Detectable correlations in Edon-$\mathcal{R}$

Peter Novotney
peternov@microsoft.com

Niels Ferguson
niels@microsoft.com

July 31, 2009

Abstract

The Edon-$\mathcal{R}$ compression function has a large set of useful differentials that produce easily detectable output bit biases. We show how to construct such differentials, and use them to create a distinguisher for Edon-$\mathcal{R}$-512 that requires around 2^{54} compression function evaluations (or 2^{28} evaluations after a pre-computation of 2^{66} evaluations). The differentials can also be used to attack a variety of MAC and KDF constructions when they use Edon-$\mathcal{R}$-512.

1 Introduction

Edon-$\mathcal{R}$ [1] is one of the candidate hash functions in the NIST SHA-3 competition.[1] It performs fewer operations per input bit than any of the other candidate functions. This makes it the fastest candidate by a significant margin [2], but also a tempting target for cryptanalysis.

One surprising property of Edon-$\mathcal{R}$ is that out of the 14 nonlinear bijective mappings used in the compression function, 7 have inputs that depend only on the message block and not on the previous chaining state. This allows the attacker to fully predict the propagation of values and differences in these functions. Due to the internal structure a differential from the message block can bypass another 4 nonlinear functions leaving only 3 'active' nonlinear functions that a differential has to pass through.

Our basic attack is a distinguishing attack. We show that an attacker can find two strings L and L' such that the function $f : X \mapsto H(X|L) \oplus H(X|L')$

[1] As we were finalizing this paper, NIST announced the round 2 candidates for the SHA-3 competition. Edon-$\mathcal{R}$ was not selected for round 2.

has easily detectable biases when H is the Edon-$\mathcal{R}$ hash function. For an ideal hash function, f behaves like a random mapping and does not have biases.

The attack can be extended to recover the intermediate hash state just before the last block, which breaks a number of common usage patterns for hash functions such as some KDF and MAC constructions.

2 An overview of Edon-$\mathcal{R}$

We give a short overview of those parts of Edon-$\mathcal{R}$ that are used in our attack. More details can be found in the Edon-$\mathcal{R}$ specifications [1].

Let $n \in \{256, 512\}$ be the output size of the hash function. (The other output sizes are simple variations of these two sizes, which we will ignore.)

Given a message M the first step is to pad it. We append a single '1' bit, and as many '0' bits as needed to make the length $2n - 64 \bmod 2n$. We then append the length of M as a 64-bit integer to get a padded message whose length is a multiple of $2n$ bits.

The padded message is split into $2n$-bit blocks $M_0, \ldots, M_{k-1}$ where $k = \lceil (\text{length}(M) + 65)/2n \rceil$. The blocks are processed by iterating the compression function:

$$H_0 := \text{some constant}$$
$$H_{i+1} := C(H_i, M_i)$$

The chaining values H_i are each $2n$ bits long; the result of Edon-$\mathcal{R}$ consists of one half of the final chaining value H_k.

Our attack involves the last compression function, shown in figure 1. The lines are n-bit values; each n-bit value is internally represented as a vector of 8 words each of 32 bits (for the $n = 256$ case) or 64 bits (for the $n = 512$ case). At the top we have the two halves of the message block M_a and M_b. The functions f and g are nonlinear bijections on n bits, and R is a function that reversed the order of the 8 words in the vector. The addition boxes represent word-by-word addition. The two halves of the chaining state come in as H_a and H_b and the final result of the hash function is H at the bottom. The colors relate to some details of our attack and can be ignored for the moment.

