Cryptanalysis of the EMD Mode of Operation

Antoine Joux

DCSSI Crypto Lab
51, Bd de Latour-Maubourg
75700 PARIS 07 SP
FRANCE
antoine.joux@m4x.org

Abstract. In this paper, we study the security of the Encrypt-Mask-Decrypt mode of operation, also called EMD, which was recently proposed for applications such as disk-sector encryption. The EMD mode transforms an ordinary block cipher operating on n–bit blocks into a tweakable block cipher operating on large blocks of size nm bits. We first show that EMD is not a secure tweakable block cipher and then describe efficient attacks in the context of disk-sector encryption. We note that the parallelizable variant of EMD, called EME that was proposed at the same time is also subject to these attacks.
In the course of developing one of the attacks, we revisit Wagner's generalized birthday algorithm and show that in some special cases it performs much more efficiently than in the general case. Due to the large scope of applicability of this algorithm, even when restricted to these special cases, we believe that this result is of independent interest.

1 Introduction

Very recently, Rogaway proposed in [5] two new modes of operation that are specifically tailored for applications such as disk-sector encryption. The first mode called EMD (Encrypt-Mask-Decrypt) mostly consists in two consecutive passes of CBC encryption/decryption. Thus it is a sequential mode. The second mode called EME is a parallelizable version of EMD. In an updated version of [5], written with Halevi, the EMD algorithm is presented under the new name CMC. In particular, during the First IEEE International Security in Storage Workshop, held Dec. 11th, 2002 in Greenbelt, Maryland, the algorithm was presented under this new name.

In order to encrypt disk-sectors, several important properties are often required. First, when encrypting a single sector, a change in any single plaintext bit should impact the complete ciphertext sector. Since disk sectors are much larger than the block-size of usual block ciphers and since ordinary modes of operation such a CBC encryption do not ensure this property, this requirement calls for a specific construction. Second, for efficiency reasons, each sector should be encrypted independently of other sectors. As a consequence, an electronic code book at the sector level is expected. However, to avoid attacks based on sectors switching, it is highly desirable to use a (slightly) different encryption for each sector. This idea can be rigorously realized by using the notion of tweakable block ciphers which was recently proposed by Liskov, Rivest and Wagner in [4]. Informally, a tweakable block cipher is a block cipher which possesses an extra input called the tweak. It should resist distinguishing attacks mounted by powerful adversaries which are allowed access to encryption and decryption oracles and can choose both the plaintext or ciphertext messages and the tweak parameters.

In order to meet these specifications, Rogaway proposed the modes of operation EMD and EME in order to build a tweakable block cipher with a large blocksize of nm bits from an ordinary block cipher with a blocksize of n bits. In these constructions, m represents the number of blocks that can be fitted into a single disk sector. In [5], he also stated a security theorem for the EMD mode

In this paper, we show that neither the EMD nor the EME modes of operation are secure tweakable block cipher. We proceed in several steps, first in section 2 we briefly recall the constructions from [5], then in section 3 we describe distinguishing attacks against the two modes, finally in section 4 we show practical attacks in the context of disk-sector encryption. In one of the practical attacks we make use of Wagner's algorithm for solving the generalized birthday problem, which he introduced at Crypto'2002 in [6]. In order to improve the attack, we need to use the algorithm in a special case which was not considered by Wagner. It turns out that in this special case, the algorithm still works and moreover it becomes much more efficient than in the general case. An heuristic analysis of Wagner's algorithm restricted to these special cases is given in appendix.

2 Description of EMD and EME

In this section, we give brief descriptions of the EMD and EME modes of operation as proposed in [5]. We only describe the forward direction (encryption). For complete specifications, the reader should refer to the original paper [5].

2.1 The EMD mode

The EMD (Encrypt-Mask-Decrypt) mode of operation is based on CBC encryption. In order to simplify the description of this mode, we use two subroutines `CBCEncrypt` and `CBCDecrypt` that respectively perform CBC encryption and CBC decryption. These routines take as input a keyed block cipher E_K and a m-uple of input blocks $X_1, \ldots, X_m$ and output a m-uple $Y_1, Y_2, \ldots, Y_m$. There is a simple but important difference between ordinary CBC decryption and the CBC decryption as used in EMD. More precisely, in the latter case, individual blocks are encrypted (by the block cipher E_K) instead of being decrypted (by its inverse E_K^{-1}). The two routines `CBCEncrypt` and `CBCDecrypt` are respectively described in tables 1 and 2.

