Practical Collisions for EnRUPT

Sebastiaan Indesteege[1,2,⋆] and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
sebastiaan.indesteege@esat.kuleuven.be
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.

Abstract. The EnRUPT hash functions were proposed by O'Neil, Nohl and Henzen [5] as candidates for the SHA-3 competition, organised by NIST [4]. The proposal contains seven concrete hash functions, each having a different digest length.
We present a practical collision attack on each of these seven EnRUPT variants. The time complexity of our attack varies from 2^{36} to 2^{40} round computations, depending on the EnRUPT variant, and the memory requirements are negligible. We demonstrate that our attack is practical by giving an actual collision example for EnRUPT-256.

Key words: EnRUPT, SHA-3 candidate, hash function, collision attack.

1 Introduction

Cryptographic hash functions are important cryptographic primitives that are employed in a vast number of applications, such as digital signatures and commitment schemes. They are expected to possess several security properties, one of which is *collision resistance*. Informally, collision resistance means that it should be hard to find two distinct messages $m \neq m'$ that hash to the same value, i.e., $h(m) = h(m')$.

Many popular hash functions, such as MD5, SHA-1 and SHA-2 share a common design principle. The recent advances in the cryptanalysis of these hash functions have raised serious concerns concerning their long-term security. This motivates the design of new hash functions, based on different design strategies. The National Institute of Standards and Technology (NIST) has decided to hold a public competition, the SHA-3 competition, to develop a new cryptographic hash function standard [4].

The EnRUPT hash functions were proposed by O'Neil, Nohl and Henzen [5] as candidates for NIST's SHA-3 competition. The proposal contains seven concrete EnRUPT variants, each having a different digest length.

⋆ F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).

Table 1. EnRUPT Parameters

EnRUPT variant	digest length h	word size w	parallelisation level P	security parameter s	number of state words H
EnRUPT-128	128 bits	32 bits	2	4	8
EnRUPT-160	160 bits	32 bits	2	4	10
EnRUPT-192	192 bits	32 bits	2	4	12
EnRUPT-224	224 bits	64 bits	2	4	8
EnRUPT-256	256 bits	64 bits	2	4	8
EnRUPT-384	384 bits	64 bits	2	4	12
EnRUPT-512	512 bits	64 bits	2	4	16

In this paper, we analyse EnRUPT and show that none of the proposed EnRUPT variants is collision resistant. We present a practical collision attack requiring only 2^{36} to 2^{40} EnRUPT round computations, depending on the EnRUPT variant. This is significantly less than the $\mathcal{O}(2^{n/2})$ hash computations which are required for a generic collision attack based on the birthday paradox.

The structure of this paper is as follows. A short description of EnRUPT is given in Sect. 2. Section 3 introduces the basic strategy we use to find collisions for EnRUPT, which is based on the work by Chabaud and Joux [2] and Rijmen and Oswald [8]. Sections 4, 5 and 6 apply this basic attack strategy to EnRUPT, step by step. Our results, including an example collision for EnRUPT-256, are presented in Sect. 7. Finally, Sect. 8 concludes.

2 Description of EnRUPT

In this section, we give a short description of the seven EnRUPT variants that were proposed as SHA-3 candidates [5]. All share the same structure and use the same round function. The only differences lie in the parameters used. Table 1 gives the values of these parameters for each EnRUPT variant.

2.1 The EnRUPT Hash Functions

The structure shared by all EnRUPT hash functions can be split into four phases: preprocessing, message processing, finalisation and output. Figure 1 contains a description of the EnRUPT hash functions in pseudocode.

In the preprocessing phase (lines 2–4) the input message is padded to be a multiple of w bits, where w is the word size. Depending on the EnRUPT variant, the word size w is 32 or 64 bits, see Table 1. The padded message is then split into an integer number of w-bit words m_i.

The internal state of EnRUPT consists of several w-bit words: H state words x_i, P 'delta accumulators' d_i, and a round counter r. All of these are initialised to zero. The parameter P is equal to 2 for all seven EnRUPT variants. The value of H depends on the digest length, as indicated in Table 1.

Then, in the message processing phase (lines 5–8), the round function is called once for each w-bit padded message word m_i. Each call to the round function updates the internal state $\langle d, x, r \rangle$. A detailed description of the EnRUPT round function is given in the next section, Sect. 2.2.

After all message words have been processed, a finalisation is performed (lines 9–13). The EnRUPT round function is called once with the length of the (unpadded) message, represented as a w-bit unsigned integer. Then, H blank rounds, i.e., calls to the round function with a zero message word input, are performed.

