Key Recovery Attack against Secret-prefix EDON-$\mathcal{R}$

Gaëtan Leurent

École Normale Supérieure – Département d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France
Gaetan.Leurent@ens.fr

Abstract. EDON-$\mathcal{R}$ is a SHA-3 candidate. In this paper we show that using EDON-$\mathcal{R}$ as a MAC with the secret prefix construction is unsafe. Our attack requires 2 queries, $2^{5n/8}$ computations, and negligible memory.
This does not directly contradict the security claims of EDON-$\mathcal{R}$ or the NIST requirements for SHA-3, but we believe it shows a strong weakness in the design.
Key words: Hash functions, SHA-3, EDON-$\mathcal{R}$, MAC, secret prefix, key recovery

1 Description of EDON-$\mathcal{R}$

EDON-$\mathcal{R}$ is a wide-pipe iterative design, based on a compression function $\mathcal{R}$, with a final truncation $\mathcal{T}$. The EDON-$\mathcal{R}$ family is based on two main designs: EDON-$\mathcal{R}$256 uses 32-bits words, while EDON-$\mathcal{R}$512 uses 64-bit words. The compression function is based on a quasi-group operation $*$, which take two inputs X and Y in $(\mathbb{F}_2^w)^8$ (*i.e.* 8 w-bit words) and compute one output in $(\mathbb{F}_2^w)^8$. It can be described as:

$$\begin{aligned} X * Y &= \pi_1(\pi_2(X) + \pi_3(Y)) \\ &= \pi_1(\pi_2(X)) + \pi_1(\pi_3(Y)) \\ &= Q_0(R_0(P_0(X))) + Q_1(R_1(P_1(Y))) \end{aligned}$$

where

- $+$ is a component-wise addition modulo 2^w (w is the word size);
- π_1, π_2, π_3 are the permutations used in the description of EDON-$\mathcal{R}$, we rewrite then with Q_i, R_i and P_i;
- P_0 and P_1 are linear over $\mathbb{Z}_{2^w}^8$;
- R_0 and R_1 are component-wise rotations of w-bit words;
- Q_0 and Q_1 are linear over $(\mathbb{F}_2^w)^8$;
- We identify $\mathbb{Z}_{2^w}^8$ and $(\mathbb{F}_2^w)^8$ with the natural mapping.

between them.

Note that quasi-group operation is very easy to invert.

The compression function takes as input $16w$ bits of message ($M_0^{(i)}$ and $M_1^{(i)}$) and $16w$ bits of chaining value ($H_0^{(i)}$ and $H_1^{(i)}$) and produces $16w$ of new chaining value ($H_0^{(i+1)}$ and $H_1^{(i+1)}$). The full compression function is described in Figure 1, where the symbol $\odot$ denotes different quasi-group operations based on $*$, with some permutations of the inputs. For more details, see [2].

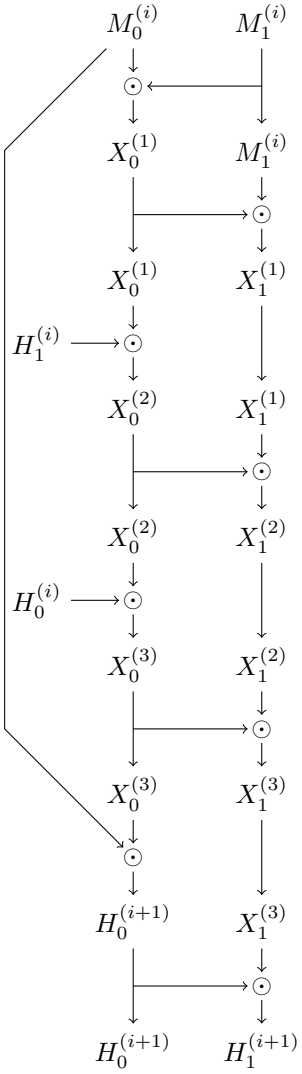


Fig. 1. EDON-$\mathcal{R}$ compression function.

