Practical 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 one of the fastest SHA-3 candidate. In this paper we study the security of EDON-$\mathcal{R}$, and we show that using EDON-$\mathcal{R}$ as a MAC with the secret prefix construction is unsafe. We present a practical attack in the case of EDON-$\mathcal{R}$256, which requires 32 queries, 2^{30} computations, negligible memory, and a precomputation of 2^{50}. This does not directly contradict the security claims of EDON-$\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 Introduction

In 2005, a team of researchers led by X. Wang produced breakthrough attacks against many widely used hash functions, including MD5 [9] and SHA-1 [8]. This has led NIST to call for a new hash function design, and to launch the SHA-3 competition. This competition has focused the attention of many cryptographers, and NIST received 64 submissions. 51 designs were accepted to the first round, and 10 out of those 51 has been conceded broken by their designers so far.

EDON-$\mathcal{R}$ is one of the fastest candidates in this competition. It has already received some attention from the cryptographic community, resulting in various attacks on the compression function. There is also a preimage attack on the full hash function, but it requires of huge amount of memory making it debatable.

In this paper we show a new attack on EDON-$\mathcal{R}$, when used in the secret-prefix MAC construction. This mode of operation is not claimed to be secure by the designers, but our attack has no memory requirement, making it somewhat less debatable than previous attacks. Our approach is similar to the one followed by Wang *et al.* who studied a similar MAC used with SHA-1 [7]: we use a non-standard MAC to show weaknesses of the hash function. Note that attacks on hash-based MACs are usually harder to build than attacks on the hash function itself because part of the state is unknown.

1.1 Secret-prefix MAC

We assume that EDON-$\mathcal{R}$ is used as a MAC with the secret-prefix construction, defined as $\mathrm{MAC}_k(M) = \text{EDON-}\mathcal{R}(k\|M)$. This kind of construction is used in some old protocols, like RFC2069 [2]. 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 wipe-pipe design, so the length extension issue does not apply. In fact, this construction is secure if the hash function is wipe-pipe and the compression function is modeled as a random oracle [1].

In the following we assume that the key is padded to a full block, so that the secret-prefix construction is equivalent to using a secret IV $(H_{0,0}, H_{0,1})$, and our attack will recover this secret IV.

However, some constructions (eg. RFC2069) do not pad the key to a full block. In this case, we can still use our attack to get the MAC of an arbitrary message in an adaptive way. Given a challenge message M, we view $\text{pad}(k\|M)$ as the key, and we apply our attack to recover EDON-$\mathcal{R}(k\|M)$. In this case, for each challenge, we need to make some queries to the MAC oracle.

1.2 Road Map

Section 2 will describe EDON-$\mathcal{R}$ and discuss previous analysis. Then, in section 3, we show how to use a pair a related queries to gather information on both the input and the output of the compression function. The idea is similar to the length extension attack against Merkle-Damgård hash functions. This reduces the key-recovery problem to solving a small equation. In section 1, we show how to solve this equation. We use simple linear algebra techniques to identify truncated differentials in the main operations of EDON-$\mathcal{R}$, and this leads to an attack with complexity $2^{5n/8}$ using only two queries to the MAC oracle. In section 5 we use more queries to the MAC oracle to build more equations, and solve the equations using a guess-and-determine technique. This gives a very efficient attack, which is even practical in the case of EDON-$\mathcal{R}$224/256.

2 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 $(\mathbb{F}_2^w)^8$. The quasi-group operation is just the sum of two permutations, and we will use a permutation based description of EDON-$\mathcal{R}$ in this paper:

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

where

- $+$ is a component-wise addition modulo 2^w (w is the word size);
- μ and ν are the permutations defining $*$; we rewrite then with Q_i, R_i and P_i;
- P_0 and P_1 are linear over $\mathbb{Z}_{2^w}^8$, each output word is the sum of five inputs;
- R_0 and R_1 are component-wise rotations of w-bit words;
- Q_0 and Q_1 are linear over $(\mathbb{F}_2^w)^8$, each output word is the xor of three inputs;

– We identify $\mathbb{Z}_{2^w}^8$ and $(\mathbb{F}_2^w)^8$ with the natural mapping between them;
– We also define $\bar{\mu}(X^{[0]}, X^{[1]}, ...X^{[7]}) = \mu(X^{[7]}, X^{[6]}, ...X^{[0]})$.

