

# Some Attacks Against a Double Length Hash Proposal

Lars R. Knudsen<sup>1</sup> and Frédéric Muller<sup>2</sup>

<sup>1</sup> Department of Mathematics, Technical University of Denmark  
DK-2800 Kgs. Lyngby, Denmark  
Lars.R.Knudsen@mat.dtu.dk

<sup>2</sup> DCSSI Crypto Lab  
51, boulevard de La Tour-Maubourg 75700 PARIS 07 SP  
Frederic.Muller@sgdn.pm.gouv.fr

**Abstract.** At FSE 2005, Nandi *et al* proposed a method to turn an  $n$ -bit compression function into a  $2n$ -bit compression function. In the black-box model, the security of this double length hash proposal against collision attacks is proven, if no more than  $\Omega(2^{2n/3})$  oracle queries to the underlying  $n$ -bit function are made.

We explore the security of this hash proposal regarding several classes of attacks. We describe a collision attack that matches the proven security bound and we show how to find preimages in time  $2^n$ . For optimum security the complexities of finding collisions and preimages for a  $2n$ -bit compression function should be respectively of  $2^n$  and  $2^{2n}$ . We also show that if the output is truncated to  $s \leq 2n$  bits, one can find collisions in time roughly  $2^{s/3}$  and preimages in time roughly  $2^{s/2}$ .

These attacks illustrate some important weaknesses of the FSE 2005 proposal, while none of them actually contradicts the proof of security.

## 1 Introduction

### 1.1 Hash Functions

Cryptographic hash functions are important primitives in cryptology. They are used in a wide range of applications including message integrity, authentication schemes or public key encryption schemes. Most importantly, they are used to speed up digital signature schemes, which otherwise would be slow and unlikely to be implemented widely. A cryptographic hash function takes an input of arbitrary size and produces an output, also called the hash value, of a fixed, predetermined size. In practice there is a limit for the length of the input, but typically this is chosen big enough for all practical applications. The important properties of a cryptographic hash function are :

- *collision-resistance* : it should be difficult to find a pair  $x \neq x'$  of inputs to the hash function  $H$  such that  $H(x) = H(x')$
- *2nd preimage-resistance* : it should be difficult, for a given  $x$  to find  $x' \neq x$  such that  $H(x) = H(x')$

- *preimage-resistance* : it should be difficult, for a given  $y$  to find  $x$  such that  $H(x) = y$

There are generic attacks which apply to any hash function. If the size of the hash value is  $n$  bits, then it is well-known that collisions can be found in time  $2^{n/2}$  and preimages can be found in time  $2^n$ . For 2nd preimages, the complexity of generic attacks ranges between  $2^{n/2}$  and  $2^n$ , depending on the length of the target message. Recent results by Kelsey and Schneier show that the complexity can be only  $2^{n/2}$  if the length of the target message is also  $2^{n/2}$  [10]. In general, hash functions are built by iterating a basic function called the **compression function**. Attacks can target either the full hash function or the compression function only, although there are connections between both approaches.

## 1.2 Recent Results in Attacking Hash Functions

Many advances have been made recently for hash function cryptanalysis :

- Some important weaknesses have been shown for popular algorithms. It is the case of MD4 [7, 17], MD5 [19], SHA-0 [2, 4, 20] and SHA-1 [18], for which it was shown how to find collisions much faster than  $2^{n/2}$ . These results illustrate some weaknesses of the underlying compression functions.
- The generic construction itself could be at risk. Most hash functions are iterative and are built using the Merkle-Damgård method [6, 12]. Recent results suggest that this construction is not necessarily a good choice [9, 10].
- Computing power is always growing. Attacks with complexity  $2^{64}$  are already accessible using distributed computing. And attacks with complexity  $2^{80}$  may also soon be feasible. Therefore hash functions with output size  $\leq 160$  bits are not a good choice for long term security.