Finally, in the output phase (lines 14–18), the message digest is generated one w-bit word at a time. The EnRUPT round function is called h/w times and, after each call, the content of the 'delta accumulator' d_0 is output.

2.2 The EnRUPT Round Function

The EnRUPT round function is based entirely on a number of simple operations on words of w bits, such as bit shifts, bit rotations, exclusive OR and addition modulo 2^w. The round function consists of $s \cdot P$ identical steps, where s and P are parameters of the hash function. As indicated in Table 1, $s = 4$ and $P = 2$ for all seven proposed EnRUPT variants. Thus, the EnRUPT round function consists of eight steps. Figure 2 gives a description of the EnRUPT round function in pseudocode.

In each step, several words of the state are selected (lines 3–7) and combined into an intermediate value f (lines 9–10). Note that line 10 could equally be described as a multiplication with 9 modulo 2^w. The intermediate value f is then used to update one state word, x_γ, and one 'delta accumulator', $d_{i \bmod P}$.

After all steps have been performed, the round counter is incremented by the number of steps that were carried out, $s \cdot P$. Finally, the input message word m is injected into one word of the internal state, the 'delta accumulator' d_{P-1}.

```
1: function EnRUPT ($M$)
2:     /* Preprocessing */
3:     $m_0, \cdots, m_t \leftarrow M \,||\, 1 \,||\, 0^{w-(|M|+1 \bmod w)}$    s.t  $\forall i, 0 \le i \le t : |m_i| = w$
4:     $d_0, \cdots, d_{P-1}, x_0, \cdots, x_{H-1}, r \leftarrow 0, \cdots, 0$
5:     /* Message processing */
6:     for $i = 0$ to $n$ do
7:         $\langle d, x, r\rangle \leftarrow \mathrm{round}(\langle d, x, r\rangle, m_i)$
8:     end for
9:     /* Finalisation */
10:    $\langle d, x, r\rangle \leftarrow \mathrm{round}(\langle d, x, r\rangle, \mathrm{uint}_w(|M|))$
11:    for $i = 1$ to $H$ do
12:        $\langle d, x, r\rangle \leftarrow \mathrm{round}(\langle d, x, r\rangle, 0)$
13:    end for
14:    /* Output */
15:    for $i = 0$ to $h/w - 1$ do
16:        $\langle d, x, r\rangle \leftarrow \mathrm{round}(\langle d, x, r\rangle, 0)$
17:        $o_i \leftarrow d_0$
18:    end for
19:    return $o_0 \,||\, \cdots \,||\, o_{h/w-1}$
20: end function
```

Fig. 1. The EnRUPT Hash Function

```
1: function round ($\langle d, x, r\rangle, m$)
2:     for $i = 0$ to $s \cdot P - 1$ do     /* An iteration of this loop is denoted a "step" */
3:         /* Compute indices */
4:         $\alpha \leftarrow r + (i + 1 \bmod P) \bmod H$
5:         $\beta \leftarrow r + i + 2P \bmod H$
6:         $\gamma \leftarrow r + i + P \bmod H$
7:         $\xi \leftarrow r + i \bmod H$
8:         /* Compute intermediate $f$ */
9:         $e \leftarrow ((x_\alpha \ll 1) \oplus x_\beta \oplus d_{i \bmod P} \oplus \mathrm{uint}_w(r + i)) \ggg w/4$
10:        $f \leftarrow (e \ll 3) \boxplus e$                 /* Multiplication with 9 modulo $2^w$ */
11:        /* Update state */
12:        $x_\gamma \leftarrow x_\gamma \oplus f$
13:        $d_{i \bmod P} \leftarrow d_{i \bmod P} \oplus x_\xi \oplus f$
14:    end for
15:    $r \leftarrow r + s \cdot P$
16:    $d_{P-1} \leftarrow d_{P-1} \oplus m$                          /* Message word injection */
17:    return $\langle d, x, r\rangle$
18: end function
```

Fig. 2. The EnRUPT Round Function

3 Basic Attack Strategy

This section gives an overview of the linearisation method for finding collision differential characteristics for a hash function, which we use to attack EnRUPT in this work. This method was introduced by Chabaud and Joux [2], who applied it to SHA-0 and simplified variants thereof. Later, it was extended further and applied to SHA-1 by Rijmen and Oswald [8].

A Linear Hash Function. Consider a hypothetical hash function that consists only of linear operations over GF(2). When the input messages are restricted to a certain length, each output bit can be written as an affine function of the input bits. The *difference* in each output bit is given by a linear function of the differences in the input bits, as the constants (if any) cancel.