2 Previous analysis of EDON-$\mathcal{R}$

Previous work [3,4] have shown various weaknesses of the compression function:

- given $M_0^{(i)}$, $M_1^{(i)}$, $H_0^{(i+1)}$ and $H_1^{(i+1)}$, it is easy to compute $H_0^{(i)}$ and $H_1^{(i)}$;
- given $H_0^{(i)}$, $H_1^{(i)}$, $M_0^{(i)}$, and $H_0^{(i+1)}$, it is easy to compute $M_1^{(i)}$, and $H_1^{(i+1)}$;
- given $H_1^{(i+1)}$, $H_0^{(i)}$ and $M_0^{(i)}$, we can find a value of $H_1^{(i)}$, $H_0^{(i+1)}$, and $M_1^{(i)}$ with $2^{n/2}$ operations.

These results can be used to mount various attacks on the hash function:

- We can apply generic attacks against single-pipe hash functions: multi-collisions, second preimages on long message, fixed points, ...

– There is a preimage attack with complexity $2^{2n/3}$ and $2^{2n/3}$ memory.

The preimage attack requires less computations than a generic attack, but the machine to carry out this attack might be more expensive than a machine to perform a parallel brute force, so it is unclear whether this should be considered as an attack.

However, these results show that the compression function of EDON-$\mathcal{R}$ is quite weak, and the security of EDON-$\mathcal{R}$ can't be based on a security proof of the Merkle-Damgård mode.

3 Secret-prefix MAC

We assume that EDON-$\mathcal{R}$ is used as a MAC with the secret-prefix construction: $\text{MAC}_k(M) = \text{EDON-}\mathcal{R}(k\|M)$. This kind of construction is used in some old protocols, like RFC2069 [1]. We assume that the key is padded to a full block, so that this construction is equivalent to using a secret IV $(H_0^{(0)}, H_1^{(0)})$, and our attack will recover this secret IV. Note that this is not the case in RFC2069, and we cannot use our key-recovery attack in this situation.

It is well known that this construction is weak, because length extension attacks can be used for forgeries, but the key is not expected to leak. Moreover, EDON-$\mathcal{R}$ is a double-pipe design, so the length extension issue does not apply. In fact, this construction is secure if the hash function is double-pipe and the compression function is modeled as a random oracle.

4 Our Results

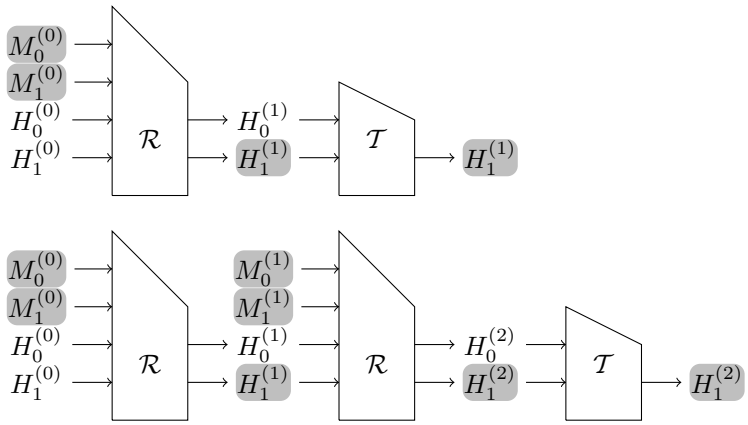
Our work shows that

– given $M_0^{(i)}$, $M_1^{(i)}$, $H_1^{(i)}$ and $H_1^{(i+1)}$, we can compute $H_0^{(i)}$ and $H_0^{(i+1)}$ with $2^{5n/8}$ operations.