In light of all this, more work is probably needed for hash function design. In particular, it is believed that a good solution is to increase the size of the internal state. This idea has been independently proposed by Lucks [11], Hirose [8] and Nandi *et al.*[14]. Unfortunately the output size of most available compression functions is not large enough, so one needs to design compression functions with an increased output length. Rather than building a new primitive from scratch, Nandi *et al.* suggested to use a secure  $n$ -bit compression function, in order to build a larger compression function (of size  $2n$ -bit for example). The small compression function could then be instantiated with one of the available function, or with a block cipher in the Davies-Meyer construction. An interesting argument for this new construction is that its security has been proven, using some assumptions on the underlying "small" compression function.

## 1.3 Our results

In this paper, we focus on the security of the new double length hash proposal of FSE 2005 [14] against all usual attacks. Regarding the proven security, the authors have only focused on collision attacks, so one may hope to find (second)

preimage attacks without contradicting the security proof. Another interesting open problem is to find a collision attack that matches the security proof claimed in [14].

First, we show that a collision can be found for this proposal in time  $2^{2n/3}$ , which fits the proven security bound (but a generic attack on a  $2n$ -bit function would cost  $2^n$ ). Secondly, we show that preimages can be found in  $2^n$ , while the best generic attack on a  $2n$ -bit compression function costs  $2^{2n}$ .

An interesting question is how these results would apply to a full hash function built using the FSE 2005 compression function. Iterated constructions generally require the compression function to be collision-resistant in order to guarantee the security of the full hash function. This is the case of the popular Merkle-Damgård construction [6, 12]. Another example was given at Crypto'05, where Coron *et al.* revisited the Merkle-Damgård construction [5]. In their analysis, the compression function is modeled as a random oracle.

Sometimes the iterative structure even allows to find better attacks against the full hash than against the compression function alone, as demonstrated in [9, 19]. However we did not take into account such scenarios.

#### 1.4 Notions of security for truncated hash

We introduce new notions of security for compression functions and hash functions. These notions are the **near-preimage resistance** and the **near-collision resistance**. The idea is that it should remain difficult to find collisions or preimages on a truncated version of the function. It is often easier to find "near" attacks than attacks against the full hash. This was illustrated in the case of the SHA family where Biham and Chen first described near collisions [1] before "real" collision attacks were later demonstrated [2, 18].

There are important motivations for taking into account near-collision and near-preimage attacks in practice. First, truncating the output diminishes the size of the hash value. This can be critical to reduce data storage or to reduce the communication complexity (case of MAC's for instance). When it is estimated that  $s$  bits are a sufficient level of security, it is customary to truncate the output. In some case, this even helps to prevent some attacks (it makes more difficult to detect internal collisions in MAC algorithms, for instance).

Secondly, another motivation is that new hash functions may need to remain compatible backward with former applications. For instance, an output of size 160 bits may be needed for compatibility with systems that previously implemented SHA-1. Therefore it is likely that new designs may end up being truncated for practical purpose. A nice illustration of hash function truncation is given by the SHA-2 family [15] : intermediate hash sizes (224 bits and 384 bits) are obtained by truncation of the larger hash sizes (256 bits and 512 bits).

It is expected that the best attacks against truncated hash function remain generic attacks. If the output size is reduced from  $n$  to  $s$  bits, then the best collision attack should cost  $2^{s/2}$  steps and the best preimage attack should cost  $2^s$  steps. In their original paper, Biham and Chen [1] considered near preimages where the **truncated positions are freely chosen by the attacker**. With

these additional degrees of freedom, the task of the attacker is easier, because he can first test several messages and choose the truncated positions only afterwards. For example, it is very easy to find a near preimage for SHA with  $s = 80$  when the attacker can choose the truncated positions. However, such scenarios are not very realistic in practice, so **we only focus on near attacks where the truncated positions are predetermined.**

On the one hand, the security of a truncated hash function is unlikely to drop dramatically compared to the full version. Suppose that one can find preimages in time  $T$  for a  $s$ -bit truncated output. Then, for a given  $n$ -bit challenge  $y$ , an attacker can simply truncate  $y$  to  $s$  bits and obtain a preimage  $x$  for the truncated value. Then,  $s$  bits of the initial challenge are already satisfied by  $x$ , and the attacker can simply hope that the remaining  $n - s$  bits also satisfy the challenge. Therefore a preimage attack for the full hash should cost :

$$T' = T \times 2^{n-s}$$

However there is no guarantee. The previous relation is true for most designs, but there may also exist special designs where this is not true.