A message difference that leads to a collision can be found by equating the output differences to zero, and solving the resulting system of linear equations over GF(2), for instance using Gauss elimination. Any pair of messages with this difference will result in a collision.

Linearising a Nonlinear Hash Function. Actual cryptographic hash functions contain (also) nonlinear components, so this method no longer applies. However, we may still be able to *approximate* the nonlinear components by linear ones and construct a linear approximation of the entire hash function. For our purpose, a good linear approximation $\lambda(x)$ of a nonlinear function $\gamma(x)$ is such that its differential behaviour is close to that of $\gamma(x)$. More formally, the equation

$$\gamma(x \oplus \Delta) \oplus \gamma(x) = \lambda(x \oplus \Delta) \oplus \lambda(x) = \lambda(\Delta) \tag{1}$$

should hold for a relatively large fraction of values x. For instance, an addition modulo 2^w could be approximated by a simple XOR operation, i.e., ignoring the carries.

Finding Collisions. A *differential characteristic* consists of a message difference and a list of the differences in all (relevant) intermediate values. For the linear approximation, it is easy to find a differential characteristic that leads to a collision with probability one. But for the actual hash function, this probability will be (much) lower.

If the differential behaviour of all the nonlinear components corresponds to that of the linear approximations they were replaced with, i.e., if (1) holds simultaneously for each nonlinear component, we say that the differential characteristic is followed. In this case, the message pair under consideration will not only collide for the linearised hash function, but also for the original, nonlinear hash function. Such a message pair is called a *conforming* message pair.

Hence, a procedure for finding a collision for the nonlinear hash function could be to find a differential characteristic leading to collisions for a linearised variant of the hash function. Then, a message pair conforming to the differential characteristic is searched. In order to lower the complexity of the attack, it is important to maximise the probability that the differential characteristic is followed, i.e., we need to find a good differential characteristic.

4 Linearising EnRUPT

We now apply this general strategy to EnRUPT. Recall the description of the EnRUPT round function in Fig. 2. Note that the only operation in this round function which is not linear over GF(2), is the modular addition in line 10. Indeed, the computation of the indices in lines 3–7 and the update of the round counter in line 15 do not depend on the message being hashed and can thus be precomputed. The same holds for the inclusion of the round counter in line 9, i.e., this can be seen as an XOR with a constant. The other operations are all linear over GF(2).

Replacing the modular addition in line 10 with an XOR operation yields a linearised round function, which we refer to as the EnRUPT-$\mathcal{L}$ round function. The EnRUPT-$\mathcal{L}$ hash function, i.e., the hash function built on this linearised round function, also consists solely of GF(2)-linear components.

5 The Collision Search

During the collision search phase, many collisions for EnRUPT-$\mathcal{L}$ are constructed, and a collision for EnRUPT is searched among them. Since only the modular additions (line 10 of Fig. 2) were approximated by XOR, these are the only places where the propagation of differences could differ between EnRUPT-$\mathcal{L}$ and EnRUPT. Instead of checking for a collision at the output, we can immediately check if the difference at the output of each modular addition, i.e., the difference Δf in the intermediate value f, still matches the differential characteristic.

5.1 An Observation on EnRUPT

We now make an important observation on the structure of the EnRUPT hash functions. It is possible to find a conforming message pair for a given differential characteristic one round at a time.

Consider the message word m_i, which is injected into the 'delta accumulator' d_{P-1} at the end of round i. In the first $(P-1)$ steps of the next round, d_{P-1} is not used, so m_i can not influence the behaviour of the modular additions in these steps. Starting from the P-th step of round $(i+1)$, however, m_i does have an influence.

We can search for a value for m_i such that the differential characteristic is followed up to and including the first $(P-1)$ steps of round $(i+2)$. Starting with the P-th step of round $(i+2)$, the next message word, m_{i+1} also influences the modular additions. Thus, we can keep m_i fixed, and use the new freedom available in m_{i+1} to ensure the differential characteristic is also followed for the next $s \cdot P$ steps.

This drastically reduces the expected number of trials required to find a collision. Let p_i denote the probability that the differential characteristic is followed in a block of $s \cdot P$ consecutive steps, starting at the P-th step of a round. Because we can construct a conforming message pair one word at a time, the expected number of trials is $\sum_i 1/p_i$ rather than $\prod_i 1/p_i$. In other words, the complexities associated with each block of $s \cdot P$ steps should be added together, rather than multiplied. This possibility was ignored in the security analysis of EnRUPT [5], leading to the wrong conclusion that attacks based on linearisation do not apply.