This can be used to recover the key if EDON-$\mathcal{R}$ is used as a MAC with the secret-prefix construction. Our attack makes only two queries and needs $2^{5n/8}$ computations with negligible memory. It can easily be parallelized. This attack is the first attack on EDON-$\mathcal{R}$ to clearly beat parallel generic attacks. Our attack uses two related messages M and M' such that $\text{pad}(M)$ is a prefix of $\text{pad}(M')$ and $\text{pad}(M')$ is one block longer than $\text{pad}(M)$.

5 Key Recovery

We will make two calls to the MAC, with two related messages, such that after the padding step, the second message is a suffix of the first one.



Then, we target the second compression function, and we have:

- $M_0^{(1)}$, $M_1^{(1)}$ are known;
- $H_1^{(1)}$ is known;
- $H_1^{(2)}$ is known.

We will show how to recover $H_0^{(1)}$. Then $H_0^{(0)}$ and $H_1^{(0)}$ can easily be recovered from $H_0^{(1)}, H_1^{(1)}$ and $M_0^{(0)}, M_1^{(0)}$. Since there are $8w$ unknown bits in the input of the compression function ($H_0^{(i)}$) and we know $8w$ bits of the output of the compression function ($H_1^{(i+1)}$), we expect one solution on average. In this setting, a preimage attack will be able to recover *the* value of $H_0^{(i)}$ and not merely *a* value that gives the same output.

If we look at the description of the compression function [2], we have:

$$\begin{aligned} H_1^{(i+1)} &= H_0^{i+1} * X_1^3 \\ &= (\overline{M_0^{(i)}} * X_0^{(3)}) * (X_1^{(2)} * X_0^{(3)}) \\ &= (\pi_1(\pi_2(\overline{M_0^{(i)}})) + \pi_1(\pi_3(X_0^{(3)}))) * (\pi_1(\pi_2(X_1^{(2)})) + \pi_1(\pi_3(X_0^{(3)})) \\ &= (U + K_0) * (U + K_1) \end{aligned}$$

where $U = \pi_1(\pi_3(X_0^{(3)}))$ is unknown, and $K_0 = \pi_1(\pi_2(\overline{M_0^{(i)}}))$, $K_1 = \pi_1(\pi_2(X_1^{(2)}))$ are known constants.

If we are able to solve the equation $H = (U + K_0) * (U + K_1)$ where U is the unknown, then we can recover $X_0^{(3)}$ from U, and this will give us $H_0^{(1)}$.

6 Solving the equation $H = (U + K_0) * (U + K_1)$

The main step of the attack is to solve the equation

$$\begin{aligned} H &= (U + K_0) * (U + K_1) \\ &= Q_0(R_0(P_0(U + K_0))) + Q_1(R_1(P_1(U + K_1))) \end{aligned}$$

More precisely, P_0, P_1 are defined by the following matrices over $\mathbb{Z}_{2^{32}}$ (*i.e* the sums are modular additions):

$$P_0 = \begin{bmatrix} 1&1&1&0&1&0&0&1 \\ 1&1&0&1&1&0&0&1 \\ 1&1&0&0&1&0&1&1 \\ 0&0&1&1&0&1&1&1 \\ 0&1&1&1&0&1&1&0 \\ 1&0&1&1&1&1&0&0 \\ 1&1&0&0&0&1&1&1 \\ 0&0&1&1&1&1&1&0 \end{bmatrix} \qquad P_1 = \begin{bmatrix} 1&1&1&0&0&1&0&1 \\ 1&1&0&1&1&0&1&0 \\ 1&1&1&1&0&1&0&0 \\ 0&0&1&1&1&0&1&1 \\ 1&1&0&1&1&1&0&0 \\ 0&0&1&0&1&1&1&1 \\ 0&1&1&0&0&1&1&1 \\ 1&0&0&1&1&0&1&1 \end{bmatrix}$$