On the other hand, truncated the hash function may improve the level of security. This situation has been observed for MAC algorithms where truncation sometimes prevents the detection of internal collisions. Therefore, it is interesting to analyze how the complexity of an attack changes when the output is truncated. For instance, the FSE 2005 double length hash proposal [14] has a security regarding collision attacks proven with a bound of  $2^{2n/3}$ . Thus, it is very tempting to truncate its output to

$$2 \times (2n/3) = 4n/3$$

bits only, since it appears to be the highest security one can achieve. Unfortunately, in that case, we show that collision attacks would become much easier than  $2^{2n/3}$ . More generally, when the hash output is truncated to  $s < 2n$  bits, we show how to find collisions in time  $2^{s/3}$  and preimages in time  $2^{s/2}$ .

## 2 Description of the Double Length Compression Function

A compression function is a function  $F : \{0,1\}^m \rightarrow \{0,1\}^n$  where  $m > n$ . Suppose that  $F$  requires  $t$  calls to either

- a block cipher of block size  $l$ .
- a smaller compression function with inputs of  $l$  bits

Then, the **rate**  $r$  of  $F$  is generally defined as the ratio :

$$r = \frac{m - n}{tl}$$

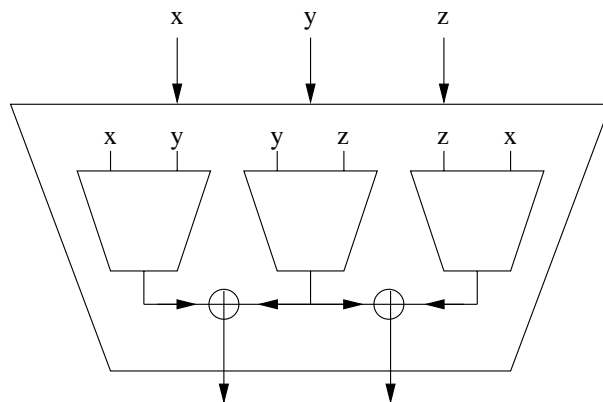
It represents the amount of data compressed for each application of the block cipher (or the smaller function). Achieving a compression function with a ratio

$r = 1$  and which is practical seems to be a very difficult task [3]. In their paper, Nandi *et al.* [14] introduce two new constructions of respective rates  $r = 1/3$  and  $r = 2/3$ . The attacks against both proposals are essentially the same, so we consider first the compression function of rate  $1/3$ .

Let  $f_i : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$  be independent random functions, for  $i = 1, 2, 3$ . We define the double-length compression function  $F : \{0, 1\}^{3n} \rightarrow \{0, 1\}^{2n}$  by :

$$\begin{aligned} F(x, y, z) &= (F_1(x, y, z) \parallel F_2(x, y, z)) \\ &= (f_1(x, y) \oplus f_2(y, z) \parallel f_2(y, z) \oplus f_3(z, x)) \end{aligned}$$

This function has a rate of  $1/3$  : it compresses one block of  $n$  bits with 3 evaluations of the "small"  $f_i$  functions. This construction is also illustrated in Figure 1.



**Fig. 1.** The double length  $1/3$ -rate construction of FSE 2005

Similarly, a function with rate  $2/3$  is proposed in [14]. The idea is to instantiate all the  $f_i$ 's with a block cipher using keys of length  $2n$  bits, in the Davies-Meyer construction. This allows to compress an input of  $4n$  bits into an output of  $2n$  bits, thereby improving the ratio from  $1/3$  to  $2/3$ . This construction could be instantiated with AES-256 for instance.

### 3 Collisions

In [14], it is proven that no collision can be exhibited for the proposed  $2n$ -bit compression function with less than  $\Omega(2^{2n/3})$  queries to the three underlying  $n$ -bit functions. In addition, it is described how to match this bound.