Note that the quasi-group operation is very easy to invert: given X and $X * Y$, we can compute Y as $\nu^{-1}(X * Y - \mu(X))$.

The compression function takes as input 16 message ($M_{i,0}$ and $M_{i,1}$) words and 16 words of chaining value ($H_{i,0}$ and $H_{i,1}$) and produces 16 words of new chaining value ($H_{i+1,0}$ and $H_{i+1,1}$). The full compression function is described in Figure 1. For more details, see [3].

2.1 Previous analysis of Edon-$\mathcal{R}$

Previous work [4,5] have shown various weaknesses of the compression function:

– given $M_{i,0}$, $M_{i,1}$, $H_{i+1,0}$ and $H_{i+1,1}$, it is easy to compute $H_{i,0}$ and $H_{i,1}$;
– given $H_{i,0}$, $H_{i,1}$, $M_{i,0}$, and $H_{i+1,0}$, it is easy to compute $M_{i,1}$, and $H_{i+1,1}$;
– given $H_{i+1,1}$, $H_{i,0}$ and $M_{i,0}$, we can find a value of $H_{i,1}$, $H_{i+1,0}$, and $M_{i,1}$ with $2^{n/2}$ operations.

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

– We can apply generic attacks against narrow-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 due to the large memory requirement, 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.

2.2 Our Results

Our work shows that

– given $M_{i,0}$, $M_{i,1}$, $H_{i,1}$ and $H_{i+1,1}$, we can compute $H_{i,0}$ and $H_{i+1,0}$ with $2^{5n/8}$ operations.
– given $M_{i,0}^{(j)}$, $M_{i,1}^{(j)}$, $H_{i,1}$ and $H_{i+1,1}^{(j)}$ for a group of 32 carefully chosen related messages ($H_{i,0}$ and $H_{i,1}$ are the same for all messages), we can compute $H_{i,0}$ with 2^{30} operations for Edon-$\mathcal{R}$256, or 2^{32} for Edon-$\mathcal{R}$512. However, there is a precomputation step to build the group of message which costs 2^{50} for Edon-$\mathcal{R}$256, and 2^{98} for Edon-$\mathcal{R}$512.

This can be used to recover the key if Edon-$\mathcal{R}$ is used as a MAC with the secret-prefix construction. Our attacks only needs a few queries and negligible memory. They can

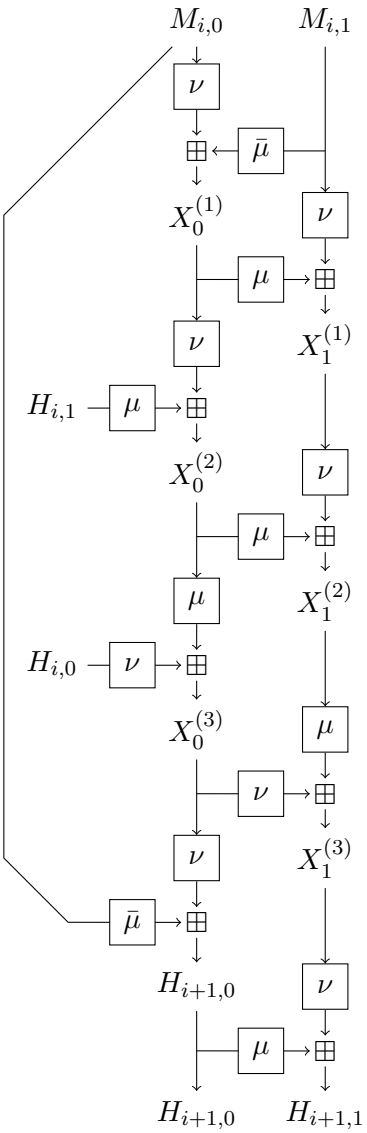

Fig. 1. EDON-$\mathcal{R}$ compression function.

easily be parallelized. Those attacks are the first attacks on EDON-$\mathcal{R}$ to clearly beat parallel generic attacks.