We will use three vectors U_0, U_1, U_2 in the kernels of some submatrices of P_0 and P_1:

$$\begin{aligned} U_0 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \\ U_1 &= \begin{bmatrix} 2 & 2 & 2 & 2 & 2^{31}-3 & 2^{31}-3 & 0 & 2^{31}-1 \end{bmatrix} \\ U_2 &= \begin{bmatrix} 1 & 0 & 0 & 0 & 2^{31}-1 & 2^{31} & 0 & 2^{31} \end{bmatrix} \end{aligned}$$

Then we have (the stars represent any non-zero value):

$$P_0 \cdot U_0 = \begin{bmatrix} * & * & 0 & 0 & * & 0 & 0 & * \end{bmatrix} \qquad P_1 \cdot U_0 = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$P_0 \cdot U_1 = \begin{bmatrix} * & * & 0 & 0 & * & 0 & 0 & * \end{bmatrix} \qquad P_1 \cdot U_1 = \begin{bmatrix} * & * & * & 0 & 0 & * & 0 & 0 \end{bmatrix}$$
$$P_0 \cdot U_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & * & 0 & * & * \end{bmatrix} \qquad P_1 \cdot U_2 = \begin{bmatrix} * & * & * & * & 0 & * & 0 & 0 \end{bmatrix}$$

Q_0, Q_1 are defined by the following matrices over $\mathbb{F}_2^8$ (*i.e* the sums are exclusive or):

$$Q_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Due to the positions of the zeros in $P_i \cdot U_j$, we have, for all $\alpha, \beta \in \mathbb{Z}_{2^{32}}$:

$$Q_0(R_0(P_0(X + \alpha U_0))) \oplus Q_0(R_0(P_0(X))) = \begin{bmatrix} * & * & * & * & * & 0 & 0 & 0 \end{bmatrix}$$
$$Q_0(R_0(P_0(X + \alpha U_1))) \oplus Q_0(R_0(P_0(X))) = \begin{bmatrix} * & * & * & * & * & 0 & 0 & 0 \end{bmatrix}$$
$$Q_0(R_0(P_0(X + \alpha U_2))) \oplus Q_0(R_0(P_0(X))) = \begin{bmatrix} * & * & * & * & * & * & * & 0 \end{bmatrix}$$
$$Q_1(R_1(P_1(Y + \beta U_0))) \oplus Q_1(R_1(P_1(Y))) = \begin{bmatrix} * & * & * & * & * & 0 & 0 & 0 \end{bmatrix}$$
$$Q_1(R_1(P_1(Y + \beta U_1))) \oplus Q_1(R_1(P_1(Y))) = \begin{bmatrix} * & * & * & * & * & 0 & * & 0 \end{bmatrix}$$
$$Q_1(R_1(P_1(Y + \beta U_2))) \oplus Q_1(R_1(P_1(Y))) = \begin{bmatrix} * & * & * & * & * & * & * & 0 \end{bmatrix}$$

This proves that the vectors U_0, U_1, U_2 do not affect some of the output words:

$$(X + \alpha U_0) * (Y + \beta U_0) \oplus X * Y = \begin{bmatrix} * & * & * & * & * & 0 & 0 & 0 \end{bmatrix} \quad (1)$$
$$(X + \alpha U_1) * (Y + \beta U_1) \oplus X * Y = \begin{bmatrix} * & * & * & * & * & 0 & * & 0 \end{bmatrix} \quad (2)$$
$$(X + \alpha U_2) * (Y + \beta U_2) \oplus X * Y = \begin{bmatrix} * & * & * & * & * & * & * & 0 \end{bmatrix} \quad (3)$$