First, we quickly remind the attack proposed by the designers. Then we argue that the number  $Q$  of oracle queries is not the proper way to estimate the complexity of a collision attack. We denote the actual time and memory needed

for the attack by  $T$  and  $M$  : while the original attack is such that  $Q = 2^{2n/3}$ , the authors of [14] do not give many details about its complexity. Apparently their attack requires  $T = M = 2^n$ . Using additional tricks, we propose a better attack which satisfies  $Q = T = M = 2^{2n/3}$ . We do not take into account constant and logarithmic factors to evaluate the complexities of all attacks.

### 3.1 The original attack

Let us pick at random  $2^{n/3}$  values for  $x$ ,  $y$  and  $z$ . We call these values  $x_i$ ,  $y_i$  and  $z_i$  for  $i = 1 \dots 2^{n/3}$ . Compute for all pairs  $(i, j)$ ,

$$A_{i,j} = f_1(x_i, y_j)$$

Then store all results in a table  $T_A(i, j)$  with  $2^{2n/3}$  entries. Similarly, compute

$$B_{i,j} = f_2(y_i, z_j)$$

and store in a table  $T_B(i, j)$ . Finally, compute

$$C_{i,j} = f_3(z_i, x_j)$$

and store in a table  $T_C(i, j)$ . At this point  $Q = 3 \times 2^{2n/3}$  queries have been made to the  $n$ -bit compression functions.

Now consider all triplets  $(x_i, y_j, z_k)$ . There are  $2^n$  such triplets and the compression function  $F$  produces  $2n$ -bit outputs. So the birthday paradox tells us that, with good probability, two triplets will give a collision on  $F$ . One table lookup to  $T_A$ , one to  $T_B$  and one to  $T_C$  are sufficient to evaluate each  $F(x_i, y_j, z_k)$ , so no new oracle query is needed. After computing the  $2^n$  outputs, we store them in a table and sort it, in order to detect if an element appears twice. Therefore a collision is expected to be found with  $Q = 2^{2n/3}$  and  $T = M = 2^n$ .

### 3.2 A better attack

While the notion of oracle queries is useful for a security proof, it is not relevant in practice : specifications of a hash function are typically public, so an attacker can evaluate off-line the functions  $f_i$ . It is therefore not natural to make a distinction between the time needed for the  $Q$  oracle queries and the rest of the analysis. According to the security proof of [14] any generic attack needs to evaluate at least  $2^{2n/3}$  times one of the  $n$ -bit compression functions. Therefore

$$T > 2^{2n/3}$$

for any generic collision attack. In this section, we describe how to reach this lower bound. Fix one of the inputs of  $F$ , for instance let  $y = y_0$ . Then, consider  $2^{2n/3}$  random values of  $x$  and  $z$ . We denote these values by  $x_i$  and  $z_i$  for  $i = 1 \dots 2^{2n/3}$ . Compute, for all  $i$ ,

$$A_i = f_1(x_i, y_0)$$

and store the results in a table  $T_A$ . Similarly, compute for all  $i$

$$B_i = f_2(y_0, z_i)$$

and store the results in a table  $T_B$ . Both tables have  $2^{2n/3}$  entries.

Next, fix an arbitrary  $2n/3$ -bit pattern  $\alpha$  and compute all pairs of elements ( $A_i \in T_A, B_j \in T_B$ ) such that  $A_i \oplus B_j$  starts by  $\alpha$  in its  $2n/3$  least significant bits. There are

$$\frac{2^{2n/3} \times 2^{2n/3}}{2^{2n/3}} = 2^{2n/3}$$

such pairs. They can be obtained with  $2^{2n/3}$  computation. This merging of  $T_A$  and  $T_B$  under the constraint of the pattern  $\alpha$  can be done by first XORing  $\alpha$  to all the elements of  $T_A$ , then sorting  $T_B$ , and finally searching for a collision between the two tables. This costs  $2^{2n/3}$  in time and memory. Such merging algorithms have been known for a long time by the folklore but have been thoroughly studied by Wagner in [16]. The resulting table is noted  $T = T_A \bowtie_{\alpha} T_B$ .