5.2 Accelerating the Collision Search

An simple optimisation can be made to the collision search, which will allow us to ignore the probability associated with one step in each round. This optimisation is analogous to Wang's 'single message modification', which was first introduced in the context of MD5 [10].

Consider the P-th step of a round. In this step, the 'delta accumulator' d_{P-1}, to which a new message word m was XORed at the end of the previous round, is used for the first time. More precisely, it is used in line 9 of Fig. 2 to compute the intermediate value e. Note however that these computations can be inverted. We can choose the value of e, and compute backwards to find what the message word m should be to arrive at this value of e.

The values of e which ensure that the difference propagation of the modular addition in line 10 of Fig. 2 corresponds to that of its linear approximation can be efficiently enumerated. Thus, rather than randomly picking values for m, we can efficiently sample *good* values for e in this step, and compute backwards to find the corresponding m. This ensures that the first modular addition affected by a message word m will always exhibit the desired propagation of differences. Thus, the P-th step of every round can be ignored in the estimation of the complexity of the attack.

6 Finding Good Differential Characteristics

The key to lowering the attack complexity is to find a good differential characteristic, i.e., a characteristic which is likely to be followed for the nonlinear hash function. A general approach to this problem, based on finding low weight codewords in a linear code, was proposed by Rijmen and Oswald [8] and extended by Pramstaller et al. in [7]. In this section, we show how to apply this approach to EnRUPT.

6.1 Coding Theory

As observed by Rijmen and Oswald [8], all of the differential characteristics leading to a collision for the linearised hash function can be seen as the codewords of a linear code.

Consider the EnRUPT-$\mathcal{L}$ hash function with a h-bit output length, and the message input restricted to messages of t message words. Since it is linear over GF(2), it is possible to express the difference in the output as a linear function of the difference in the input message m:

$$[\Delta o]_{1\times h} = [\Delta m]_{1\times tw} \cdot [\mathbf{O}]_{tw\times h} \quad . \tag{2}$$

As the modular additions, or rather the multiplications with 9, in the EnRUPT round function are approximated, we are also interested in the differences that enter each of these operations. For EnRUPT restricted to t message blocks, there are $t \cdot s \cdot P$ such operations in total. Hence, we can combine the input differences to these operations in a $1 \times tsPw$ bit vector Δe. Again, for the linear approximation, Δe is simply a linear function of the message difference Δm:

$$[\Delta e]_{1\times tsPw} = [\Delta m]_{1\times tw} \cdot [\mathbf{E}]_{tw\times tsPw} \quad . \tag{3}$$

Putting this together results in a linear code described by the following generator matrix

$$\mathbf{G} = \left[\mathbf{I}_{tw\times tw} \middle| \mathbf{E}_{tw\times tsPw} \middle| \mathbf{O}_{tw\times h}\right] \quad . \tag{4}$$

Each codeword contains a message difference, the input differences to all approximated modular additions, and finally the output difference.

Thus, each codeword is in fact simply a differential for EnRUPT-$\mathcal{L}$, and all differentials for EnRUPT-$\mathcal{L}$ are codewords of this code. To restrict ourselves to collision differentials, i.e., differentials ending in a zero output difference, we can use Gauss elimination to force the h rightmost columns of the generator matrix G to zero.

It is well known that the differential behaviour of modular addition can be approximated by that of XOR when the Hamming weight of the input difference, ignoring the most significant bit, is small [2,3,7,8]. As the input differences to the modular additions are part of the codewords, we will attempt to find a codeword with a low Hamming weight in this part of the codeword.

6.2 Low Weight Codewords

To find low weight codewords, we used a simple and straightforward algorithm that is based on the assumption that a codeword of very low weight exists in the code. For our purposes, this is a reasonable assumption, as only a very low weight codeword will lead to an attack faster than a generic birthday attack. The algorithm is related to the algorithm of Canteaut and Chabaud [1] and the algorithm used to find low weight codewords for linearised SHA-1 by Pramstaller et al. [7].

Let G be the generator matrix of the linear code as in (4). We randomly select a set I of (appropriate) columns of the generator matrix G and force them to zero using Gauss elimination, until only d rows remain, where d is a parameter of the algorithm. Then, the remaining space of 2^d codewords is searched exhaustively. This procedure is repeated until a codeword of sufficiently low weight is encountered. By replacing only the oldest column(s) in I, instead of restarting from the beginning every time, the algorithm can be implemented efficiently in practice.

