Cryptanalysis of GRINDAHL

Thomas Peyrin

France Télécom R&D, Issy-les-Moulineaux, France
AIST, Tokyo, Japan
University of Versailles, France
thomas.peyrin@orange-ftgroup.com

Abstract. Due to recent breakthroughs in hash functions cryptanalysis, some new hash schemes have been proposed. GRINDAHL is a novel hash function, designed by Knudsen, Rechberger and Thomsen and published at FSE 2007. It has the particularity that it follows the RIJNDAEL design strategy, with an efficiency comparable to SHA-256. This paper provides the first cryptanalytic work on this new scheme. We show that the 256-bit version of GRINDAHL is not collision resistant. With a work effort of approximatively 2^{112} hash computations, one can generate a collision.

Key words: GRINDAHL, hash functions, RIJNDAEL.

1 Introduction

Hash functions are one of the most utilized primitives in cryptography. Basically, a hash function H is a function that maps an input of variable size to a fixed length output value. A cryptographic hash function has the additional feature that it must satisfy some security properties such as preimage resistance, second preimage resistance and collision resistance. For an ideal hash function with an n-bit output, one expects that compromising these properties should require 2^n, 2^n and $2^{n/2}$ operations respectively [12].

A possible way of building a hash function has been introduced by the pioneering work of Merkle and Damgård [22,10], using an iterative process: at each iteration, a fixed-length input function h (the compression function) updates an internal state called *chaining variable* with some part of the message. With some appropriate padding of the message to be hashed, the problem of building a collision-resistant hash function H is then reduced to the problem of building a collision-resistant compression function h. However, due to recent attacks [16,18,17,14] against this iterative process, other hash domain extensions have been introduced [2,5].

Almost all the proposed hash functions define a compression function to be used with any hash domain extension algorithm. There are basically three different ways of building a compression function. First, one can relate the security of h to a hard

The author is supported by the Japan Society for Promotion of Science and the French RNRT SAPHIR project (http://www.crypto-hash.fr).

problem, such as factorisation [9], finding small vectors in lattices [3], syndrome decoding [1] or solving multivariate quadratic equations [6]. The usually bad efficiency of these schemes is compensated by the proofs of security they provide. Another very active domain is the construction of secure compression functions based on block ciphers. The problem of building a secure n-bit compression function from an ideal n-bit block cipher is more or less resolved [27,28,7] and due to a need of bigger output size the cryptographic community is now concentrated on the problem of building a secure $(k \times n)$-bit compression function from an ideal n-bit block cipher [13,26,30]. Finally, the most common and efficient way of building a compression function is from scratch, for example the well known and standardized SHA-1 [25] or MD5 [29]. However, almost all of this type of hash functions have been broken by novel cryptanalysis results [31,32,33,34,8].

To anticipate further improvements of the attacks, the NIST is initiating an effort [24] to develop one or more additional hash algorithms through a public competition, similar to the development process for the Advanced Encryption Standard [23]. In parallel, new hash functions have been published very recently, such as FORK-256 [15] (broken in [21]), RADIO-GATÙN [4] or GRINDAHL [20]. We show here that for the GRINDAHL hash function one can find a collision (resp. a second preimage) with a work effort of 2^{112} (resp. 2^{224}) hash computations approximatively, whereas $2^{n/2}$ (resp. 2^n) is expected for an ideal hash function. Note that the conceptors of GRINDAHL only claimed a (second) preimage security of $2^{n/2}$ operations, already providing an attack requiring lower than 2^n operations.

The paper is organized as follows. In Section 2 we quickly recall the specification of the GRINDAHL hash function and in Section 3 we begin the analysis with various observations on the scheme and the general methodology that allows us to build a differential path. Then, in Section 4, we provide the first collision attack on GRINDAHL. Finally, we discuss possible patches in Section 5 and we conclude in Section 6.

2 Description of GRINDAHL

GRINDAHL is a family of hash functions based on the so-called *Concatenate-Permute-Truncate* strategy, where in our case the permutation uses the design principles of RIJNDAEL [11], well known for being the winning candidate of the Advanced Encryption Standard (AES) process [23]. Two algorithms are defined, a version with a 256-bit output and a 512-bit one. Also, a compression function mode is given, taking only fixed-length inputs, to be used with any hash domain extension algorithm. We give in this section a quick description of the GRINDAHL hash function with a 256-bit output. For a more detailed specification of the algorithm, we refer to [20].

Let $n = 256$ be the number of output bits of the hash function H, with an *internal state* s of 48 bytes (384 bits), and let M be the message (appropriately padded) to be hashed. M is split into m blocks $M_1, \dots, M_m$ of 4 bytes each (32 bits). At each iteration k, the message block M_k will be used to update the internal state s_{k-1}. We call *extended internal state* $\hat{s}_k$ the concatenation of the message block M_{k+1} and the internal state s_k, i.e. $\hat{s}_k = M_{k+1}||s_k$. We thus have $|\hat{s}_k| = (4+48) \times 8 = 416$ bits. We denote by $trunc_t(x)$ the least significant t bits of x. Let $P : \{0,1\}^{416} \longmapsto \{0,1\}^{416}$ be a non-linear permutation, and let s_0 be the *initial internal state* defined by $s_0 = \{0\}^{384}$. Then, for each iteration k with $0 < k < m$, we have $s_k = trunc_{384}(P(\hat{s}_{k-1}))$. For the last iteration, the truncation is omitted: $\hat{s}_m = P(\hat{s}_{m-1})$. Finally, we apply eight *blank rounds* $\hat{s}_k = P(\hat{s}_{k-1})$, for $m < k \leq m + 8$, and the output of the hash function is $trunc_{256}(\hat{s}_{m+8})$.