Finally, compute  $F$  for the  $2^{2n/3}$  triplets  $(x_i, y_0, z_j)$  corresponding to elements of  $T$ . It is guaranteed that the  $2n$ -bit output always starts by the prefix  $\alpha$ . Hence the probability of collision among two such triplets is  $2^{-4n/3}$  instead of  $2^{-2n}$ . Since there are  $2^{2n/3}$  triplets to test, the birthday paradox tells us that a collision is expected. To summarize, our improved collision attack requires about

$$T = 2^{2n/3}$$

computations steps, which is an optimal result, according to the security proof of [14]. The memory required is of the order of  $M = 2^{2n/3}$ .

For an ideal compression function with a  $2n$ -bit output, finding a collision should require the computation of  $2^n$  function values. Therefore the FSE hash proposal is not optimal. Also, one might be tempted to truncate the output of the  $F_i$ -functions, e.g., to  $2n/3$  bits each, thereby obtaining a hash result of  $s = 4n/3$  bits. However, as we shall show next, this enables one to find collisions in time less than  $2^{2n/3}$ .

### 3.3 Near-collisions

If the output of  $F$  is truncated to  $s \leq 2n$  we show how to find a near-collision with  $T = 2^{s/3}$ , that is, two inputs to  $F$  which are equal in  $s$  fixed bit-positions.

When  $F_1$  and  $F_2$  are truncated by the same number of bits, the method is exactly similar to the one above, replacing  $2n$  by  $s$ .

Fix the input  $y$  of  $F$  to a value  $y_0$ . Then, consider  $2^{s/3}$  random values of  $x$  and  $z$ . We denote these values by  $x_i$  and  $z_i$  for  $i = 1, \dots, 2^{s/3}$ . Compute, for all  $i$ ,  $A_i = f_1(x_i, y_0)$  and store the results in a table  $T_A$ . Similarly, compute for all  $i$   $B_i = f_2(y_0, z_i)$  and store the results in a table  $T_B$ . Both tables have  $2^{s/3}$  entries. In both tables, we truncate the outputs of  $f_1$  and  $f_2$  as it is done in  $F$ . Then, we fix an arbitrary  $s/3$ -bit pattern  $\beta$  on the  $s/2$  remaining bits, and merge  $T_A$

and  $T_B$  according to this pattern. We use the same algorithm as in Section 3.2. The result is a table  $T = T_A \bowtie_{\alpha} T_B$  containing :

$$\frac{2^{s/3} \times 2^{s/3}}{2^{s/3}} = 2^{s/3}$$

elements of  $s/3$  bits. Finally, we apply  $F$  to all triplets  $(x_i, y, z_j)$  of  $T$ . It is guaranteed that the first  $s/3$  bits of all outputs of  $F_1$  are equal to  $\beta$ . Hence the probability of having a collision among two such triplets in all the  $s$  bits is only  $2^{-2s/3}$  instead of  $2^{-s}$ . Since there are  $2^{s/3}$  triplets to test, the birthday paradox tells us that a collision is expected.

Now suppose  $F_1$  is truncated to  $s_1$  bits and  $F_2$  is truncated to  $s_2$  bits, with  $s = s_1 + s_2$ . The pattern  $\beta$  has length  $s/3$  bits, while the elements in  $T$  have length  $s_1$  bits. So when  $s_1 < s/3$  we may have problems in the previous algorithm. In that case, we need to exchange the roles of  $F_1$  and  $F_2$ , but the idea remains essentially the same.

To summarize, independently of how the truncation is made, we find a near-collision in  $s$  bits with about  $T = 2^{s/3}$  computation. The memory required is also of the order of  $M = 2^{s/3}$ . The number of oracle queries is also of  $Q = 2^{s/3}$

## 4 Preimages

For a  $2n$ -bit compression function, it is expected that  $2^{2n}$  evaluations should be needed in order to find an input  $x$  that maps to  $y = F(x)$  for a given challenge  $y$ . This requirement is generally expressed as preimage resistance. Unfortunately, the hash proposal of [14] does not satisfy this property. In this section, we describe a preimage attack with complexity of  $2^n$  steps.