Input: E_K, X_1, X_2, $\ldots$, X_m
$Y_0 \leftarrow 0^n$
For i from 1 to m do
  $Y_i \leftarrow E_K(X_i \oplus Y_{i-1})$
Output: Y_1, Y_2, $\ldots$, Y_m

Table 1. The `CBCEncrypt` subroutine

Input: E_K, X_1, X_2, $\ldots$, X_m
$X_0 \leftarrow 0^n$
For i from 1 to m do
  $Y_i \leftarrow E_K(X_i) \oplus X_{i-1}$
Output: Y_1, Y_2, $\ldots$, Y_m

Table 2. The `CBCDecrypt` subroutine

Given these two subroutines, the description of the EMD works as follow. Starting for a plaintext sector P_1, P_2, $\ldots$, P_m, first apply `CBCEncrypt`, then perform the exclusive-or of the intermediate values with a mask which is computed as a function of the tweak parameter T and the first and last intermediate values. Afterward, reverse the block order and apply `CBCDecrypt`. This is summarized in table 3.

Note that the mask computation makes use of a function `multx` which represents multiplication by x in the finite field $\mathbb{F}_{2^n}$. Furthermore, in order to perform decryption, it should be noted that the mask value can also be computed as a function of CCC_1 and CCC_m. Indeed, $CCC_1 \oplus CCC_m$ is always equal to $PPP_1 \oplus PPP_m$.

Input: E_K, T, P_1, P_2, $\ldots$, P_m
$(PPP_1, PPP_2, \cdots, PPP_m) \leftarrow \texttt{CBCEncrypt}(E_K, P_1, P_2, \cdots, P_m)$
$Mask \leftarrow \texttt{multx}(PPP_1 \oplus PPP_m) \oplus T$
$(CCC_1, CCC_2, \cdots, CCC_m) \leftarrow (PPP_m \oplus Mask, PPP_{m-1} \oplus Mask, \cdots, PPP_1 \oplus Mask)$
$(C_1, C_2, \cdots, C_m) \leftarrow \texttt{CBCDecrypt}(E_K, CCC_1, CCC_2, \cdots, CCC_m)$
Output: C_1, C_2, $\ldots$, C_m

Table 3. The EMD mode of operation

Input: E_K, X_1, X_2, ..., X_m
$L \leftarrow E_K(0^n)$
For i from 1 to m do
$\quad Y_i \leftarrow E_K(X_i \oplus iL)$
Output: Y_1, Y_2, ..., Y_m

Table 4. The `ParaEncrypt` subroutine

Input: E_K, X_1, X_2, ..., X_m
$L \leftarrow E_K(0^n)$
For i from 1 to m do
$\quad Y_i \leftarrow E_K(X_i) \oplus iL$
Output: Y_1, Y_2, ..., Y_m

Table 5. The `ParaDecrypt` subroutine

2.2 The EME mode

The EME mode is a parallelizable variant of EMD, it makes use of various techniques that were introduced for parallelizable modes of operations in papers such as [3, 2, 1]. In order to simplify the description of this mode, we introduce two subroutines `ParaEncrypt` and `ParaDecrypt`. These routines take as input a keyed block cipher E_K and a m-uple of input blocks $X_1, \ldots, X_m$ and output a m-uple $Y_1, Y_2, \ldots, Y_m$. They are described in tables 4 and 5. In the two subroutines, the notation iL represents the multiplication in $\mathbb{F}_{2^n}$ of the element L by the element having the same binary representation as the integer i.

Given these two subroutines, the description of the EME works as follow. Starting for a plaintext sector $P_1, P_2, \ldots, P_m$, first apply `ParaEncrypt`, then perform the exclusive-or of the intermediate values with a mask which is computed as a function of the tweak parameter T and the exclusive-or of all intermediate blocks. Afterward, apply `ParaDecrypt` and obtain the ciphertext. This is summarized in table 6. In this table, the equation $CCC \leftarrow PPP \oplus Mask$ is a shorthand notation that means that the exclusive-or with the mask value is performed, in parallel, on each block of PPP.