This is a very important part of the attack, so let us explain into more detail what equation (3) means. Using notations similar to the one from [2], the last output word of $X * Y$ is computed as:

$$(X * Y)^{[7]} = (T_X^{[2]} \oplus T_X^{[3]} \oplus T_X^{[5]}) + (T_Y^{[4]} \oplus T_Y^{[6]} \oplus T_Y^{[7]})$$

where

$$T_X^{[2]} = (X^{[0]} + X^{[1]} + X^{[4]} + X^{[6]} + X^{[7]}) \lll 8$$
$$T_X^{[3]} = (X^{[2]} + X^{[3]} + X^{[5]} + X^{[6]} + X^{[7]}) \lll 13$$
$$T_X^{[5]} = (X^{[0]} + X^{[2]} + X^{[3]} + X^{[4]} + X^{[5]}) \lll 22$$
$$T_Y^{[4]} = (Y^{[0]} + Y^{[1]} + Y^{[3]} + Y^{[4]} + Y^{[5]}) \lll 15$$
$$T_Y^{[6]} = (Y^{[1]} + Y^{[2]} + Y^{[5]} + Y^{[6]} + Y^{[7]}) \lll 25$$
$$T_Y^{[7]} = (Y^{[0]} + Y^{[3]} + Y^{[4]} + Y^{[6]} + Y^{[7]}) \lll 27$$

We now consider $X' = X + \alpha U_2$ and $Y' = Y + \beta U_2$:

$$(X' * Y')^{[7]} = (T'^{[2]}_X \oplus T'^{[3]}_X \oplus T'^{[5]}_X) + (T'^{[4]}_Y \oplus T'^{[6]}_Y \oplus T'^{[7]}_Y)$$

where

$$\begin{aligned}
T'^{[2]}_X &= (X^{[0]} + \alpha + X^{[1]} + X^{[4]} + \alpha(2^{31} - 1) + X^{[6]} + X^{[7]} + \alpha 2^{31}) \lll 8 \\
T'^{[3]}_X &= (X^{[2]} + X^{[3]} + X^{[5]} + \alpha 2^{31} + X^{[6]} + X^{[7]} + \alpha 2^{31}) \lll 13 \\
T'^{[5]}_X &= (X^{[0]} + \alpha + X^{[2]} + X^{[3]} + X^{[4]} + \alpha(2^{31} - 1) + X^{[5]} + \alpha 2^{31}) \lll 22 \\
T'^{[4]}_Y &= (Y^{[0]} + \beta + Y^{[1]} + Y^{[3]} + Y^{[4]} + \beta(2^{31} - 1) + Y^{[5]} + \beta 2^{31}) \lll 15 \\
T'^{[6]}_Y &= (Y^{[1]} + Y^{[2]} + Y^{[5]} + \beta 2^{31} + Y^{[6]} + Y^{[7]} + \beta 2^{31}) \lll 25 \\
T'^{[7]}_Y &= (Y^{[0]} + \beta + Y^{[3]} + Y^{[4]} + \beta(2^{31} - 1) + Y^{[6]} + Y^{[7]} + \beta 2^{31}) \lll 27
\end{aligned}$$

We see that the α and β terms cancels out:

$$\begin{aligned}
T^{[2]}_X &= T'^{[2]}_X & T^{[3]}_X &= T'^{[3]}_X & T^{[5]}_X &= T'^{[5]}_X \\
T^{[4]}_Y &= T'^{[4]}_Y & T^{[6]}_Y &= T'^{[6]}_Y & T^{[7]}_Y &= T'^{[7]}_Y
\end{aligned}$$

and as a consequence

$$(X' * Y')^{[7]} = (X * Y)^{[7]}$$

This works because U_2 was chosen in the kernel of the linear forms that define $T^{[2]}_X$, $T^{[3]}_X$, $T^{[5]}_X$, $T^{[4]}_Y$, $T^{[6]}_Y$, and $T^{[7]}_Y$. Similarly, U_1 is in the kernel of the linear forms involved in the computation of $(X*Y)^{[5,7]}$ and U_0 is in the kernel of the linear forms involved in the computation of $(X*Y)^{[5,6,7]}$.