### 4.1 The preimage attack

Let  $h$  be a given target of length  $2n$  bits. Our goal is to find a preimage  $(x, y, z)$  such that  $F(x, y, z) = h$ . We can rewrite  $h$  as  $(h_1, h_2)$  and re-express our goal as :

$$F_1(x, y, z) = f_1(x, y) \oplus f_2(y, z) = h_1 \tag{1}$$

$$F_2(x, y, z) = f_2(y, z) \oplus f_3(z, x) = h_2 \tag{2}$$

The basic idea is to consider many triplets  $(x, y, z)$ , and to first eliminate those which do not satisfy (1). Actually, merging algorithms can again be used to check this constraint efficiently. If there are enough remaining candidates, one is expected to satisfy (2).

More precisely, let us fix an arbitrary  $y$  and compute, for all possible  $x$ ,  $A_x = f_1(x, y)$ . Results are stored in a table  $T_A$  with  $2^n$  entries. Similarly, we compute all  $B_z = f_2(y, z)$  and store the results in a table  $T_B$ . Using a merging algorithm [16] as in Section 3.2, we compute

$$T = T_A \bowtie_{h_1} T_B$$

$T$  contains all pairs of  $(A_x, B_z)$  such that  $A_x \oplus B_z = h_1$ , so there should be :

$$\frac{2^n \times 2^n}{2^n}$$

entries. The corresponding time complexity is about  $2^n$ . By construction, all triplets  $(x, y, z)$  in table  $T$  satisfy relation (1). Then we compute  $F_2(x, y, z)$  for each of them. We expect that  $h_2$  will be reached, since the probability for a random triplet to satisfy (2) is  $2^{-n}$ . Therefore  $T$  should contain one preimage by  $F$  for the target  $h = (h_1, h_2)$ .

To summarize, we propose a preimage attack against the proposal of [14] with time complexity of  $T = 2^n$  computation steps. In addition, the memory requirement is about  $M = 2^n$ . The number of oracle queries to the function  $f_i$ 's is also about  $2^n$ .

For an ideal compression function of  $2n$  bits, finding a preimage should require about  $2^{2n}$  computation. As was the case for collisions, it is next shown that truncating the output of the hash function will not give ideal security for the truncated construction.

## 4.2 Near-preimages

Let  $h$  be a given target of length  $s < 2n$  bits. We can find a preimage  $(x, y, z)$  such that  $F(x, y, z)$  truncated to  $s$  bits yields  $h$  in time roughly  $2^{s/2}$ . If both functions  $F_i$ 's are truncated to  $s/2$  bits, then the method is in essence the same as in the previous section, simply replace  $n$  by  $s/2$ .

Suppose that both halves of the hash proposal are not truncated equally. For instance,  $F_1$  is truncated to  $s_1$  bits and  $F_2$  to  $s_2$  bits, with

$$s_1 + s_2 = s$$

Without loss of generality, we suppose that  $s_1 > s/2 > s_2$ . In this case, we fix an arbitrary value of  $y$  and consider  $2^{s/2}$  arbitrary values of  $x$  and  $z$ . We compute all  $f_1(x, y)$  and store in table  $T_A$  and similarly compute all  $f_2(y, z)$  into a table  $T_B$ . As in the previous section, we truncate the elements in both tables, and then use a merging algorithm. We verify the constraint on the  $s_1$  bits of  $h_1$ . The result is a table

$$T = T_A \bowtie_{h_1} T_B$$

of size

$$\frac{2^{s/2} \times 2^{s/2}}{2^{s_1}} = 2^{s-s_1} = 2^{s_2}$$

At this point, we are sure to hit the target  $h_1$  for all triplets of  $T$ . Since there are  $2^{s_2}$  such triplets, one of them should also hit the target  $h_2$  and therefore provide a valid preimage.

Therefore, if the double length hash is truncated to  $s$  bits (it does not matter which bits of the output are removed), then a preimage attack costs only  $2^s/2$ .