If a codeword of very low weight exists in the code, it is likely that all of the columns in the randomly constructed set I will coincide with zeroes in the codeword, which implies that the codeword will be found in the exhaustive search. In the case of the codes originating from the seven linearised EnRUPT variants we consider, this algorithm finds a codeword of very low weight in a matter of minutes on a PC. Repeated runs of the algorithm always find the same codewords, so it is reasonable to assume that these are indeed the best codewords we can find.

6.3 Estimating the Attack Complexity

Actually, the weight of a codeword is only a heuristic for the attack complexity resulting from the corresponding differential. Codewords with a lower weight are expected to result in a lower attack complexity, but we can easily enhance our algorithm to optimise the actual attack complexity, rather than just a crude heuristic.

The Differential Probability. The probability that a differential characteristic is followed, is determined by the differences that are input to each of the multiplications with 9 (line 10 in Fig. 2) that were approximated using XOR operations. Denote by $\mathrm{DP}^{\times 9}(\mathit{\Delta})$ the probability that the propagation of differences through this nonlinear operation coincides with that of its linear approximation:

$$\mathrm{DP}^{\times 9}(\mathit{\Delta}) = \Pr_x \Big[(x \times 9) \oplus ((x \oplus \mathit{\Delta}) \times 9) = \mathit{\Delta} \oplus (\mathit{\Delta} \ll 3) \Big] \ . \tag{5}$$

The differential probability of modular addition was studied by Lipmaa and Moriai [3]. Applying their results to this situation, and taking into account that the three least significant bits of $(x \ll 3)$ are always zero, we find the following estimate for $\mathrm{DP}^{\times 9}(\mathit{\Delta})$:

$$\mathrm{DP}^{\times 9}(\mathit{\Delta}) \approx 2^{-\,\mathrm{wt}\left(\left(\mathit{\Delta} \vee (\mathit{\Delta} \ll 3)\right) \wedge \mathtt{01 \cdots 1000}_b\right)} \ . \tag{6}$$

Even though this estimate ignores the dependency between x and $(x \ll 3)$, this confirms the intuition that a difference $\mathit{\Delta}$ with a low Hamming weight (ignoring the most significant bit and the three least significant bits) results in a large probability $\mathrm{DP}^{\times 9}(\mathit{\Delta})$. We use this as a heuristic to find a good differential characteristic: we want to minimise the Hamming weight of the relevant parts of the differences that are input to the modular additions. In other words, we want to find a low weight codeword of the aforementioned linear code, where only the bits that impact $\mathrm{DP}^{\times 9}(\mathit{\Delta})$ are counted.

Exact Computation of the Differential Probability. Computing the exact value of $\mathrm{DP}^{\times 9}(\mathit{\Delta})$ for any given difference $\mathit{\Delta}$ can be done by counting all the values x for which the differences propagation is as predicted by the linear approximation. This can be done efficiently as the modular addition can be represented compactly as a trellis, where each path through the trellis corresponds to a 'good' value of x. Using a slight variant of the Viterbi algorithm [9], the number of paths in the trellis can be counted efficiently. While this is very useful for evaluating the attack complexity, it lacks the intuition we can gather from (6).

Computing the Attack Complexity. Let $p_{r,i}$ be the differential probability associated with the modular addition in step i of round r of the differential characteristic. Recall the observation made in Sect. 5.1, i.e., finding a conforming message pair can be done one round at a time, or rather one message word at a time, as this does not coincide precisely with the round boundaries. Taking this into account, the complexity of finding the j-th word of a conforming message pair can thus be computed as

$$C_j = \left(\prod_{i=P-1}^{sP-1} \frac{1}{p_{j+1,i}} \right) \left(\prod_{i=0}^{P-2} \frac{1}{p_{j+2,i}} \right) \; . \tag{7}$$

Due to the acceleration technique presented in Sect. 5.2, we can set $p_{P-1} = 1$. With the default EnRUPT parameters (see Table 1), this then becomes

$$C_j = \frac{1}{p_{j+1,2}} \cdot \frac{1}{p_{j+1,3}} \cdot \frac{1}{p_{j+1,4}} \cdot \frac{1}{p_{j+1,5}} \cdot \frac{1}{p_{j+1,6}} \cdot \frac{1}{p_{j+1,7}} \cdot \frac{1}{p_{j+2,0}} \; . \tag{8}$$

Finally, as was explained in Sect. 5.1, note that each message word can be found independent of the previous ones, due to the newly available degrees of freedom in each message word. Hence, the overall attack complexity can simply be computed as the sum of these round complexities:

$$C_{\text{tot}} = \sum_{j=0}^{t} C_j \; . \tag{9}$$

Note that, given a differential characteristic, it is easy to compute the associated attack complexity. Hence, when searching for a good differential characteristic using the algorithm described in Sect. 6.2, we can use the actual attack complexity instead of the weight of the codeword. The algorithm still implicitly uses the weight of a codeword as a heuristic, but now attempts to optimise the actual attack complexity.