The description is not complete since P has not yet been defined. This permutation follows the design principle of RIJNDAEL (the reader is expected to be familiar with the transformation defined in the RIJNDAEL specifications) and thus the extended state $\hat{s}$ is viewed as a matrix of bytes. However, instead of a $(4, 4)$ bytes matrix, we have a matrix α of 4 rows and 13 columns in the case of the 256-bit version of GRINDAHL. The entry of the matrix α located at the i-th row and the j-th column is a byte denoted by $\alpha_{i,j}$. Thus, we have:

$$\alpha = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,12} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,12} \\ \alpha_{2,0} & \alpha_{2,1} & \cdots & \alpha_{2,12} \\ \alpha_{3,0} & \alpha_{3,1} & \cdots & \alpha_{3,12} \end{pmatrix}.$$

By splitting the extended internal state $\hat{s}$ into 52 8-bit chunks $x_0, \ldots, x_{51}$, we can define the conversion from $\hat{s}$ to α by $\alpha_{i,j} = x_{i+4\times j}$. This mapping has a natural inverse. Basically, before each iteration, the first column of α is overwritten with the incoming message block. Finally, the permutation P is defined as

$$P(\alpha) = \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes} \circ \text{AddConstant}(\alpha).$$

MixColumns. This transformation is defined as in the RIJNDAEL specifications.

ShiftRows. This transformation cyclically shifts bytes a number of positions along each row. Thus, the i-th row is rotated by ρ_i positions to the right, with $\rho_0 = 1$, $\rho_1 = 2$, $\rho_2 = 4$ and $\rho_3 = 10$.

SubBytes. The only non-linear part of the permutation, exactly defined as the SubBytes function of RIJNDAEL.

AddConstant. This function is simply defined by $\alpha_{3,12} \longleftarrow \alpha_{3,12} \oplus \texttt{01}$, where `01` is the byte-wise hexadecimal value of 1.

Note that the 512-bit version of GRINDAHL is based on the same principle as the 256-bit version, but the extended internal state is bigger (8 rows instead of 4). The compression function mode for GRINDAHL-256 (without optional input) simply consists in hashing 40 4-byte message blocks for each compression function call.

3 Overall Analysis

In this section, we study possible ways of finding a good differential path for the 256-bit version of GRINDAHL. More precisely, we look for a trail of k iterations starting from s_0 and so that with two different messages M and M' we have the same hash output, i.e. $trunc_{256}(\hat{s}_{m+8}) = trunc_{256}(\hat{s}_{m'+8})$. Thus, we only care about collision and second preimage resistance. Finding a differential path including the blank rounds seems hard since no message block is inserted during this last operation and thus we have very few control on this part. However, the problem looks much easier when trying to find an internal collision: a differential path excluding the blank rounds, i.e. $\hat{s}_m = \hat{s}_{m'}$. Here, we explain how to find such a path, with the constraint that we want this path to have a good probability of success.

3.1 A Known Potential Attack and the Truncated-Differences

In the original paper from FSE 2007, a section explains a potential attack method, pointed out by an anonymous reviewer. This method seems quite natural: the attacker does not look at the actual values of differences inserted in the bytes of the internal state, but only checks if there is a difference or not (this greatly simplifies the analysis). We call this kind of zero or non-zero differences *truncated-differences* in reference to the very similar truncated differences used by Knudsen in [19]. Then, a chain of truncated-differences in which in every round the number of actives bytes (bytes with a non-zero truncated-difference) is low must be found. In this differential path, the truncated-differences can only be erased during two stages of an iteration: during a MixColumns transformation or during the truncation at the end of the iteration. In other words, the number of truncated-differences in a column can be reduced and their position changed by a clever use of the MixColumns transformation (note however that one can never erase all the truncated-differences of a column at a time). Otherwise, a truncated-difference is deleted if it goes to the first column of α at the end of the iteration, due to the truncation. Since at this stage of the attack the differential trail is already settled, one can not force anything for the truncation but one can play with the message blocks inserted at each iteration, in order to force a good behavior in the MixColumns processes (see Section 3.2). In fact, the message bytes act as *active/passive bits* in the sense that new input bytes do not affect some parts of the internal state for a limited number of rounds (see Section 3.3). The feasibility of this method was left as an open problem, and we argue in Section 3.4 that there is a better way of finding a collision on GRINDAHL.

3.2 Analysis of Differences Propagation in MixColumns

The MixColumns transformation used in GRINDAHL is the same as in RIJNDAEL, and its MDS property ensures maximal difference propagation. More precisely, the sum of the number of active bytes of the input and the output is greater or equal to 5. In other words, the number of non-zero truncated-differences of the input and the output of MixColumns is greater or equal to 5.