## 5 Application to the 2/3 rate compression function

[14] also specifies a rate 2/3 compression function and gives an example of an implementation of the scheme using a block cipher as the underlying cryptographic primitive. Here we give only the generic description of the proposal using randomly chosen functions as building blocks.

Let  $f_i : \{0, 1\}^{3n} \rightarrow \{0, 1\}^n$  be independent random functions, for  $i = 1, 2, 3$ . Define the compression function  $F : \{0, 1\}^{4n} \rightarrow \{0, 1\}^{2n}$

$$\begin{aligned} F(x, y, z, w) &= (F_1(x, y, z, w) \mid F_2(x, y, z, w)) \\ &= (f_1(x, y, w) \oplus f_2(y, z, w) \mid f_2(y, z, w) \oplus f_3(x, z, w)) \end{aligned}$$

This function has a rate of 2/3: it compresses two blocks of  $n$  bits with three evaluations of the  $f$ -functions. Note however that this scheme is not directly comparable to the first schemes presented above, since the underlying functions are of a different nature.

Nonetheless, the collision and preimage attacks presented earlier also apply to this variant. This is easy to observe : by fixing the value of  $w$  in the rate 2/3 scheme, one gets exactly the rate 1/3 scheme. It follows easily that all the attacks described in the previous sections also apply to the implementation of the proposal using a block cipher.

## 6 Some general considerations

There is one important property of the compression function of [14] that makes our attacks possible : two of three of the underlying subfunctions  $f_i$  can be attacked independently, by fixing one input variable. Another important observation is that (part of) the output is the sum of the outputs of smaller subfunctions. This opens the door for techniques more efficient than the usual birthday attack. Consider a compression function of the form

$$h(x) = h_1(x_1 \mid y) \oplus h_2(x_2 \mid y),$$

where  $x_1$  can be varied independently of  $x_2$  and vice versa. Then in a search for a collision on  $h$ , one is looking for values  $x_1, x'_1, x_2, x'_2$ , such that

$$h_1(x_1 \mid y) \oplus h_2(x_2 \mid y) \oplus h_1(x'_1 \mid y) \oplus h_2(x'_2 \mid y) = 0,$$

a solution to which is known to be faster than the birthday attack [16].

One possible way to remove this freedom for an attacker could be to use subfunctions whose outputs depend on all (three) input variables. We can do so in a rate 1/3 construction using the subfunctions of the (insecure) rate 2/3 proposal of [14]. Let  $f_i : \{0, 1\}^{3n} \rightarrow \{0, 1\}^n$  be independent random functions, for  $i = 1, 2, 3$ . Define the compression function  $F : \{0, 1\}^{3n} \rightarrow \{0, 1\}^{2n}$

$$F(x, y, z) = (f_1(x, y, z) \oplus f_2(x, y, z) \mid f_2(x, y, z) \oplus f_3(x, y, z))$$

Evidently this reduces to a construction of the form

$$F(x, y, z) = (g_1(x, y, z) \parallel g_2(x, y, z)).$$

The construction of secure double length compression function of this form is further investigated in recent papers by Lucks [11] and Nandi [13].

## 7 Conclusion

In this paper, we have investigated a new double block length hash function proposed at FSE 2005 by Nandi *et al.*. Their idea is to turn a "small", secure,  $n$ -bit compression function into a  $2n$ -bit compression function. The advantage of their method is to offer a proof of security regarding collisions attacks.

Although, we do not contradict this security proof, we show that this construction is not fully satisfying. Indeed, its security level is much worse than a generic  $2n$ -bit compression function. Table 1 summarizes all these results.