In this paper we will describe two attack: a first one that requires only two queries, and a second with more queries but the complexity will be practical:

	Queries	Time	Memory	Precomputation
Edon-$\mathcal{R}$224/256	2	2^{160}	-	-
Edon-$\mathcal{R}$224/256	32	$\simeq 2^{30}$	-	2^{50}
Edon-$\mathcal{R}$384/512	2	2^{320}	-	-
Edon-$\mathcal{R}$384/512	32	$\simeq 2^{32}$	-	2^{98}

3 Key Recovery Using Related Queries

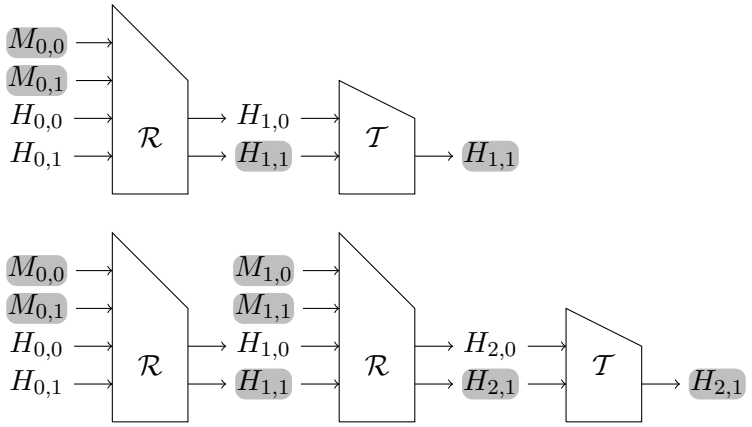

Fig. 2. The first message $\text{pad}(M) = M_{0,0}M_{0,1}$ allow to recover $H_{1,1}$ while the second message $\text{pad}(M') = M_{0,0}M_{0,1}M_{1,0}M_{1,1}$ allows to recover $H_{2,1}$.

We will make two calls to the MAC, with two related messages, such that after the padding step, the first message is a prefix of the second one. The first message M is chosen arbitrarily such that after the padding it fits in one block $\text{pad}(M)$. The second message M' has $\text{pad}(M)$ as its first block, and has to fit in two blocks after the padding. For instance, we can use:

M	*(empty)*
$\text{pad}(M)$	80000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
M'	80000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$\text{pad}(M')$	80000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 80000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000200

This is similar to the length extension attack on narrow-pipe hash function. Applied to a wide-pipe design such as Edon-$\mathcal{R}$, this gives us some information on the input and output of the second compression function (see Fig 2):

– $M_{1,0}$ and $M_{1,1}$ are known;
– $H_{1,1}$ is known;
– $H_{2,1}$ is known.

We will show how to recover $H_{1,0}$. Then $H_{0,0}$ and $H_{0,1}$ can be recovered from $H_{1,0}, H_{1,1}$ and $M_{0,0}, M_{0,1}$ because the compression function of Edon-$\mathcal{R}$ is easy to invert [4]. Since there are 8 unknown words in the input of the compression function ($H_{1,0}$) and we know 8 words of the output of the compression function ($H_{2,1}$), we expect one solution on average. In this setting, a preimage attack will be able to recover *the* value of $H_{1,0}$ and not merely *a* value that gives the same output.

If we look at the description of the compression function [3], we have:

$$\begin{aligned} H_{2,1} &= H_{2,0} * X_1^{(3)} \\ &= (\overline{M_{1,0}} * X_0^{(3)}) * (X_1^{(2)} * X_0^{(3)}) \\ &= (\bar{\mu}(M_{1,0}) + \nu(X_0^{(3)})) * (\mu(X_1^{(2)}) + \nu(X_0^{(3)})) \\ &= (U + C_0) * (U + C_1) \end{aligned}$$

where $U = \nu(X_0^{(3)})$ is unknown, and $C_0 = \bar{\mu}(M_{1,0})$, $C_1 = \mu(X_1^{(2)})$ are known constants.