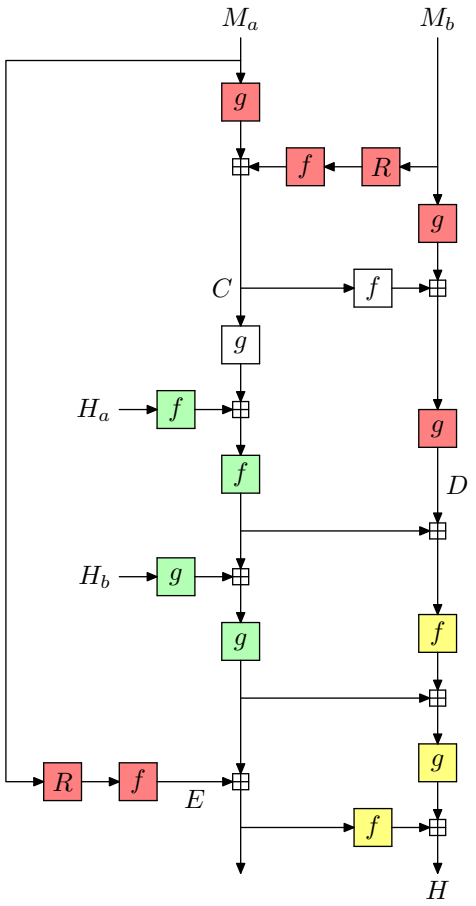

Figure 1: Edon-$\mathcal{R}$ compression function

The f and g functions are the nonlinear elements used in Edon-$\mathcal{R}$. They have limited diffusion; each input bit will affect at least 15 of the output bits, but that is nowhere near full diffusion for a 256-bit or 512-bit function.

Our presentation is a little bit different from the one used in the Edon-$\mathcal{R}$ specifications. The Quasigroup operation $A * B$ from [1] can be written as $f(A) + g(B)$ where f and g can be expressed in terms of the π functions from [1] section 2.1.2 as

$$f(x) := \pi_1(\pi_2(x))$$
$$g(x) := \pi_1(\pi_3(x))$$

These are the f and g boxes in our figure. In the canonical description there are 16 of these functions. In two cases, the same function is applied twice to the same data; we have optimized that in our figure and have only 14 f and g functions.

3 Our attack

Our attack is a differential attack. We treat the chaining value (H_a, H_b) as unknown and try to find a differential from (M_a, M_b) to H. If we can find an input difference that leads to a detectable bias in the output, then we have a distinguishing attack on Edon-$\mathcal{R}$.

In more detail, our attack finds a length m and two strings L and L' such that $H(X|L) \oplus H(X|L')$ has biased bits when X varies over all m-bit strings. We always choose m to be a multiple of $2n$; we can then treat the hashing of X as choosing a random chaining value (H_a, H_b) as chaining input to the last compression function, and the strings L consist of the message in the last message block.

We use the names of intermediate values as shown in figure 1. For the differential, (H_a, H_b) is fixed; C, D, E, and H are values in the compression function that processes L, and C', D', E', and H' values in the compression function that processes L'.

To reduce the mixing of the message and the chaining value we always choose L and L' such that $C = C'$. This means that the white functions in the figure have inputs that do not change in our differential. The red Functions have inputs that depend only on the message, which is known. The green functions have inputs that depend only on the chaining value. The three yellow functions are the only ones whose inputs depend on both the message and the chaining value.

3.1 Biases when ignoring the padding

We first show how we can construct a differential from (M_a, M_b) to H if we ignore the padding that is always part of M_b. We choose a random fixed value for C and choose low Hamming-weight values for D and D'. (Thus, D and D' have most bits set to 0.) As both f and g are invertible, these values determine M_a, M_b, E, M'_a, M'_b, and E'.

w	median bias	largest bias
1	$\approx 2^{-2.9}$	$\approx 2^{-1.7}$
2	$\approx 2^{-3.6}$	$\approx 2^{-2.7}$
4	$\approx 2^{-5.8}$	$\approx 2^{-4.6}$
6	$\approx 2^{-8.4}$	$\approx 2^{-6.1}$
8	$\approx 2^{-11.6}$	$\approx 2^{-10.3}$
10	$\approx 2^{-13.6}$	$\approx 2^{-10.9}$

Table 1: Biases of the most biased output bit for D value of weight w

The values H_a and H_b represent the intermediate result of hashing the string X. To measure biases in $H(X \mid L) \oplus H(X \mid L')$ we choose random values for (H_a, H_b) and compute the compression function with this chaining value and both (M_a, M_b) and (M'_a, M'_b) to get H and H'. We then look at the bits of $H \oplus H'$ for biases taken over the random choice of the chaining values.