More formally, let $V = (A, B, C, D)$ be an input vector of four bytes A, B, C and D; and let $W = (A', B', C', D')$ be an output vector of four bytes A', B', C' and D'. We denote the function MixColumns by $MC : V \longmapsto W$ or $MC : (A, B, C, D) \longmapsto (A', B', C', D')$. We also denote by $D_i(V_1, V_2)$ the function returning 1 if the i-th byte of the 4-byte vectors V_1 and V_2 are different, and 0 otherwise. Finally, $ND(V_1, V_2)$ returns the number of such differences, i.e. $ND(V_1, V_2) = \#\{i \mid D_i(V_1, V_2) = 1\}$. We thus have that if $W_1 = MC(V_1)$ and $W_2 = MC(V_2)$ with $V_1 \neq V_2$, then

$$ND(V_1, V_2) + ND(W_1, W_2) \geq 5.$$

Another interesting property is that any input byte of MixColumns defines a permutation for any output byte. Thus, with $W_1 = MC(V_1)$, $W_2 = MC(V_2)$ and $V_1 \neq V_2$ drawn uniformly and randomly in $\{0,1\}^{4\times 8}$, we have for any $1 \leq i \leq 4$:

$$P_D = P[D_i(W_1, W_2) = 0] = \frac{256^3 - 1}{256^4 - 1} \simeq 2^{-8}, \quad (1)$$

$$\overline{P_D} = P[D_i(W_1, W_2) = 1] = 1 - P_D \simeq 1 - 2^{-8}. \quad (2)$$

Our goal is to compute the probability that a fixed mask of input truncated-differences maps to a fixed mask of output truncated-differences (later this will be often utilized in order to compute the probability of success of the differential path). For example, we want to be able to know the probability that given two input words V_1 and V_2 distinct on their 2 first bytes give two output words different on their 3 first bytes through MixColumns (note that this is slightly different from the event that any 2-byte difference input maps to any 3-byte difference output). We can compute those probabilities in two ways, formally or empirically by testing exhaustively all the input values: since MixColumns is linear, dealing with differences or values is the same (during the test, instead of looking for differences or non-differences, we checked for zero values or non-zero values). We give in Table 1 an approximation of those probabilities.

D_I \ D_O	0	1	2	3	4
0	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0
2	$-\infty$	$-\infty$	$-\infty$	-8	0
3	$-\infty$	$-\infty$	-16	-8	0
4	$-\infty$	-24	-16	-8	0

Table 1. Approximate probability that two 4-byte input words with D_I different bytes on predefined positions maps to two 4-byte output words with D_O different bytes on predefined positions through MixColumns. The values are base 2 logarithms.

3.3 Existence of Control Bytes

Modifying some message bytes will obviously modify quite quickly the internal state, but not necessarily immediately. For each modified byte of the message M_k, we give in Table 2 the columns of s (in its matrix representation α) affected by this modification after 1, 2 and 3 iterations. Note that for more than 3 iterations, any message byte affect all the internal state. This *active/passive bytes* feature will allow us to attack different columns of different iterations independently. More precisely, we will control independently the behaviour of some MixColumns transitions thanks to the active/passive bytes.

	0	1	2	3	4	5	6	7	8	9	10	11	12
A_k		✓											
B_k			✓										
C_k					✓								
D_k											✓		

	0	1	2	3	4	5	6	7	8	9	10	11	12
A_k			✓	✓		✓						✓	
B_k				✓	✓		✓						✓
C_k		✓				✓	✓		✓				
D_k		✓						✓				✓	✓

	0	1	2	3	4	5	6	7	8	9	10	11	12
A_k	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓
B_k	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		
C_k			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
D_k	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓

Table 2. Influences on the columns of the extended internal states for a modification of a byte of the message block $M_k = (A_k, B_k, C_k, D_k)$ incoming at iteration k. We denote by ✓ if the column is affected (or active) and void if not. The first table shows influences on s_{k-1}, the second on s_k and the third on s_{k+1}.

3.4 General Strategy

We now have all the necessary tools in order to build a truncated-differential path and evaluate its probability of success. But how to actually find one ? The natural intuition one would have (as the anonymous reviewer suggested) is to always maintain a low number of truncated-differences along the path (to increase the probability). However, finding one such path seems really difficult as one can convince oneself with Property 1 from the original paper:

Property 1. An internal collision for GRINDAHL-256 requires at least 5 iterations. Moreover, any characteristic starting or ending in the extended state with no difference contains at least on round where at least half the extended state bytes (excluding the first column) are active.

This property can be verified with a meet-in-the-middle exhaustive search, as explained in the original paper. However, with a small speed improvement of this algorithm, one can check that an internal collision for GRINDAHL-256 requires at least 6 iterations. Another observation is that by introducing differences in the state, after a few iterations we quickly come to an "all-difference" pair of extended states. Moreover, this "all-difference" pair of extended states is almost stable: the probability that an all-difference pair of columns remains an all-difference pair of columns through MixColumns is approximatively $P_A = (1 - 2^{-8})^4$, so for the twelve columns of the extended state (excepted the first column) we have a probability of $P_A^{12} \simeq 2^{-0,27}$. Thus, our first idea is to not search for a path starting from a zero difference but from an all-difference pair of extended states (which is very easy to get). The overwhelming probability P_A^{12} allows us to start with as much valid starting states as we want.

3.5 Finding a Truncated Differential Path

Searching for a differential path starting from an all-difference pair of extended internal states is quite easy. One method is to go backward almost exhaustively. Indeed, in GRINDAHL the truncated differences propagate in the forward direction as quickly as in the backward direction. More precisely, if we look for a collision at the end of iteration k, we try all the possible truncated difference masks for the message blocks inserted at iterations k, $k-1$, etc. and all the possible transitions of truncated differences through MixColumns, until we come to an all-difference pair of extended states. This algorithm can be greatly improved with an early-abort strategy: we compute a lower bound on the cost of the current trail we are building (taking in account the control provided by the active/passive bytes, see Section 4) and we stop the search branch if the complexity of the attack is already greater or equal to 2^{128} operations. We also stop the search if we go too far in terms of number of iterations[1].