Thanks to this property, we can do an exhaustive search with early abort. We extend U_0, U_1, U_2 into a basis $U_0, U_1, ...U_7$ of $\mathbb{Z}^8_{2^{32}}$, and we will represent U in this basis: $U = \sum_{i=0}^{7} \alpha_i U_i$. We define $V = (U + K_0) * (U + K_1)$. Due to the properties of U_0, U_1, U_2, we know that:

- α_0 has no effect on $V^{[5]}$, $V^{[6]}$ and $V^{[7]}$;
- α_1 has no effect on $V^{[5]}$ and $V^{[7]}$;
- α_2 has no effect on $V^{[7]}$.

The full algorithm is given by Algorithm 1 and is quite simple. We first iterate over $\alpha_3, \alpha_4, ...\alpha_7$ and we filter the elements such that $V = (U + K_0) * (U + K_1)$ matches H on the last coordinates. If it does not match, we don't need to iterate over $\alpha_0, \alpha_1, \alpha_2$ because this wont modify $V^{[7]}$, so we can abort this branch. For the choices that match, we iterate over α_2 and check $V^{[5]}$. If it matches $H^{[5]}$, we iterate over α_1 and check $V^{[6]}$. If it matches $H^{[5]}$, we can then iterate over α_0.

The time complexity is $2^{5w} = 2^{5n/8}$:

- the first loop is executed 2^{5w} times;
- each matching reduces the number of candidates to 2^{4w};
- each subsequent loop raises the number of candidates to 2^{5w}.

The memory requirement are negligible because we do not need to store a list a candidate, we just iterate over a set and filter out the candidates as they come.

Once we have recovered $U = \pi_1(\pi_3(X^{(3)}_0))$, it is easy to invert the permutations and recover $X^{(3)}_0$. From that we find $H^{(i)}_0$ by inverting a quasi-group operation, and we have all the variables of the compression function. We can then recover the key $H^{(0)}_0, H^{(0)}_1$ by inverting the first compression function (it is easy when the output and the message are known)

Algorithm 1 Solving $H = (U + K_0) * (U + K_1)$

Input: K_0, K_1, H
Output: U

for all $\alpha_3, \alpha_4, ...\alpha_7 \in \mathbb{Z}_{2^{32}}$ **do**
  $U \leftarrow \sum_{i=3}^{7} \alpha_i U_i$
  $V \leftarrow (U + K_0) * (U + K_1)$
  if $V^{[7]} = H^{[7]}$ **then**
    for all $\alpha_2 \in \mathbb{Z}_{2^{32}}$ **do**
      $U \leftarrow \sum_{i=2}^{7} \alpha_i U_i$
      $V \leftarrow (U + K_0) * (U + K_1)$
      if $V^{[5]} = H^{[5]}$ **then**
        for all $\alpha_1 \in \mathbb{Z}_{2^{32}}$ **do**
          $U \leftarrow \sum_{i=1}^{7} \alpha_i U_i$
          $V \leftarrow (U + K_0) * (U + K_1)$
          if $V^{[6]} = H^{[6]}$ **then**
            for all $\alpha_0 \in \mathbb{Z}_{2^{32}}$ **do**
              $U \leftarrow \sum_{i=0}^{7} \alpha_i U_i$
              $V \leftarrow (U + K_0) * (U + K_1)$
              if $V = H$ **then**
                U is a solution

References

1. Franks, J. and Hallam-Baker, P. and Hostetler, J. and Leach, P. and Luotonen, A. and Sink, E. and Stewart, L.: RFC2069: An extension to HTTP: Digest access authentication. Internet RFCs (1997)
2. Gligoroski, D., Ødegård, R.S., Mihova, M., Knapskog, S.J., Kocarev, L., Drápal, A., Klima, V.: Cryptographic Hash Function EDON-R. Submission to NIST (2008)
3. Khovratovich, D., Nikolić, I., Weinmann, R.P.: Cryptanalysis of Edon-R. Available online (2008)
4. Klima, V.: Multicollisions of EDON-R hash function and other observations. Available online (2008)