Our differential consists of two paths; a low Hamming-weight difference from D going down through two functions, and a heavy differential from E going through one function. These two differences are combined to give the difference in H.

We experimentally measured the biases this produces in Edon-$\mathcal{R}$-512. For each maximum weight w we ran 10 experiments. In each experiment we chose C random, and chose D and D' randomly in the set of all values with Hamming weight w. We then computed the corresponding M_a, M'_a, M_b, M'_b, E, and E', and finally computed $H \oplus H'$ for 2^{30} random values of (H_a, H_b). We then measured the bias of the most biased bit. Table 1 reports the median and largest bias of our 10 experiments for each of the maximum weights w. For a truly random function, we'd expect one of the 512 output bits to have a bias of around 3.1 standard deviations, which is a bias of $2^{-14.4}$ for our 2^{30} samples. As can be seen, the biases in Edon-$\mathcal{R}$-512 are easily detectable. Even the median bias for $w = 10$ is 5.2 standard deviations away from the mean.

3.2 Dealing with the padding

The procedure above does not produce an M_b and M'_b with suitable padding. We can construct a differential that respects the padding rules using some more computing power.

L	B9E8C2EB4052E4A897599BAE4E429C7015C5D754EA06AE2C1B7BD38706DA9EF4 3329A53CDD47883F63E72A67917E4BBF64983BB7E50B9C0CCBE9A04C23158B5F 28687DBE5D5063EA85AFBDDD839DB59A1AFC715B4469EB056320447244C3B302 76A1020D19507242CD5E081FBCF17C793366B7D2BE63A285BF333E2F3E119427
L'	5D57AE8FCA5E979AD6A78D0C4213D42A32DDFE07C394C2F4CD0140A1B44ECEE2 EFEC661C5DB2DA5FA4EF9A40672C7CC679E93CA5207F1C6DCDA6F81C9E7574CF 045CA1D71E9B634E0EA06AA3A4F00F3F73FB75DD3C11194DE92AF59AE360FF9C CBB512243ABAE0A25FBFC6D8412E935B79B15F1188CC225FBF333E2F3E119427

Table 2: Trailing strings for $m = 2851923422810615808$ with bias $2^{-6.6}$

To get the most freedom, we restrict ourselves to Edon-$\mathcal{R}$-512 and always choose our last message block to contain $2n-65$ bits of message, one padding bit, and 64 bits of length encoding.

If we are given a length m for the string X then we have a 65-bit restriction on the value of M_b and M'_b. We thus expect to have to try 2^{65} different values for D before we find one with the right padding value. (There are $\binom{512}{10} \approx 2^{68}$ values of Hamming-weight 10, so we can use $w = 10$.) Another 2^{65} tries will produce a suitable value for D' so there is a one-off cost of 2^{66} to find suitable L and L' for a given length m.

Thus, for any length of X that is a multiple of $2n$ we can, in about 2^{66} operations, find values for L and L' that produce easily detectable biases.

If the length m is only partially specified or can be freely chosen, we can do better. We choose random D values in our low Hamming-weight set and we keep those whose corresponding M_b has an acceptable length value. (There are 11 bits in M_b that always have to have an exact value; the padding bit must be '1' and the 10 least significant bits of the length field must encode the integer $1024 - 65 = 959$.) Once we collect enough suitable values for D we will find two that have the same length value.