Obviously, by always adding truncated differences to all the message blocks inserted is the fastest way to reach this goal. However, we will use the message bytes inserted as *control bytes* to attack some parts of the differential path independently and thus increase the probability of success. Thus, it may be better not to go too fast on adding truncated differences in order to increase the total number of iterations during the differential path. This will increase the total number of message blocks inserted and therefore provide more control bytes. For example, we can find a path starting from an all-difference pair of extended internal states and requiring only 4 iterations to get a collision, with a probability of success of approximatively 2^{-312}. However, another path requiring 8 iterations to get a collision, with a probability of success of approximatively 2^{-440} may be better. Indeed, in the latter case, even if the probability of success has been divided by a factor 2^{138}, we have inserted 8 message word pairs instead of only 4 in the former case. Thus, we get roughly $2 \times 4 \times 4 \times 8 = 256$ degrees of freedom compared to the former case (4 pairs of message of 4 bytes each). Thus, we obtained more degrees of freedom than what we paid for the probability drop. Obviously, a limit exists: at some point, adding more iterations does not improve things anymore.

[1] in some particular cases, the overall complexity of the attack can remain stable even if the number of iterations of the differential path increases.

4 Finding a Collision

In this Section, from the previous observations, we give a complete collision attack for the 256-bit version of GRINDAHL.

4.1 The Differential Path

Before describing the collision attack, we give in Figure 1 the differential path used and which has been generated thanks to a program implementing the previously explained technique (see Section 3.5). This trail is the best found (among other possible candidates leading to the same complexity). Several candidates were possible and we kept the one providing the best collision attack. We denote by k the number of the last iteration of our differential path, i.e. the last line of Figure 1. First, one can check that all the MixColumns transitions are valid. This differential path has a probability of success of approximatively $2^{-55\times 8} = 2^{-440}$, but we will see that we also have a lot of message blocks inserted allowing to attack some parts independently.

Our aim is to find a pair of messages following the expected differential trail. For this, we don't take care of each iteration one by one, but we deal with each of the 4-byte message words inserted one by one. Said in other words, we will fix the four bytes of a message word pair and check that the newly imposed MixColumns differential transitions are the ones expected in the truncated-differential path. If so, we continue to the next message word pair until we get a collision.

In Table 3, we give all the dependencies of the MixColumns transitions with the message blocks inserted, used as control bytes during the differential path from Figure 1. The cost of all the transitions are given (see Section 3.2) also with the number of control bytes inserted at each iteration (see Section 3.3). The second column of the Table gives the position of the columns of the state in which we force a differential transition during a MixColumns transformation, and the first column indicates in which iteration this event occurs. For each transition, we give in the third column its cost in terms of number of bytes (i.e. for a cost c, the transition has a probability of $2^{-c\times 8}$). Then, each of the seven other columns of the table represents a pair of message words that will be used as control bytes (the letters a or A, b or B, c or C and d or D represent respectively the first, second, third and fourth byte of the 4-byte message inserted). Capital letters means that we have 2 control bytes (we insert a difference for this block) and small letters means that we only have 1 control byte (no difference inserted for this message block). In the core of the table a dash or a cross represents the fact that the MixColumns transition indicated by the corresponding line is affected by the control byte indicated by the corresponding column. We divided those dependencies for the sake of simplicity, the crosses are the dependencies that will be used for the attack: they represent for each MixColumns transition the dependencies of the last involved message word. Finally, the last line gives the cost of each message word insertion in terms of number of bytes (the sum gives the total complexity of the attack).

Note that a lot of the inserted message bytes provide two one-byte degrees of freedom (capital letters) in the case where we introduce a difference for this message block (we can make independently both messages of the pair vary). From Table 3, one can check that we need to test $2^{14\times 8} = 2^{112}$ all-difference pairs of internal state in order to