3 Distinguishers against EMD and EME

In order to show that the EMD and EME are not secure as tweakable block ciphers, we propose simple attacks that distinguish these constructions from random tweakable permutations. These distinguishing attacks are extremely efficient, since they require only 3 calls to the encryption/decryption oracle and succeed with overwhelming probability.

Input: E_K, T, P_1, P_2, $\ldots$, P_m
$(PPP_1, PPP_2, \cdots, PPP_m) \leftarrow \texttt{ParaEncrypt}(E_K, P_1, P_2, \cdots, P_m)$
$Mask \leftarrow \texttt{multx}(PPP_1 \oplus \cdots \oplus PPP_m) \oplus T$
$CCC \leftarrow PPP \oplus Mask$,
$(C_1, C_2, \cdots, C_m) \leftarrow \texttt{ParaDecrypt}(E_K, CCC_1, CCC_2, \cdots, CCC_m)$
Output: C_1, C_2, $\ldots$, C_m

Table 6. The EME mode of operation

We briefly recall the usual framework for distinguishing attacks. Assume that we are given access to a black-box oracle that contains either an implementation of EMD (or EME) or a truly random tweakable permutation (i.e., a family of truly random permutations indexed by the tweak value). We are allowed to perform both encryption and decryption queries with the oracle and we fully control the plaintext/ciphertext values and the tweak values in these queries. After some queries, we should guess whether the black-box is an implementation of EMD (resp. EME) or a truly random tweakable permutation. We are considered successful if our probability of producing a correct guess is significatively better than 1/2.

3.1 The case of EMD

In order to attack EMD we proceed as follows. First, choose a random plaintext $P^{(1)} = (P_1^{(1)}, P_2^{(1)}, \cdots, P_m^{(1)})$, a random tweak value T and obtain the corresponding ciphertext $C^{(1)} = (C_1^{(1)}, C_2^{(1)}, \cdots, C_m^{(1)})$. Then, ask for the decryption of the ciphertext $C^{(1)}$ with tweak $T \oplus 1$. Denote by $P^{(2)}$ the corresponding plaintext.

Finally, form the plaintext $P^{(3)}$ as follows:

- $P_1^{(3)} = P_1^{(1)}$,
- $P_2^{(3)} = P_2^{(2)} \oplus 1$,
- $P_3^{(3)} = P_3^{(1)} \oplus 1$,
- For i in $\{4, \cdots, m\}$ let $P_i^{(3)} = P_i^{(1)}$.

Then we ask for the encryption $C^{(3)}$ of $P^{(3)}$ with tweak T and check whether the following conditions hold.

- Check that $C_m^{(3)} = C_m^{(1)} \oplus 1$,
- No condition to check on $C_{m-1}^{(3)}$.
- For i in $\{1, \cdots, m-2\}$ verify that $C_i^{(3)} = C_i^{(1)}$.

If the conditions hold, we guess that the black-box oracle contains an implementation of EMD, otherwise, we guess that it is a truly random tweakable permutation. Clearly, with a truly random tweakable permutation, the conditions hold with negligible probability $2^{-(m-1)n}$. However, with an implementation of EMD it is easy to check that they always hold. In order to verify this claim, assume that the intermediate values PPP and CCC in the above computations are respectively denoted by $PPP^{(1)}$, $CCC^{(1)}$, $PPP^{(2)}$, $CCC^{(2)}$ and $PPP^{(3)}$, $CCC^{(3)}$.

Then, remark that $CCC^{(2)} = CCC^{(1)}$, and furthermore that $PPP^{(2)} = PPP^{(1)} \oplus 1$, where the exclusive-or is performed on every block in $PPP^{(1)}$. For the third call to EMD, we have $PPP_1^{(3)} = PPP_1^{(1)}$, since $P_1^{(3)} = P_1^{(1)}$, and

$$\begin{aligned} PPP_2^{(3)} &= E_K(P_2^{(3)} \oplus PPP_1^{(3)}) \\ &= E_K(P_2^{(3)} \oplus PPP_1^{(1)}) \\ &= E_K(P_2^{(2)} \oplus 1 \oplus PPP_1^{(1)}) \\ &= E_K(P_2^{(2)} \oplus PPP_1^{(2)}) \\ &= PPP_2^{(2)} = PPP_2^{(1)} \oplus 1. \end{aligned}$$