We implemented this variant with no restriction on the length (other than the 11 bits mentioned above) and for length $m = 2851923422810615808$ found the strings L and L' which are shown, including the padding, in table 2. To generate such pairs we have to try 2^{11} values for D to generate one valid M_b value, and then collect 2^{27} valid M_b values before we get a collision on the 54 remaining length bits. Thus the total computational cost of finding the L's is around 2^{38}. This took less than a day on one of our

home machines. This pair produces an output bias in one bit of $2^{-6.6}$.

The bias produced by the L values is easily measured by computing just the last block with random chaining inputs. But to measure the bias using only the full hash function requires the hashing of very long X values. If we want to minimize the overall cost of creating the L values and verifying the bias using the full hash function we can restrict m to be at most 2^k for some k. We have to try $2^{11+64-k}$ values for D to get a valid M_b value, and then collect $2^{(k-10)/2}$ valid M_b values to create the collision on the length value. Finally, detecting a bias of 2^{-14} (for $w = 10$) requires around 2^{28} evaluations each of which uses $2 \cdot 2^{k-10}$ block computations for a cost of 2^{k+19}. The total cost is thus $2^{11+64-k+k/2-5} + 2^{k+19} = 2^{70-k/2} + 2^{k+19}$ which is minimal when we choose $k \approx 34$. We thus estimate that finding suitable L and L' values and then detecting the resulting bias on the full hash function can be done in about 2^{54} compression function evaluations.

3.3 Further attacks

Suppose we have an oracle with an unknown string K that is a multiple of $2n$ bits long. On input of a non-empty string L the oracle returns $H(K \mid L)$. We can use our differentials to recover the intermediate state after hashing K, and thus impersonate the oracle in future.

We use our differentials in the same way differentials are used in key-recovery attacks on block ciphers. We think of (H_a, H_b) as the 'key', the green functions in figure 1 as the key schedule, and the yellow functions as the block cipher. We generate a large set of differential pairs (L, L') for the length of K. We then guess the value for one or more bits of the last 'round key' (e.g. the output of the lowest green g function) and experimentally compute the expected bias for each of our differential pairs for this guess. We then compare that to the actual results. With enough (L, L') pairs it quickly becomes obvious what the right value is for the key bit. Once we know a few of the key bits, the biases will tend to increase and make our work even simpler.

Our biases for $w = 1$ are in the order of 2^{-3} so we need around 2^6 differential pairs for one bit. (We can choose a new random C value for each pair so we don't have to use heavier values for D.) It costs 2^{66} to produce each differential pair so the total cost of the attack is around 2^{72} per recovered bit. Thus, we expect that the full (H_a, H_b) state can be recovered in around

2^{16} queries and 2^{82} computational steps.

There are several common constructions that are susceptible to this type of attack. For example, many key derivation functions, including NIST SP800-56A, can be attacked in this way, giving the attacker the power to compute all derived keys.

Also $\mathrm{MAC}(K, M) := H(K|M)$ is a strong MAC function if H is a good hash function, but our attack allows existential forgeries in around 2^{15} queries after a pre-computation of 2^{82} steps when Edon-$\mathcal{R}$-512 is used as the hash function.

3.4 Edon-$\mathcal{R}$-256

We have not tried our methods on Edon-$\mathcal{R}$-256. Because the block size is smaller the diffusion is slightly better, so we expect the workload of the attack to increase somewhat. We think it is likely that applying our techniques to Edon-$\mathcal{R}$-256 will result in an attack, but the computational cost might be too large for us to generate an actual example.