If we are able to solve the equation $H = (U + C_0) * (U + C_1)$ where U is the unknown, then we can recover $X_0^{(3)} = \nu^{-1}(U)$, and this will give us $H_{1,0} = \nu^{-1}(X_0^{(3)} - \mu(X_0^{(2)}))$.

4 Solving the equation $H = (U + C_0) * (U + C_1)$

The main step of the attack is to solve the equation

$$\begin{aligned} H &= (U + C_0) * (U + C_1) \\ &= Q_0(R_0(P_0(U + C_0))) + Q_1(R_1(P_1(U + C_1))) \end{aligned}$$

All the variables are 8-uples of w bit words, and U is the unknown. To solve this equation, we will express U over a basis of $\mathbb{Z}_{2^w}^8$ such that some basis vector do not affect some words of $(U + C_0) * (U + C_1)$. Then we can solve the equation more efficiently than by brute force because we do not need to explore the full space.

More precisely, P_0, P_1 are defined by the following matrices over $\mathbb{Z}_{2^w}$ (*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):

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

Q_0, Q_1 are defined by the following matrices over $\mathbb{F}_2^w$ (*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 $\alpha, \beta \in \mathbb{Z}_{2^w}$:

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

This proves that the vectors U_0, U_1, U_2 do not affect some of the output words. This property can be seen as a truncated differential for the $*$ operation:

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

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

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

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 [3], 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

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

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:

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

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 $T_X^{[2]}$, $T_X^{[3]}$, $T_X^{[5]}$, $T_Y^{[4]}$, $T_Y^{[6]}$, and $T_Y^{[7]}$. 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 $\mathbb{Z}_{2^w}^8$, and we will represent U in this basis: $U = \sum_{i=0}^{7} \alpha_i U_i$. We define $V = (U + C_0) * (U + C_1)$. 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 $V = (U + C_0) * (U + C_1)$ 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 $H^{[6]}$, 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.

Algorithm 1 Solving $H = (U + C_0) * (U + C_1)$

Input: C_0, C_1, H
Output: U

for all $\alpha_3, \alpha_4, ...\alpha_7 \in \mathbb{Z}_{2^w}$ **do**
  $U \leftarrow \sum_{i=3}^{7} \alpha_i U_i$
  $V \leftarrow (U + C_0) * (U + C_1)$
  if $V^{[7]} = H^{[7]}$ **then**
    for all $\alpha_2 \in \mathbb{Z}_{2^w}$ **do**
      $U \leftarrow \sum_{i=2}^{7} \alpha_i U_i$
      $V \leftarrow (U + C_0) * (U + C_1)$
      if $V^{[5]} = H^{[5]}$ **then**
        for all $\alpha_1 \in \mathbb{Z}_{2^w}$ **do**
          $U \leftarrow \sum_{i=1}^{7} \alpha_i U_i$
          $V \leftarrow (U + C_0) * (U + C_1)$
          if $V^{[6]} = H^{[6]}$ **then**
            for all $\alpha_0 \in \mathbb{Z}_{2^w}$ **do**
              $U \leftarrow \sum_{i=0}^{7} \alpha_i U_i$
              $V \leftarrow (U + C_0) * (U + C_1)$
              if $V = H$ **then**
                U is a solution

Once we have recovered $U = \nu(X_0^{(3)})$, it is easy to invert the permutations and recover $X_0^{(3)}$. From that we find $H_{1,0}$ 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)

5 Using more queries

In this section, we improve this attack using more queries to the MAC oracle. We gather more equations of the form $H = (U + C_0) * (U + C_1)$, and this enables us to mount a

very efficient attack. In the case of Edon-$\mathcal{R}256$, it requires about 32 queries and can recover the secret key with about 2^{30} computations after a precomputation of about 2^{50} operations, which makes it a practical attack.