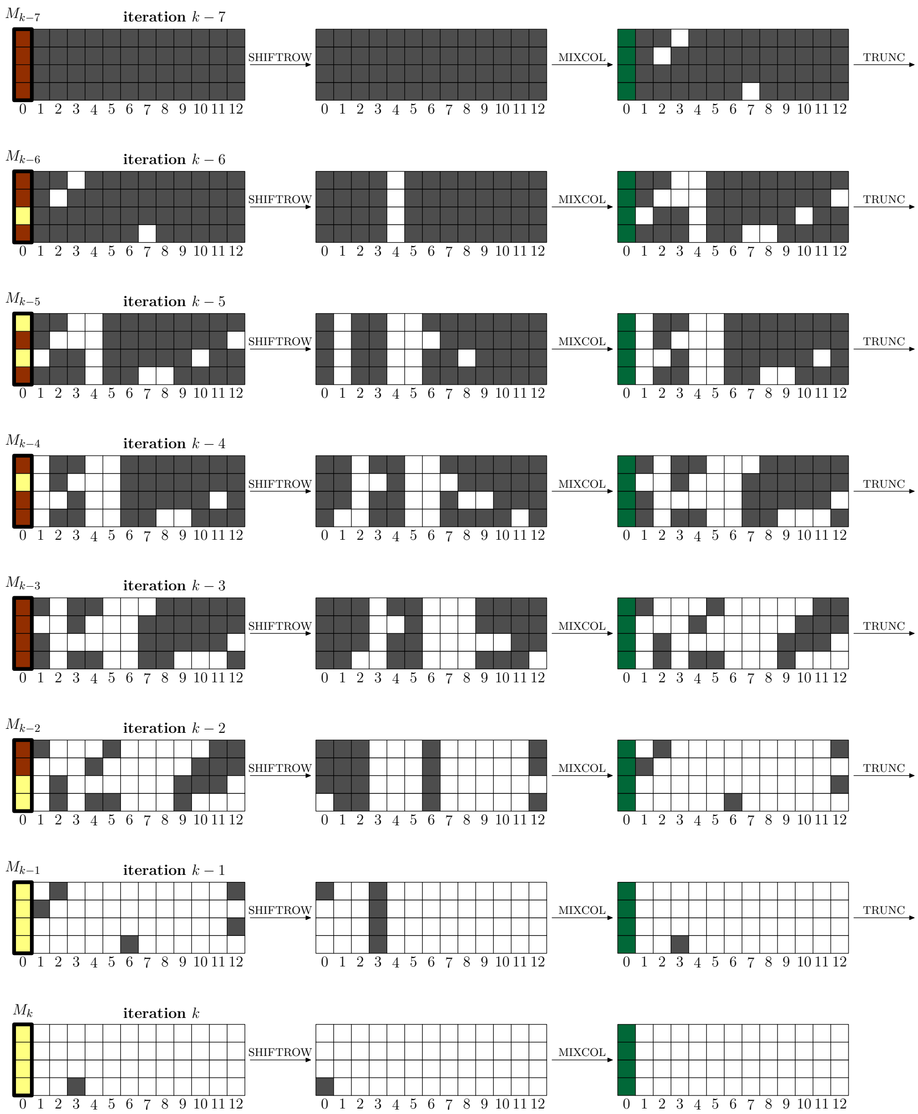

Fig. 1. Truncated-differential path in 8 iterations starting from an all-difference pair of states. The dark cells mean that we have a non-zero difference for this byte, and the light cells stand for no difference. Each row represents an iteration. The first column gives the differences in the state just after its update with the 4-byte message word, and the second column gives the same state after application of the ShiftRows transformation. Finally, the third column represents the internal state just after application of the MixColumns function. Note that the AddConstant and SubBytes functions have no effect on the differential path, thus they are omitted here. Each first 4-byte column of the first column states represents the message words inserted at each iteration, that will later be used as control bytes. The first 4-byte column of the state after every MixColumns transition can have whatever difference mask since those bytes will be immediately truncated.

have a good probability of obtaining a collision. More precisely, the collision attack is as follows.

4.2 The Collision Attack

First step: start with the predefined initial value and compute some iterations with lots of truncated-differences in the incoming message blocks in order to quickly come to an all-difference pair of states denoted A after a few iterations. This step is omitted in the complexity analysis since very largely negligible.

Second step: from this pair of states A, generate $2^{14*8} = 2^{112}$ all-difference pairs of states $A_1, \ldots, A_{2^{112}}$. This step requires $2^{112} \times 2^{0,27} = 2^{112,27}$ iterations computations.

Third step: we continue the attack by fixing the control bytes iteration per iteration: for the message blocks inserted at the beginning of iterations $k-8$, $k-7$, $k-6$ of our truncated-differential path from Table 3, we have more control bytes incoming than necessary. Indeed, we have for the messages inserted at iterations $k-8$, $k-7$ and $k-6$, 8, 8 and 7 control bytes available respectively, whereas we only require 2, 7 and 7 bytes of degrees of freedom respectively. More precisely, for each pair of message words $(M_{k-i}, {M'}_{k-i})$ inserted, its bytes are used in order to adjust the behavior of the MixColumns transitions where crosses appear at column M_{k-i} in Table 3 [2]. For each step, the total cost is equal to the sum of the costs of all the MixColumns transitions involved, minus the number of control bytes available from M_{k-i}. Thus, at this point of the attack, we maintain 2^{112} pairs of messages and states following the differential trail. For the message words inserted at iteration $k-5$, we have 6 control bytes for 7 bytes of conditions, thus we only keep 1 out of 2^8 message pairs and we go to the $(k-4)$-th message word with 2^{104} valid pairs. We continue in the same way for the three lasting message words $k-4$, $k-3$ and $k-2$, having 7, 8 and 4 control bytes respectively[3] and requiring 9, 14 and 9 bytes of conditions respectively. We thus expect to have one pair of messages following the differential trail with a good probability by starting with $2^{14\times8} = 2^{112}$ all-difference pairs of states.

Fourth step: add a $(k+1)$-th message block without truncated-difference in order to force a truncation after the last iteration k of the differential trail (the final blank rounds are done without truncation).

4.3 Discussion on the Attack

For the sake of clarity, we explain more precisely how to deal with the control bytes by giving an example. Let set ourselves when the attacker has to fix the message pair

[2] since in Table 3 the crosses represent the last message word involved for the transition, the previous dependencies (represented by a dash) are already fixed at this point.