7 Results and Discussion

We constructed differential characteristics for each of the seven EnRUPT variants in the EnRUPT SHA-3 proposal [5]. Table 2 lists the attack complexity and the length of the best characteristic we found for each variant. Recall that we fixed the length of the characteristic a priori. Note however that nothing prevents our search algorithm from proposing a shorter characteristic, padded with rounds without any difference,

Table 2. Summary of our attacks. Only the best attack is listed for each EnRUPT variant.

EnRUPT variant	estimated time complexity [EnRUPT rounds]	length of collision differential [message words]
EnRUPT-128	$2^{36.04}$	6
EnRUPT-160	$2^{37.78}$	7
EnRUPT-192	$2^{38.33}$	8
EnRUPT-224	$2^{37.02}$	6
EnRUPT-256	$2^{37.02}$	6
EnRUPT-384	$2^{39.63}$	8
EnRUPT-512	$2^{38.46}$	10

which we also observed in practice. We experimented with (much) longer maximum characteristic lengths, but found no better long characteristics.

The time complexities vary from 2^{36} to 2^{40} round computations, depending on the EnRUPT variant, which is remarkable. It means that the collision resistance in absolute terms of each of these EnRUPT variants is more or less the same, regardless of the digest length. Relative to the expected collision resistance of $\mathcal{O}(2^{n/2})$ for an n-bit hash function, however, the (relative) collision resistance of EnRUPT is much lower for the variants with a longer digest length than for those with a shorter digest length.

As an example, Table 3 lists a differential characteristic for EnRUPT-256 with an associated attack complexity of 2^{37} EnRUPT round computations. The table also includes the differential probabilities of each step, which were used to compute the attack complexity. A star ('$\star$') indicates that the differential probability can be ignored in that step because of the technique presented in Sect. 5.2. A collision example for EnRUPT-256, obtained using this characteristic, is given in Table 4.

Discussion. In response to these collision attacks, the designers of EnRUPT proposed to double the s parameter to 8 [6]. As a consequence of this, the number of steps between two message word injections is doubled. Experiments with these EnRUPT variants indicate that this tweak seems to be effective at stopping the attacks described in this paper. For EnRUPT-256 with $s = 6$, we were still able to find a differential with an associated attack complexity of about 2^{110} EnRUPT rounds, which is still below the birthday bound. For higher values of the s parameter, all the differential characteristics we could find would result in attack com-