5.1 Building the queries

To get new equations, we will query the MAC oracle with new messages $M^{(i)}$ so that $\mathrm{pad}(M)$ is a prefix of all the $M^{(i)}$'s. Each query will give some equation involving the same $H_{1,0}$, and we will deduce an equation of the form $H^{(i)} = (U + C_0^{(i)}) * (U + C_1^{(i)})$ as in the previous section. Remember that we have $U = \nu(X_0^{(3)}) = \nu(\nu(H_{1,0}) + \mu(X_0^{(2)}))$. We will build our messages so that the value of $X_0^{(2)}$ is the same for all the $M^{(i)}$'s, or equivalently, $X_0^{(1)}$ is the same for all the $M^{(i)}$'s. This means that all the equations will involve the same U, and recovering this U will allow to recover $H_{1,0}$.

Let us assume that we have two such equations, with $C_0^{(i)} = C_0^{(j)}$.

$$\begin{aligned} H^{(i)} &= Q_0(R_0(P_0(U + C_0^{(i)}))) + Q_1(R_1(P_1(U + C_1^{(i)}))) \\ H^{(j)} &= Q_0(R_0(P_0(U + C_0^{(j)}))) + Q_1(R_1(P_1(U + C_1^{(j)}))) \\ H^{(i)} - H^{(j)} &= Q_1(R_1(P_1(U + C_1^{(i)}))) - Q_1(R_1(P_1(U + C_1^{(j)}))) \end{aligned}$$

since P_1 is linear over $\mathbb{Z}_{2^w}^8$, we can consider $\tilde{U} = P_1 \cdot U$ and $\tilde{C}_1^{(i)} = P_1 \cdot C_1^{(i)}$

$$H^{(i,j)} = Q_1(R_1(\tilde{U} + \tilde{C}_1^{(i)})) - Q_1(R_1(\tilde{U} + \tilde{C}_1^{(j)})) \tag{4}$$

If we consider $\tilde{U} = P_1 \cdot U$ to be the unknown, this gives a simpler equation than in the previous section, where $H^{(i,j)} = H^{(i)} - H^{(j)}$, $\tilde{C}_1^{(i)}$ and $\tilde{C}_1^{(j)}$ are known constants.

However, if we have a pair of messages $M^{(i)}, M^{(j)}$ where $C_0^{(i)} = C_0^{(j)}$ and $X_0^{(1)}$ is constant, then we have $M^{(i)} = M^{(j)}$ and we don't get any equation. Instead, we use messages such that $C_0^{(i)}$ and $C_0^{(j)}$ have some relations. Namely, if we have

$$(P_0 \cdot C_0^{(i)})^{[2]} = (P_0 \cdot C_0^{(j)})^{[2]} \quad (P_0 \cdot C_0^{(i)})^{[3]} = (P_0 \cdot C_0^{(j)})^{[3]} \quad (P_0 \cdot C_0^{(i)})^{[5]} = (P_0 \cdot C_0^{(j)})^{[5]} \tag{5}$$

then

$$\begin{aligned} H^{(i,j)[7]} &= \Big(\mu(U + C_0^{(i)}) + \nu(U + C_1^{(i)})\Big)^{[7]} - \Big(\mu(U + C_0^{(j)}) + \nu(U + C_1^{(j)})\Big)^{[7]} \\ &= \Big(\nu(U + C_1^{(i)}) - \nu(U + C_1^{(j)})\Big)^{[7]} + \Big(\mu(U + C_0^{(i)}) - \mu(U + C_0^{(j)})\Big)^{[7]} \\ &= \Big(\nu(U + C_1^{(i)}) - \nu(U + C_1^{(j)})\Big)^{[7]} \\ &+ \Big(P_0(U + C_0^{(i)})^{[2]} \ggg 8 \oplus P_0(U + C_0^{(i)})^{[3]} \ggg 13 \oplus P_0(U + C_0^{(i)})^{[5]} \ggg 22\Big) \\ &- \Big(P_0(U + C_0^{(j)})^{[2]} \ggg 8 \oplus P_0(U + C_0^{(j)})^{[3]} \ggg 13 \oplus P_0(U + C_0^{(j)})^{[5]} \ggg 22\Big) \end{aligned}$$