[3] for the $k-2$ case, we only have 4 control bytes and not 6 as indicated in Table 3. Indeed, since c and d are not involved in any MixColumns transition, they can not be considered as control bytes.

it	col	cost	message blocks inserted																											
			$k-8$				$k-7$				$k-6$				$k-5$				$k-4$				$k-3$				$k-2$			
			A	B	C	D	A	B	C	D	A	B	c	D	a	B	c	D	A	b	C	D	A	B	C	D	A	B	c	d
k-7	2	1	–					×																						
	3	1	×	×																										
	7	1				×																								
k-6	1	1		–		–			–	–	×																			
	2	1	–		–	–	–					×																		
	3	2	–	–	–	–	×	×																						
	7	1	–	–	–					×																				
	8	1	–	–	–	–			×																					
	10	1		–	–									×																
	12	1	–		–	–		×		×																				
k-5	2	1	–	–	–	–	–		–	–	–					×														
	3	1	–	–	–	–	–	–	–	–	×	×																		
	8	1	–	–	–	–	–	–	–	–			×																	
	9	1	–	–	–	–	×	×	×	×																				
	11	1	–	–	–	–			–	–	×		×	×																
k-4	1	1	–	–	–	–	–	–	–	–		–		–			–	–	×											
	3	1	–	–	–	–	–	–	–	–	–	–	–	–	×	×														
	4	2	–	–	–	–	–	–	–	–	–	–		–		–					×									
	7	1	–	–	–	–	–	–	–	–	–	–	–					×												
	9	1	–	–	–	–	–	–	–	–	×	×	×	×																
	10	1	–	–	–	–	–	–	–	–		–	–									×								
	11	1	–	–	–	–	–	–	–	–			–	–	×		×	×												
	12	1	–	–	–	–	–	–	–	–	–		–	–		×		×												
k-3	1	3	–	–	–	–	–	–	–	–	–	–	–	–		–		–			–	–	×							
	2	2	–	–	–	–	–	–	–	–	–	–	–	–	–		–	–	–					×						
	4	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–		–		–					×					
	5	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	×		×									
	9	2	–	–	–	–	–	–	–	–	–	–	–	–	×	×	×	×												
	10	2	–	–	–	–	–	–	–	–	–	–	–	–		–	–									×				
	11	1	–	–	–	–	–	–	–	–	–	–	–	–			–	–	×		×	×								
	12	2	–	–	–	–	–	–	–	–	–	–	–	–	–		–	–		×		×								
k-2	1	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–		–		–			–	–	×			
	2	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–		–	–	–					×		
	6	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–			×	×					
	12	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–		–	–		×		×				
k-1	3	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	×	×		
COST			**0**				**0**				**0**				**1**				**2**				**6**				**5**			

Table 3. Dependencies of the message blocks used as control bytes and inserted during the truncated-differential path from Figure 1, for a collision at the end of iteration k.

incoming at step $k-5$ (seventh column in Table 3). The previous message words have already been fixed during the attack, thus we only have to deal with the crosses in Table 3. Some MixColumns differential transitions have to behave as required by the truncated-differential path, and this has a cost. For example, at the second column of the $(k-5)$-th iteration, we need a 4-truncated-differences to 3-truncated-differences transition and this will happen with probability 2^{-8}, thus with a cost of 1 byte. However, to make this event occur, we can use the message word inserted at iteration $k-5$ (more precisely its second byte) in order to randomize the instantiation of the transition. Note that there are several ways of doing this step, and this is discussed below. We actually have a good probability to find 2^8 valid pairs of message bytes for this transition: two control bytes for one byte of condition. We do the same process for the seventh column transition of iteration $k-4$ with the fourth byte of the message word: again two control bytes for one byte of condition. Then we identify the subset of the cross product of the two sets of 2^8 byte pairs such that the twelfth column transitions of iteration $k-4$ is verified (depending only on the two previously fixed pairs of message bytes), which costs one byte of condition. So, we maintain 2^8 valid possibilities. Then, we fix the first byte of the message word to deal with the third column transition of iteration $k-4$: since this costs one control byte for one byte of condition, we still maintain 2^8 valid possibilities. Finally, with the lasting byte of the message word (the third), we look for a good transition for the ninth column of iteration $k-3$: this costs one control byte for two bytes of conditions but we had maintained 2^8 valid possibilities before. Thus, in the end, we have a good probability to find a valid message word for all the transitions cited. However, we didn't take care of the eleventh column of iteration $k-4$, which costs us one byte of condition. To summarize, this whole step will cost us 2^8 tries because we had a total of six control bytes for a total of seven bytes of conditions. Repeating this reasoning for all the message words inserted at each iteration of the differential path explains the 2^{112} tries cost for the whole collision attack.

One may argue that we indeed need to try 2^{112} all-difference pairs of states but the basic operation is costly when playing with the control bytes. Indeed, with the previous example, some steps require to pass through 2^8 or 2^{16} values of message words, each requiring only a SubBytes computation on a whole column, or one or two iteration processes (depending on which column of the state the transition occur). Even if it is still an attack, the complexity would be a slightly higher. This argument is true if the attacker uses a naive search method. However, unexpensive precomputations allow to reduce the computational cost of the search table lookups. For example, with as few as 2^{32} precomputation time and memory, one can generate all the informations needed to quickly execute the search needed during the third step of the collision search. Only a few table lookups would then be required. One might also wonder why we did not count the complexity of the few 4-truncated-differences to 4-truncated-differences transitions. Such transitions always have a great probability to happen $P_A = (1-2^{-8})^4 \simeq 2^{-0,02}$. Therefore they have very little effect on the complexity of the attack. This operation is clearly less costly than doing a whole iteration process. Moreover, the compression function mode performs 40 iterations for one compression call. Thus our attack actually runs in less than 2^{112} hash computations, all the complexity coming from the generation of 2^{112} all-difference pairs of states.