Table 3. Differential Characteristic for EnRUPT-256

Round	Step	Δe	$\rightarrow$	Δf	DP$^{\times 9}$	totals
inject message word difference Δm_{-1} = 0000000008000000$_x$						
0	0	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	$\mathbf{2^{-0.00}}$
	1	0000000000000800$_x$	$\rightarrow$	0000000000004800$_x$	$\star$	
	2	9000000000000000$_x$	$\rightarrow$	1000000000000000$_x$	$2^{-0.85}$	
	3	4800000000000800$_x$	$\rightarrow$	0800000000004800$_x$	$2^{-3.70}$	
	4	9000000000000000$_x$	$\rightarrow$	1000000000000000$_x$	$2^{-0.85}$	
	5	4800280000000800$_x$	$\rightarrow$	0801680000004800$_x$	$2^{-7.28}$	
	6	90000002d0000000$_x$	$\rightarrow$	1000001450000000$_x$	$2^{-6.43}$	
	7	0000280168000800$_x$	$\rightarrow$	0001680a28004800$_x$	$2^{-11.02}$	
inject message word difference Δm_0 = 0000002280000000$_x$						
1	0	90000002d0000000$_x$	$\rightarrow$	1000001450000000$_x$	$2^{-6.43}$	$\mathbf{2^{-36.56}}$
	1	0000280168000000$_x$	$\rightarrow$	0001680a28000000$_x$	$\star$	
	2	90000002d0000000$_x$	$\rightarrow$	1000001450000000$_x$	$2^{-6.43}$	
	3	4800280000000000$_x$	$\rightarrow$	0801680000000000$_x$	$2^{-5.43}$	
	4	90000002d0000000$_x$	$\rightarrow$	1000001450000000$_x$	$2^{-6.43}$	
	5	0000080000000000$_x$	$\rightarrow$	0000480000000000$_x$	$2^{-1.85}$	
	6	9000000240000000$_x$	$\rightarrow$	1000001040000000$_x$	$2^{-3.70}$	
	7	4800080120000000$_x$	$\rightarrow$	0800480820000000$_x$	$2^{-6.54}$	
inject message word difference Δm_1 = 0000002288000000$_x$						
2	0	9000000240000000$_x$	$\rightarrow$	1000001040000000$_x$	$2^{-3.70}$	$\mathbf{2^{-34.08}}$
	1	0000080048000000$_x$	$\rightarrow$	0000480208000000$_x$	$\star$	
	2	9000000240000000$_x$	$\rightarrow$	1000001040000000$_x$	$2^{-3.70}$	
	3	4800080168000000$_x$	$\rightarrow$	0800480a28000000$_x$	$2^{-9.28}$	
	4	9000000240000000$_x$	$\rightarrow$	1000001040000000$_x$	$2^{-3.70}$	
	5	0000200000000000$_x$	$\rightarrow$	0001200000000000$_x$	$2^{-1.85}$	
	6	9000000000000000$_x$	$\rightarrow$	1000000000000000$_x$	$2^{-0.85}$	
	7	4800200000000000$_x$	$\rightarrow$	0801200000000000$_x$	$2^{-3.70}$	
inject message word difference Δm_2 = 0000000208000000$_x$						
3	0	9000000000000000$_x$	$\rightarrow$	1000000000000000$_x$	$2^{-0.85}$	$\mathbf{2^{-23.91}}$
	1	0000280120000000$_x$	$\rightarrow$	0001680820000000$_x$	$\star$	
	2	9000000090000000$_x$	$\rightarrow$	1000000410000000$_x$	$2^{-3.70}$	
	3	4800280168000000$_x$	$\rightarrow$	0801680a28000000$_x$	$2^{-11.02}$	
	4	9000000090000000$_x$	$\rightarrow$	1000000410000000$_x$	$2^{-3.70}$	
	5	0000080048000000$_x$	$\rightarrow$	0000480208000000$_x$	$2^{-4.70}$	
	6	9000000090000000$_x$	$\rightarrow$	1000000410000000$_x$	$2^{-3.70}$	
	7	4800080000000000$_x$	$\rightarrow$	0800480000000000$_x$	$2^{-3.70}$	
inject message word difference Δm_3 = 0000000200000000$_x$						
4	0	9000000090000000$_x$	$\rightarrow$	1000000410000000$_x$	$2^{-3.70}$	$\mathbf{2^{-34.19}}$
	1	0000080000000800$_x$	$\rightarrow$	0000480000004800$_x$	$\star$	
	2	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	
	3	0000080000000800$_x$	$\rightarrow$	0000480000004800$_x$	$2^{-3.70}$	
	4	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	
	5	4800080048000800$_x$	$\rightarrow$	0800480208004800$_x$	$2^{-8.39}$	
	6	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	
	7	4800080048000800$_x$	$\rightarrow$	0800480208004800$_x$	$2^{-8.39}$	
inject message word difference Δm_4 = 0000000200000000$_x$						
5	0	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	$\mathbf{2^{-20.49}}$
	1	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$\star$	
	$\vdots$	$\vdots$	$\rightarrow$	$\vdots$	$\vdots$	
	7	0000000000000000$_x$	$\rightarrow$	0000000000000000$_x$	$2^{-0.00}$	$\mathbf{2^{-0.00}}$