3.5 Possible improvements

Our attack is the result of a very preliminary analysis of Edon-$\mathcal{R}$. Rather than study the propagation of differentials through the Edon-$\mathcal{R}$ function we used brute computational force to show that correlations exist. This takes less time, but it ignores a lot of the structure of the function, and thus misses out on many opportunities to improve the attack. Below are just some of the areas that we believe improvements can be made in:

better differentials Our choices for D and D' have been purely random in the set of values with weight w, and we have computed the resulting output biases experimentally. Even within the small set of experiments that we ran we found that some differences lead to much higher biases than other differences. A more detailed analysis of the differential propagation will no doubt result in ways of finding better differentials.

subtraction vs. xor We looked at $H(X \mid L) \oplus H(X \mid L')$, but given that the last operation in the compression function is an addition, it might be interesting to look at $H(X \mid L) \boxminus H(X \mid L')$ where $\boxminus$ is the word-

wise subtraction. This preserves the group structure of the last mixing operations and might lead to better biases.

Multi-bit correlations For simplicity we have limited ourselves to single bit biases. We expect that analyzing multiple output bits together (e.g. using a χ^2 test) will produce biases that are more easily detectable.

More attention to detail In several places we ignore details that can help the attacker, or use a simple but pessimistic estimate of the effectiveness of the attack. A more detailed analysis should improve our attacks.

4 Comments on Edon-$\mathcal{R}$

Looking at figure 1 it is surprising to see how much processing is done on the message block without involving the chaining value. Half of the 14 nonlinear bijections have inputs that do not depend on the chaining value.

If we rewrite Edon-$\mathcal{R}$ a bit, we can think of the pair $(g(C), D)$ as the message block. The 7 red and white functions become an expensive message expansion function that computes a third block value E. The remaining 7 nonlinear functions perform the actual compression. In this representation the padding rule becomes complicated, but that affects only the last block.

Intuitively this feels like an inefficient use of computational resources. Half the time is spent in the message expansion to compute a single extra block that then affects the output of the compression function almost directly.

Another question is whether it is useful to apply the f and g functions to H_a and H_b respectively. These would be useful if an attacker could get non-random patterns in the chaining value, but an attacker that can do that can create non-random patterns in the hash function output too.

An alternate design for a compression function based on 14 nonlinear permutations would be to build a block cipher using a 14-round Feistel network with a very simple key schedule, and run this in one of the standard hashing modes. This would achieve a similar speed as Edon-$\mathcal{R}$ in software, but it would seem to be much harder to attack.

4.1 Edon-$\mathcal{R}$'s proof of security against differential cryptanalysis

In [1] section 3.5 the Edon-$\mathcal{R}$ submitters provide a proof that Edon-$\mathcal{R}$ is secure against differential cryptanalysis. They show that a single bit difference in M_a or M_b will not lead to a detectable difference patterns in the output.

We believe this analysis is incomplete. It shows that a single-bit input difference does not lead to a detectable output difference, but it does not take differentials into account that start out with many bits, then narrow down to one or just a few bits halfway through the computation, and then fan out again. From experience we know that the highest probability differentials are often of this form, and the proof provides no upper bound on their probability.

Our attack is exactly of that form. We have a big difference in the message block which narrows down to a low Hamming-weight difference halfway through the computation.

5 Acknowledgements

We would like to thank Danilo Gligoroski and the other members of the Edon-$\mathcal{R}$ team for their encouragement and support. They were also kind enough to provide us with the description of the inverse f and g functions.

6 Conclusion

Edon-$\mathcal{R}$ has insufficient mixing between the message block and the chaining state. This leads to message differentials with detectable biases in the output, which can be used to recover the chaining state input to the last compression function if the attacker controls only the last message block. This breaks a variety of protocols and algorithms in which hash functions are used.

References

[1] Danilo Gligoroski, Rune Steinsmo Ødesgård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, "Cryptographic Hash Function EDON-$\mathcal{R}$" `http://people.item.ntnu.no/~danilog/Hash/Edon-R/Supporting_Documentation/EdonRDocumentation.pdf`, Submission to NIST, 2008

[2] Skein team. "Engineering comparison of SHA-3 candidates", `http://www.skein-hash.info/sha3-engineering`, retrieved April 19, 2009.