**Table 1.** Summary of all attacks against [14]

Type of Attack	Time	Memory	Oracle Query
Collision [14]	$2^n$	$2^n$	$2^{2n/3}$
Collision	$2^{2n/3}$	$2^{2n/3}$	$2^{2n/3}$
Near-collision ( $s$ bits)	$2^{s/3}$	$2^{s/3}$	$2^{s/3}$
Preimage	$2^n$	$2^n$	$2^n$
Near-preimage ( $s$ -bits)	$2^{s/2}$	$2^{s/2}$	$2^{s/2}$

In addition, we have introduced new notions of security for compression functions, *i.e.* **near-collision** and **near-preimage** resistance. These notions are important, because it is quite usual that hash function outputs are truncated for practical purposes. One could be tempted to truncate the output of [14] to  $4n/3$  bits or less, in order to fit to the proven security bound. Our results show that this would be a bad idea because it would deteriorate the security of the construction below  $2^{2n/3}$ .

## References

1. E. Biham and R. Chen. Near-Collisions of SHA-0. In M. Franklin, editor, *Advances in Cryptology – CRYPTO’04*, volume 3152 of *Lectures Notes in Computer Science*, pages 290–305. Springer, 2004.

2. 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, *Advances in Cryptology – Eurocrypt’05*, volume 3494 of *Lectures Notes in Computer Science*, pages 36–57. Springer, 2005.
3. J. Black, M. Cochran, and T. Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In R. Cramer, editor, *Advances in Cryptology – Eurocrypt’05*, volume 3494 of *Lectures Notes in Computer Science*, pages 526–541. Springer, 2005.
4. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO’98*, volume 1462 of *Lectures Notes in Computer Science*, pages 56–71. Springer, 1998.
5. J-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In V. Shoup, editor, *Advances in Cryptology – Crypto’05*, volume 3621 of *Lectures Notes in Computer Science*, pages 430–448. Springer, 2005.
6. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology – Crypto’89*, volume 435 of *Lectures Notes in Computer Science*, pages 416–427. Springer, 1990.
7. H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption – 1996*, volume 1039 of *Lectures Notes in Computer Science*, pages 53–69. Springer, 1996.
8. S. Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In C. Park and S. Chee, editors, *Information Security and Cryptology – ICISC’04*, volume 3506 of *Lectures Notes in Computer Science*. Springer, 2005.
9. A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. Franklin, editor, *Advances in Cryptology – CRYPTO’04*, volume 3152 of *Lectures Notes in Computer Science*, pages 306–316. Springer, 2004.
10. 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’05*, volume 3494 of *Lectures Notes in Computer Science*, pages 474–490. Springer, 2005.
11. S. Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive, Report 2004/253, 2004. <http://eprint.iacr.org/>.
12. R. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – Crypto’89*, volume 435 of *Lectures Notes in Computer Science*, pages 428–446. Springer, 1990.
13. M. Nandi. Designs of Efficient Secure Large Hash Values. Cryptology ePrint Archive, Report 2004/296, 2004. <http://eprint.iacr.org/>.
14. M. Nandi, W. Lee, K. Sakurai, and S. Lee. Security analysis of a 2/3-rate double length compression function in black-box model. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – FSE’05*, volume 3557 of *Lectures Notes in Computer Science*, pages 243–254. Springer, 2005.
15. National Institute of Standards and Technology (NIST). Secure Hash Standard FIPS Publication 180-2, August 2002. Available at <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.
16. D. Wagner. A Generalized Birthday Problem. In M. Yung, editor, *Advances in Cryptology – Crypto’02*, volume 2442 of *Lectures Notes in Computer Science*, pages 288–303. Springer, 2002. Extended Abstract.
17. 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’05*, volume 3494 of *Lectures Notes in Computer Science*, pages 1–18. Springer, 2005.

18. X. Wang, Y. Yin, and H. Yu. Finding Collisions in the Full SHA1. In V. Shoup, editor, *Advances in Cryptology - Crypto'05*, volume 3621 of *Lectures Notes in Computer Science*, pages 17–36. Springer, 2005.
19. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology - Eurocrypt'05*, volume 3494 of *Lectures Notes in Computer Science*, pages 19–35. Springer, 2005.
20. X. Wang, H. Yu, and Y. Yin. Efficient Collision Search Attacks on SHA0. In V. Shoup, editor, *Advances in Cryptology - Crypto'05*, volume 3621 of *Lectures Notes in Computer Science*, pages 1–16. Springer, 2005.