The two last terms cancel out by linearity of P_0 over $\mathbb{Z}_{2^w}^8$

$$\begin{aligned} H^{(i,j)[7]} &= \left(\nu(U + C_1^{(i)}) - \nu(U + C_1^{(j)})\right)^{[7]} \\ &= Q_1(R_1(\tilde{U} + \tilde{C}_1^{(i)}))^{[7]} - Q_1(R_1(\tilde{U} + \tilde{C}_1^{(j)}))^{[7]} \end{aligned} \tag{6}$$

We can see (6) as a weaker version of (4): we only have an equation on one word, instead of eight. We can build similar equations restricted to any word by choosing appropriate relations between $C_0^{(i)}$ and $C_0^{(j)}$: if we want an equation restricted to word k we just need to have an equality between $P_0 \cdot C_0^{(i)}$ and $P_0 \cdot C_0^{(j)}$ on the three words used in the computation of $Q_0^{[k]}$.

5.2 Dealing with the padding

Another problem that we face to gather these equations is the padding. Edon-$\mathcal{R}$ uses a padding with Merkle-Damgård strengthening, so there are 65 bits in $M_{1,1}$ that must be kept untouched (129 bits in Edon-$\mathcal{R}$384/512).

To find proper messages, we use a preprocessing step. First, we fix some arbitrary value for $X_0^{(1)}$. Then we take a set of random $M_{1,1}$ satisfying the padding, we compute the corresponding $M_{1,0}$ and we look for a collision in three words of $P_0 \cdot C_0^{(i)}$ according to (5). Each collision costs 2^{48} computations on average (2^{96} for Edon-$\mathcal{R}$384/512), and gives one equation. Note that this is independent of the key we are attacking. It can be done as a preprocessing step, and we only need to store a the message pairs that will be used to extract the equations. Since we need 16 collisions, the time complexity of this preprocessing step will be $\sqrt{16} \times 2^{48}$ for Edon-$\mathcal{R}$256 and $\sqrt{16} \times 2^{96}$ for Edon-$\mathcal{R}$512 [6].

5.3 Solving

To recover the value of U, we gather some equation of the type of (6). We can rewrite them as:

$$\begin{aligned} &\left((\tilde{U}^{[4]} + \tilde{C}_1^{(i)[4]}) \ggg 17 \oplus (\tilde{U}^{[6]} + \tilde{C}_1^{(i)[6]}) \ggg 7 \oplus (\tilde{U}^{[7]} + \tilde{C}_1^{(i)[7]}) \ggg 5\right) - \\ &\left((\tilde{U}^{[4]} + \tilde{C}_1^{(j)[4]}) \ggg 17 \oplus (\tilde{U}^{[6]} + \tilde{C}_1^{(j)[6]}) \ggg 7 \oplus (\tilde{U}^{[7]} + \tilde{C}_1^{(j)[7]}) \ggg 5\right) = H^{(i,j)[7]} \end{aligned} \tag{7}$$

We will solve these equations using a guess-and-determine approach. First we guess the 18 lower bits of $\tilde{U}^{[4]}$, the 8 lower bits of $\tilde{U}^{[6]}$, and the 6 lower bits of $\tilde{U}^{[7]}$. This allows us to compute the least significant bit of the left hand side of (7), and we check this bit against the right hand side. If we have enough equations, we can filter out many candidates. Then we guess one more bit of $\tilde{U}^{[4]}$, $\tilde{U}^{[6]}$, and $\tilde{U}^{[7]}$. We can now compute one more bit of (7), and again reduce the number of candidates. We repeat this step until all the bits of $\tilde{U}^{[4]}$, $\tilde{U}^{[6]}$, and $\tilde{U}^{[7]}$ have been guessed. Each time we guess some bits, the number of candidates grows, but it will shrink when we check the new bit of (7). The cost of this step is at least 2^{32} because we have to guess 32 bits in the beginning. If we have