Table 4. A Collision Example for EnRUPT-256

M	13$_x$, c8$_x$, 4b$_x$, 45$_x$, 62$_x$, 70$_x$, 17$_x$, 6e$_x$, 04$_x$, f9$_x$, 31$_x$, 7e$_x$, c3$_x$, 6c$_x$, e7$_x$, d3$_x$, e1$_x$, 21$_x$, 78$_x$, 6a$_x$, 34$_x$, 74$_x$, 11$_x$, 19$_x$, 7f$_x$, 64$_x$, a3$_x$, c9$_x$, 40$_x$, 07$_x$, 75$_x$, 76$_x$, a1$_x$, 4f$_x$, 90$_x$, 86$_x$, fd$_x$, c7$_x$, 33$_x$, 4a$_x$, 41$_x$, 3a$_x$, 76$_x$, 91$_x$, 96$_x$, 06$_x$, 2c$_x$, a1$_x$.
M'	13$_x$, c8$_x$, 4b$_x$, 45$_x$, 6a$_x$, 70$_x$, 17$_x$, 6e$_x$, 04$_x$, f9$_x$, 31$_x$, 5c$_x$, 43$_x$, 6c$_x$, e7$_x$, d3$_x$, e1$_x$, 21$_x$, 78$_x$, 48$_x$, bc$_x$, 74$_x$, 11$_x$, 19$_x$, 7f$_x$, 64$_x$, a3$_x$, cb$_x$, 48$_x$, 07$_x$, 75$_x$, 76$_x$, a1$_x$, 4f$_x$, 90$_x$, 84$_x$, fd$_x$, c7$_x$, 33$_x$, 4a$_x$, 41$_x$, 3a$_x$, 76$_x$, 93$_x$, 96$_x$, 06$_x$, 2c$_x$, a1$_x$.
EnRUPT-256(M) = EnRUPT-256(M') =	bd$_x$, 67$_x$, 51$_x$, 7c$_x$, a6$_x$, c0$_x$, 41$_x$, 20$_x$, 82$_x$, e0$_x$, 3b$_x$, 74$_x$, 5f$_x$, fc$_x$, 4a$_x$, 64$_x$, e9$_x$, f0$_x$, 92$_x$, c2$_x$, 58$_x$, c3$_x$, 98$_x$, b8$_x$, 44$_x$, 9a$_x$, fe$_x$, cb$_x$, 7f$_x$, c8$_x$, 6f$_x$, 72$_x$.

plexities that are far greater than the birthday bound, and thus should not be considered to be real attacks.

Note that the failure of this heuristic attack method for $s = 8$ does not preclude the possibility of attacks based on linearisation. Our experiments only show that it is unlikely that the particular attack used in this work can be applied directly to EnRUPT with $s = 8$.

8 Conclusion

We presented collision attacks on all seven variants of the EnRUPT hash function [5] that were proposed as candidates to the NIST SHA-3 competition [4]. The attacks require negligible memory and have time complexities ranging from 2^{36} to 2^{40} EnRUPT round computations, depending on the EnRUPT variant. The practicality of the attacks has been demonstrated with an example collision for EnRUPT-256.

Acknowledgements

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The collision example for EnRUPT-256 was obtained utilizing high performance computational resources provided by the University of Leuven, `http://ludit.kuleuven.be/hpc`.

References

1. Anne Canteaut and Florent Chabaud, *"A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511,"* IEEE Transactions on Information Theory, vol. 44, nr. 1, pp. 367–378, 1998.
2. Florent Chabaud and Antoine Joux, *"Differential Collisions in SHA-0,"* In Advances in Cryptology – CRYPTO 1998, Lecture Notes in Computer Science, vol. 1462, pp. 56–71, Springer, 1998.
3. Helger Lipmaa and Shiho Moriai, *"Efficient Algorithms for Computing Differential Properties of Addition,"* In Fast Software Encryption – FSE 2001, Lecture Notes in Computer Science, vol. 2355, pp. 336–350, Springer, 2002.
4. National Institute of Standards and Technology, *"Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family,"* Federal Register, vol. 72, nr. 212, pp. 62212–62220, November 2007.
5. Sean O'Neil, Karsten Nohl and Luca Henzen, *"EnRUPT Hash Function Specification,"* Submission to the NIST SHA-3 competition, 2008. Available online at `http://www.enrupt.com/SHA3/`.
6. Sean O'Neil, personal communication, 20 January 2009.
7. Norbert Pramstaller, Christian Rechberger and Vincent Rijmen, *"Exploiting Coding Theory for Collision Attacks on SHA-1,"* In Cryptography and Coding, 10th IMA International Conference, Lecture Notes in Computer Science, vol. 3796, pp. 78–95, Springer, 2005.
8. Vincent Rijmen and Elisabeth Oswald, *"Update on SHA-1,"* In Topics in Cryptology – CT-RSA 2005, Lecture Notes in Computer Science, vol. 3376, pp. 58–71, Springer, 2005.
9. Andrew J. Viterbi, *"Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,"* IEEE Transactions on Information Theory, vol. 13, nr. 3, pp. 260–269, 1967.
10. Xiaoyun Wang and Hongbo Yu, *"How to Break MD5 and Other Hash Functions,"* In Advances in Cryptology – EUROCRYPT 2005, Lecture Notes in Computer Science, vol. 3494, pp. 19–35, Springer, 2005.