT-POCKET, a
00003d0: 6e64 206c 6f6f 6b65 6420 6174 2069 742c  nd looked at it,
00003e0: 2061 6e64 2074 6865 6e20 6875 7272 6965   and then hurrie
00003f0: 6420 6f6e 2c20 416c 6963 6520 7374 6172  d on, Alice star
0000400: 7465 6420 746f 2068 6572 2066 6565 742c  ted to her feet,
0000410: 2066 6f72 2069 7420 666c 6173 6865 6420   for it flashed
0000420: 6163 726f 7373 2068 6572 206d 696e 6420  across her mind
0000430: 7468 6174 2073 6865 2068 6164 206e 6576  that she had nev
0000440: 6572 2062 6566 6f72 6520 7365 656e 2061  er before seen a
0000450: 2072 6162 6269 7420 7769 7468 2065 6974   rabbit with eit
0000460: 6865 7220 6120 7761 6973 7463 6f61 742d  her a waistcoat-
0000470: 706f 636b 6574 2c20 6f72 2061 2077 6174  pocket, or a wat
0000480: 6368 2074 6f20 7461 6b65 206f 7574 206f  ch to take out o
0000490: 6620 6974 2c20 616e 6420 6275 726e 696e  f it, and burnin
00004a0: 6720 7769 7468 2063 7572 696f 7369 7479  g with curiosity
00004b0: 2c20 7368 6520 7261 6e20 6163 726f 7373  , she ran across
00004c0: 2074 6865 2066 6965 6c64 2061 6674 6572   the field after
00004d0: 2069 742c 2061 6e64 2066 6f72 7475 6e61   it, and fortuna
00004e0: 7465 6c79 2077 6173 206a 7573 7420 696e  tely was just in
00004f0: 2074 696d 6520 746f 2073 6565 2069 7420   time to see it
0000500: 706f 7020 646f 776e 2061 206c 6172 6765  pop down a large
0000510: 2072 6162 6269 742d 686f 6c65 2075 6e64   rabbit-hole und
0000520: 6572 2074 6865 2068 6564 6765 2e20 496e  er the hedge. In
0000530: 2061 6e6f 7468 6572 206d 6f6d 656e 7420   another moment
0000540: 646f 776e 2077 656e 7420 416c 6963 6520  down went Alice
0000550: 6166 7465 7220 6974 2c20 6e65 7665 7220  after it, never
0000560: 6f6e 6365 2063 6f6e 7369 6465 7269 6e67  once considering
0000570: 2068 6f77 2069 6e20 7468 6520 776f 726c   how in the worl
0000580: 6420 7368 6520 7761 7320 746f 2067 6574  d she was to get
0000590: 206f 7574 2061 6761 696e 2e0a             out again..
```

Fig. 11. 31-round SHA-0: original message (part 2)

```
0000000: 6093 e793 8844 423f cf3e 4140 3479 5078  `....DB?.>A@4yPx
0000010: f8ac 0a92 7e6a 1956 d8b7 b004 1bf9 027f  ....~j.V........
0000020: 13fd 7b20 5cbd 783c 9b3d 78d2 e0bd 8106  ..{ \.x<.=x.....
0000030: fee5 2a1d 8efe 23eb 6bd8 7621 354f 0c9c  ..*...#.k.v!5O..
0000040: 9b86 3bbf 6469 db87 b11d 9195 707d 3f5a  ..;.di......p}?Z
0000050: 277b 582e 44fa 9440 a57c be61 14bc 7c39  '{X.D..@.|.a..|9
0000060: aabc 785e 3c7d 85ef 35bd 855d 1b7d 84fd  ..x^<}..5..].}..
0000070: a7d6 c497 a55a d1ae 21ea 5210 19cc f5e1  .....Z..!.R.....
0000080: b6a5 86d7 e20e 085d e7ab ab81 dd74 ffad  .......].....t..
0000090: 6a33 7421 b5cf 5fa2 c709 48b3 836d 6f2a  j3t!.._...H..mo*
00000a0: 8d3d 7e50 eefd 793c 2cbd 84ea d83d 78bc  .=~P..y<,....=x.
00000b0: 7d7b 64a9 483c 18f3 f559 a0d5 bf69 d5f8  }{d.H<...Y...i..
00000c0: 5e7d 920f 9cbe 10a2 0d5d 5bb1 453d 7b31  ^}......][.E={1
00000d0: d03d 7f7f fe6d 019b 5fa4 fed5 fbf5 79dd  .=...m.._.....y.
00000e0: 37bd 7ced ddfd 79aa 18fd 7da7 063d 8622  7.|...y...}..=."