enough equations and they give an independent filtering, we expect the complexity to be about 2^{32}. We did some experiments with random constants to check our assumptions. Experiments shows that with only 10 equations we can solve (7) for Edon-$\mathcal{R}$256 by exploring slightly more than 2^{32} nodes. This take a few minutes on a desktop PC. For Edon-$\mathcal{R}$512, we have to guess 56 bits, and we expect a complexity of 2^{56}.

Another way to solve this system is to guess the carries instead of guessing the low order bits. In this case, we only use 4 equations, because we have to guess the carries in each equations. We have only 24 carry bits to guess, but the 4 equations have many solutions, so we use extra equations to check each of these solutions until a single solution is left. According to our experiments, this takes about one minute on a desktop PC, and we have about 2^{16} solutions when using 4 equations (the search goes through 2^{30} nodes). Note that the complexity of this technique is independent of the rotation amounts, so it can be applied with any output word, not necessarily the seventh as in (6). More importantly, it is about as efficient on Edon-$\mathcal{R}$512: it take about 20 minutes to explore 2^{33} nodes, and gives about 2^{20} solutions.

This first step gives us $\tilde{U}^{[4]}$, $\tilde{U}^{[6]}$, and $\tilde{U}^{[7]}$. Next, we need an equation similar to (6), but involving the fifth word instead of the seventh:

$$\begin{aligned}&\Big((\tilde{U}^{[3]} + \tilde{C}_1^{(i)[3]}) \ggg 21 \oplus (\tilde{U}^{[4]} + \tilde{C}_1^{(i)[4]}) \ggg 17 \oplus (\tilde{U}^{[6]} + \tilde{C}_1^{(i)[6]}) \ggg 7\Big) - \\ &\Big((\tilde{U}^{[3]} + \tilde{C}_1^{(j)[3]}) \ggg 21 \oplus (\tilde{U}^{[4]} + \tilde{C}_1^{(j)[4]}) \ggg 17 \oplus (\tilde{U}^{[6]} + \tilde{C}_1^{(j)[6]}) \ggg 7\Big) = H^{(i,j)[5]}\end{aligned} \tag{8}$$

Since this equation only involves one unknown word $\tilde{U}^{[3]}$, it is quite easy to solve. We use the same technique as previously: we guess the carry bits. We only have 2 carry bits to guess so this step is negligible. We will repeat this using some more equations and we can recover the words of $\tilde{U}$ one by one. Then, we can recover $U = P_1^{-1} \cdot \tilde{U}$, and finally $H_{1,0}$.

The number of queries needed for the attack is

- 2×10 to recover three words in the first step;
- 2 for each extra word.

Conclusion

We have shown a practical key-recovery attack against secret-prefix Edon-$\mathcal{R}$. While this construction is not required to be secure by NIST, it is a natural construction that is used in some protocols. We believe that a strong cryptographic hash function should not leak the key when used in this setting.

References

1. Chang, D., Nandi, M.: Improved Indifferentiability Security Analysis of chopMD Hash Function. In Nyberg, K., ed.: FSE. Volume 5086 of Lecture Notes in Computer Science., Springer (2008) 429–443
2. 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)
3. 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)
4. Khovratovich, D., Nikolić, I., Weinmann, R.P.: Cryptanalysis of Edon-R. Available online (2008)
5. Klima, V.: Multicollisions of EDON-R hash function and other observations. Available online (2008)
6. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptology **12**(1) (1999) 1–28
7. Wang, X., Wang, W., Jia, K., Wang, M.: New Distinguishing Attack on MAC using Secret-Prefix Method. In Dunkelman, O., ed.: FSE. Lecture Notes in Computer Science, Springer (2009)
8. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
9. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 19–35