Note that we checked that this kind of attack also works with a complexity of at most 2^{120} hash computations for all the rotation constants providing the best diffusion, which seems to indicate that the internal state of GRINDAHL is not big enough.

We provide in Appendix the extension of this technique for the second preimage case applied to the 256-bit version of GRINDAHL. However, note that the GRINDAHL conceptors only claimed a 128-bit security for (second) preimage resistance, showing that (second) preimages can be found in less than 2^{256} operations.

5 Discussion on the Attack and Possible Patches

Most of the difficulty of the presented attack is to actually find a good differential path, and this is possible by letting the differences totally spread and start from an all-difference pair of states. Moreover, even if better differential trails may be found by maintaining a low weight of differences (which is hard to find), we think that the complexity will not drastically decrease compared to our attack. Indeed, the complexity cost grows quickly due to the last iterations of the differential trail (where very few control bytes are available), and these steps will remain very costly whatever the differential trail used. Said in other words, we can compute a lower bound on the complexity of an attack using any truncated-differential path and control bytes. For example, a short program gives us that a similar truncated-differential attack for the 256-bit version of GRINDAHL requires at least 2^{104} operations (whatever the truncated-differential path). Note that this does not mean that such an attack exists.

Thus it would be very interesting to think of a new version of GRINDAHL (with a comparable efficiency) that resists the presented attack but also any attack dealing with truncated-differences and control bytes. Thus, one wants the lower bound on the complexity of an attack using truncated-differential path and control bytes to be greater or equal to 2^{128} operations, and even greater for a good security margin. If this is possible, an attacker that wants to find a collision would have to first find a differential trail and then to deal with the actual values of differences in order to lower the complexity. The SubBytes transformation would therefore discourage this kind of attack and we would obtain a hash function with a strong security argument. A new GRINDAHL version with such a property and a reasonable efficiency could be designed by adding some more columns in the states. The question of the number of the columns to be added or other possible patches is left open for future researches.

6 Conclusion

We showed in this work that the 256-bit version of GRINDAHL is not collision resistant. By introducing a non-intuitive technique in order to find a good differential path and with a careful use of the control bytes available, we presented an attack finding collisions with no more than 2^{112} hash computations. We believe that such a reasoning would apply for the 512-bit version of GRINDAHL, even if the search space for a differential path in this case would be much bigger. Finally, we provided possible patches for the 256-bit version of GRINDAHL that may lead to new versions with stronger security arguments.

Acknowledgements

The author would like to thank the conceptors of GRINDAHL (Lars Knudsen, Christian Rechberger, Søren Thomsen) and Henri Gilbert, Olivier Billet and Yannick Seurin for their valuable remarks on the attack and discussions on the GRINDAHL design.

References

1. D. Augot, M. Finiasz and N. Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In E. Dawson and S. Vaudenay, editors, *Progress in Cryptology – Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer-Verlag, 2005.
2. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 2006.
3. K. Bentahar, D. Page, M-J.O. Saarinen, J.H. Silverman and N.P. Smart. LASH. In Proceedings of *Second NIST Cryptographic Hash Workshop*, 2006 . Available from: `www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.
4. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche. RadioGatun, a Belt-and-Mill Hash Function. In Proceedings of *Second NIST Cryptographic Hash Workshop*, 2006 . Available from: `www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.
5. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions: HAIFA. In Proceedings of *Second NIST Cryptographic Hash Workshop*, 2006 . Available from: `www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.
6. O. Billet, M.J.B. Robshaw and T. Peyrin. On Building Hash Functions From Multivariate Quadratic Equations. In J. Pieprzyk, editor, *Information Security and Privacy – ACISP 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
7. J. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 2002.
8. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006.
9. S. Contini, A.K. Lenstra and R. Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer-Verlag, 2006.
10. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.
11. J. Daemen and V. Rijmen The Design of Rijndael. Springer-Verlag, 2002.
12. A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
13. S. Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In M.J.B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer-Verlag, 2006.

14. J.J. Hoch and A. Shamir. Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In M.J.B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2006.
15. D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon and S. Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In M.J.B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2006.
16. A. Joux. Multi-collisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer-Verlag, 2004.
17. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer-Verlag, 2006.
18. J. Kelsey and B. Schneier. Second Preimages on n-bit Hash Functions for Much Less Than 2^n Work. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer-Verlag, 2005.
19. L.R. Knudsen. Truncated and Higher Order Differentials. In B. Preneel, editor, *Fast Software Encryption – FSE 1994*, in volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1995.
20. L.R. Knudsen, C. Rechberger and S.S. Thomsen. Grindahl - A family of hash functions. In A. Biryukov, editor, *Fast Software Encryption – FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
21. K. Matusiewicz, T. Peyrin, O. Billet, S. Contini and J. Pieprzyk. Cryptanalysis of FORK-256. In A. Biryukov, editor, *Fast Software Encryption – FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
22. R.C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.
23. National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard, November 2001 . Available from: `www.csrc.nist.gov`.
24. National Institute of Standards and Technology. Advanced Hash Standard . Available from: `www.csrc.nist.gov/pki/HashWorkshop/index.html`.
25. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002 . Available from: `www.csrc.nist.gov`.
26. T. Peyrin, H. Gilbert, F. Muller and M.J.B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 315–331. Springer-Verlag, 2006.
27. B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
28. B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In D.R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer-Verlag, 1993.
29. Ronald L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992 . Available from: `www.ietf.org/rfc/rfc1321.txt`.
30. Y. Seurin and T. Peyrin. Security Analysis of Constructions Combining FIL Random Oracles. In A. Biryukov, editor, *Fast Software Encryption – FSE 2007*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

31. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2005.
32. X. Wang, Y.L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005.
33. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.
34. X. Wang, H. Yu and Y.L. Yin. Efficient Collision Search Attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005.