00000f0: ece1 65d6 0372 499e 9c7c 8472 5267 8c88  ..e..rI..|.rRg..
0000100: fa9e 8747 255d a7e9 cafd 73dd b87d 3785  ...G%]....s..}7.
0000110: b63d 3c42 2e35 3292 771b 690c a41b 77f1  .=<B.52.w.i...w.
0000120: abfd 84fa d93d 8646 9c3d 7774 b23d 7c79  .....=.F.=wt.=|y
0000130: aef9 1db8 c192 413e d8ef 6d8b b39e f536  ......A>..m....6
0000140: 0fa1 c66f 3ffd 955e 6f3b c780 3265 afa6  ...o?..^o;..2e..
0000150: 76ac 6b63 fa32 6784 510b 5c5d cd0d 5413  v.kc.2g.Q.\]..T.
0000160: babd 6b15 c5fd 7cab b17d 7c12 a97d 7d5a  ..k...|..}|..}}Z
0000170: d313 a994 f376 99d2 49b4 e6df 154a 5d84  .....v..I....J].
0000180: 38a0 0a47 d12e 07c9 9065 778b 1b7d 7f34  8..G.....ew..}.4
0000190: 54bc dbfd 2cb4 96c2 0ebb 3db1 8afb 8442  T...,.....=....B
00001a0: 74bd 7b59 25fd 7951 86fd 7ff1 717d 78be  t.{Y%.yQ....q}x.
00001b0: 5357 37b3 6524 7861 6ab2 ec05 8f4c 966e  SW7.e$xaj....L.n
00001c0: ec5d 8b9f 2d7d 6fb7 f36b fba1 eb6d 7b34  .]..-}o..k...m{4
00001d0: bdc5 8179 08c5 5b61 89fd 3b15 2b7d 59ab  ...y..[a..;.+}Y.
00001e0: f07d 7fcc 36fd 7c85 3cbd 7eac 45fd 85c4  .}..6.|.<.~.E...
00001f0: 752d aeef df79 9808 a886 8285 a5dd ff34  u-...y.........4
0000200: 5c8d 9e8f b2ba 8079 167d 657a c33d 43bc  \......y.}ez.=C.
0000210: 1db9 76d0 e3e9 70df 986d 7c1e 657d 8363  ..v...p..m|.e}.c
0000220: 613d 7750 3e3d 7944 fa7d 77a5 373d 7765  a=wP>=yD.}w.7=we
0000230: c560 ac62 e5b2 47dd 01fe aebe e8ac e99a  .`.b..G.........
0000240: 887d 930f 5f7c 0fc3 f789 7790 de7d 7f71  .}.._|....w..}.q
0000250: b4bd 7ba9 4d3d 6c8a 1579 75b8 c439 84d2  ..{.M=l..yu..9..
0000260: 513d 7b27 a3bd 7f43 357d 7fa9 e9bd 7704  Q={'...C5}....w.
0000270: ff1d 6a35 02bd 3859 2703 d027 4915 5452  ..j5..8Y'..'I.TR
0000280: dd05 9eb7 577a 8263 01a2 a46f d8bd 5daa  ....Wz.c...o..].
0000290: eebd 72a2 21db 732a 98b3 f657 d033 fb18  ..r.!.s*...W.3..
00002a0: 987d 82f5 f2bd 7c08 2dfd 85c8 38fd 82ca  .}....|.-...8...
00002b0: 5939 ee8e 140f 5b3d 0cc9 9c81 9c92 5965  Y9....[=......Ye
00002c0: 3b9d 96af 8b47 7d9f e2ff 8392 c6ac ff71  ;....G}........q
00002d0: b5f3 81bd d482 750b 5749 f1aa 4cfc e77a  ......u.WI..L..z
00002e0: b1fd 7ead e23d 7900 aabd 7f55 3cbd 83f5  ..~..=y....U<...
00002f0: 97bb e4dd 6941 50cd 567f 37d0 3e5c 9e26  ....iAP.V.7.>\.&
```

Fig. 12. 31-round SHA-0: second preimage (part 1)

```
0000300: 7a23 d3cf cdbc 6851 fc6b 6fdc 0a73 e75c  z#....hQ.ko..s.\
0000310: 5c53 e94b c211 c83c 9d3b 59c7 77fd 7a5a  \S.K...<.;Y.w.zZ
0000320: 9afd 7b0b 883d 835f c8fd 7f30 98bd 7f34  ..{..=._...0...4
0000330: 570a e920 9bc7 4e38 9d9f 7faa 7e51 9dbd  W.. ..N8....~Q..
0000340: 0f0c c697 20e5 9f98 9c99 fff8 442d 7383  .... .......D-s.
0000350: 583a 2e86 7bc5 a5a9 48e1 57da 0675 61ce  X:..{...H.W..ua.
0000360: 1a3d 78d0 23bd 7ac5 24fd 804e 473d 7aa0  .=x.#.z.$..NG=z.
0000370: b7c3 6cdc 9ce1 2251 87d2 dbef 4739 a47c  ..l..."Q....G9.|
0000380: 9d15 92a7 4a9c bcc5 74a9 579c 41dd 7e99  ....J...t.W.A.~.
0000390: a8db 7a99 398f 4864 1fa4 54bd 9d6c 7c8e  ..z.9.Hd..T..l|.
00003a0: 57bd 7ac7 12fd 84b9 703d 7a02 9cbd 7c37  W.z.....p=z...|7
00003b0: f88f b361 8ec1 1971 f419 9d71 beb2 f4ca  ...a...q...q....
00003c0: 1c42 eccf 31e1 3783 3e6d bf75 3765 83a6  .B..1.7.>m.u7e..
00003d0: 41cc 5f17 c588 0436 df79 4dd9 fafd 752f  A._....6.yM...u/
00003e0: 353d 7fcc fffd 79e5 057d 7cc1 c93d 84b5  5=....y..}|..=..
00003f0: 9080 9f98 75f5 c427 c6d3 ffbb 2d55 00d0  ....u..'....-U..
0000400: 3c01 d6c7 410b 7bcd 8d7c f79e c27d 7b5c  <...A.{..|...}{\
0000410: f6dc 7047 4bd6 6e66 2ab7 84a2 2e7d 8676  ..pGK.nf*....}.v
0000420: b1fd 795b dbbd 7e58 043d 82bf 9b3d 836b  ..y[..~X.=...=.k
0000430: fbc6 0485 29f2 5213 6b02 b802 3b6a 30df  ....).R.k...;j0.
0000440: fa7d 8887 177d 4027 298e 7ba9 145b 7aed  .}...}@').{..[z.
0000450: 303d 8219 9cbd 7c5f 1cf9 36b5 b439 3dee  0=....|_..6..9=.
0000460: b63d 76d4 9bfd 7b6c bdbd 83b8 7e3d 8463  .=v...{l....~=.c
0000470: 93b0 32ab c928 2966 29aa ae16 6ec5 9ad0  ..2..()f)...n...
0000480: 067e 86bf 306d 7b87 f77d ffb8 446d 7bcf  .~..0m{..}..Dm{.
0000490: 143d 35b6 e879 39cf d7b9 5c05 79bd 571f  .=5..y9...\.y.W.
00004a0: 2cfd 8640 4f7d 80d7 bf3d 85b5 7d7d 7e35  ,..@O}...=..}}~5
00004b0: ef2e 8255 95e1 8361 6086 946e e1ce 3da9  ...U...a`..n..=.
00004c0: e88c eab7 23f1 0da3 261b 7baf ce35 6bae  ....#...&.{..5k.
00004d0: 2f39 e040 12a1 a732 463d 693f d915 7566  /9.@...2F=i?..uf
00004e0: bfbd 7d9d 853d 7bee f6bd 7d1e 1e3d 7afe  ..}..={...}..=z.
00004f0: 8ecb 8c22 62eb 7e25 7d3d fbc1 0f75 350d  ..."b.~%}=...u5.
0000500: d281 c797 9775 6000 77df 9f95 3737 7fbb  .....u`.w...77..
0000510: 485c 79e1 0b9c 7585 0344 efea 56e4 f0e6  H\y...u..D..V...
0000520: 4b7d 78a6 2efd 7fc3 f03d 80c3 3f3d 827a  K}x......=..?=.z
0000530: 30c8 3047 1144 d3a9 104a 7c41 3947 4120  0.0G.D...J|A9GA 
0000540: 49a0 8a9f 5c1d 026b e885 6374 2775 8269  I...\..k..ct'u.i
0000550: cb7d 017c fcb4 c107 50fb 6c2e 37bb 71a6  .}.|....P.l.7.q.
0000560: eb7d 821c d3bd 8633 6ffd 7cbd 81fd 77e7  .}.....3o.|...w.
0000570: b2c4 fef3 1c48 7d72 136a 2995 0afe 99d5  .....H}r.j).....
0000580: 6420 7368 6520 7761 7320 746f 2067 6574  d she was to get
0000590: 206f 7574 2061 6761 696e 2e0a             out again..
```

Fig. 13. 31-round SHA-0: second preimage (part 2)