Appendix

Extending the Collision Attack to Second Preimage Resistance. Our previously explained collision attack has a nice feature for an attacker: one does not care about the actual values of the differences. Thus, we have very few constraints during the differential path. This remark allows us to extend our collision attack to second preimage resistance if the second preimage challenge has a reasonable number of message blocks. For example, let us look at the differential path from Figure 2. If one wants to find a second preimage using this path, only the number of control bytes will change as compared with the collision attack case: when we previously had two control bytes because of the insertion of a non-zero truncated-difference (capital letters in Table 3), we only get one control byte since the first message block is fixed by the challenge. For the same reason, when a zero truncated-difference is inserted, we have one control byte for the collision case (small letters in Table 3) and we have no more control byte in the second preimage case.

Using exactly the same techniques as for the collision attack, one can find a second preimage in approximatively $2^{28\times 8} = 2^{224}$ hash computations whereas 2^{256} hash computations should be required for an ideal 256-bit hash function. The drawback of this method is that we require the challenge to contain enough message blocks in order to have enough iterations to follow our differential path (around 8 iterations: 3 to reach an all-difference pair of states, 4 to follow the path from Figure 2 and 1 to force the truncation at the end of our differential trail). Moreover, we need approximatively 7 more iterations if we also take in account that we need to generate 2^{224} all-difference pairs of internal state to pass the differential trail. Thus, our attack works for a challenge of at least 15 message words.

Note that the GRINDAHL designers only claimed a 2^{128} security for their 256-bit version, and provided in their original paper a (second) preimage algorithm requiring 2^{176} operations and memory with a meet-in-the-middle reasoning on the internal state size.

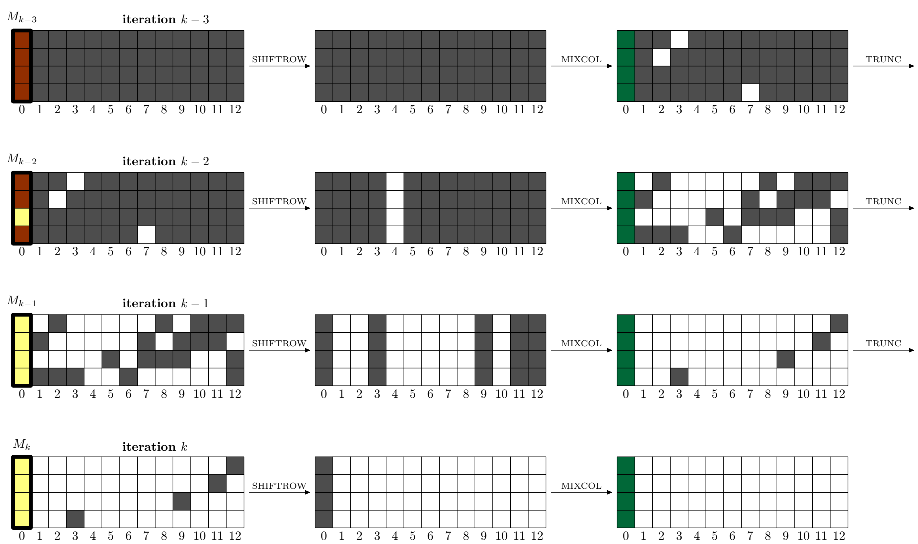

Fig. 2. Truncated-differential path in 4 iterations starting from an all-difference pair of states, to be used for a second preimage attack.

it	col	cost	**message blocks inserted**											
			$k-4$				$k-3$				$k-2$			
			A	B	C	D	A	B	C	D	A	B	c	D
k-3	2	1					×							
	3	1	×	×										
	7	1				×								
k-2	1	2		–		–			–	–	×			
	2	2	–		–	–	–					×		
	3	3	–	–	–	–	×	×						
	5	3	–	–	–	–	×		×					
	6	3	–	–	–			×	×					
	7	2	–	–	–					×				
	8	2	–	–	–	–			×					
	9	2	×	×	×	×								
	10	2		–	–									×
	11	2			–	–	×		×	×				
	12	1	–		–	–		×		×				
k-1	3	3	–	–	–	–	–	–	–	–	×	×		
	9	3	–	–	–	–	×	×	×	×				
	11	3	–	–	–	–			–	–	×		×	×
	12	3	–	–	–	–	–		–	–		×		×
COST			**0**				**16**				**12**			

Table 4. Dependencies of the message blocks used as control bytes and inserted during the truncated-differential path from Figure 2 in a second preimage attack, for an internal collision at the end of iteration k. Note that for the pairs of message words that will be used as control bytes, since we set ourselves in the second preimage attack case, capital letters means that we have one control byte (we insert a difference for this block) and small letters means that we have no control byte (no difference inserted for this message block).