Furthermore

$$\begin{aligned} PPP_3^{(3)} &= E_K(P_3^{(3)} \oplus PPP_2^{(3)}) \\ &= E_K(P_3^{(3)} \oplus PPP_2^{(2)}) \\ &= E_K(P_3^{(1)} \oplus 1 \oplus PPP_2^{(1)} \oplus 1) \\ &= E_K(P_3^{(1)} \oplus PPP_2^{(1)}) \\ &= PPP_3^{(1)}. \end{aligned}$$

We can also check that for all i from 4 to m, we have $PPP_i^{(3)} = PPP_i^{(1)}$. As a consequence, $CCC^{(3)}$ and $CCC^{(1)}$ are equal on all blocks except one. The single block that differs is block number $m-1$, which satisfy the relation $CCC_{m-1}^{(3)} = CCC_{m-1}^{(1)} \oplus 1$. The relations between ciphertexts $C^{(1)}$ and $C^{(3)}$ immediately follows.

As a consequence, the attacker we just proposed always outputs the correct answer when the black box contains an implementation of EMD. Moreover, when the black box contains a truly random tweakable permutation the answer is correct with probability $1 - 2^{-(m-1)n}$. Thus the attacker succeeds with overwhelming probability and the EMD mode of operation does not realize a secure tweakable block cipher.

3.2 The case of EME

With EME, a variant of the above attack can be derived. As before, choose a random plaintext $P_1^{(1)}$, $P_2^{(1)}$, $\ldots$, $P_m^{(1)}$, a random tweak value T and obtain the corresponding ciphertext $C_1^{(1)}$, $C_2^{(1)}$, $\ldots$, $C_m^{(1)}$. Then, ask for the decryption of the ciphertext $C^{(1)}$ with tweak $T \oplus 1$. Denote by $P_1^{(2)}$, $P_2^{(2)}$, $\ldots$, $P_m^{(2)}$ the corresponding plaintext. Form the plaintext $P^{(3)}$ as follows:

- $P_1^{(3)} = P_1^{(2)}$,
- $P_2^{(3)} = P_2^{(2)}$,
- For i in $\{3, \cdots, m\}$ let $P_i^{(3)} = P_i^{(1)}$.

Then, ask for the encryption $C^{(3)}$ of $P^{(3)}$ with tweak T and check whether the following conditions hold.

- No condition to check on $C_1^{(3)}$ and $C_2^{(3)}$.
- For i in $\{3, \cdots, m\}$ verify that $C_i^{(3)} = C_i^{(1)}$.

It is easy to verify that the mask values in the first and third encryption are equal. As a consequence, the above conditions always hold. Therefore, as for EMD, we obtain an efficient attacker that distinguishes between the EME mode and a truly random tweakable permutation.

4 Practical attacks in the disk-sector encryption context

In this section, we revisit the attacks on EMD and EME to turn them into practical attacks that can decrypt any ciphertext present on a disk sector. We study the security of the modes in the context of disk-sector encryption for multi-user operating system. We assume that the disk encryption key is fixed, known to the operating system but protected from malicious users. We also assume that malicious users can read and write any ciphertext sectors at the hardware level. However, the operating system ensures that users can read or write plaintext sectors only when the sector is part of a file which belongs to the user. Note that the operating system cannot prevent a malicious user to access the disk at hardware level in all cases. Indeed, with physical access to the machine, it is possible to reboot using a different system or even to remove the hard disk from the computer and access it with separate harware. We consider two different kinds of implementations. In basic implementations, we assume that the tweak value of a sector is the sector number. Moreover, we assume that a malicious user can choose the numbers of the sectors that are allocated to his files. In cautious implementations, we assume that the tweak values are obtained by hashing the sector numbers to full-size tweaks and that no user can choose his sectors. As a consequence, the tweak values are known but cannot be controled by malicious users.

In the system, at least two users are present, the target of the attack Alice and the attacker Eve. Alice has written sensitive information, say $P_1, \ldots, P_m$, in a disk sector with tweak value T . Eve has obtained the associated ciphertext $C_1, \ldots, C_m$ by accessing the disk hardware. Her goal is by playing around with other sectors which she legitimately accesses to fool the operating system and force it to reveal P.

4.1 Attack of basic implementations

With basic implementations, Eve chooses two convenient values δ_1 and δ_2. She asks the operating system to give her ownership of the three sectors numbered

$T \oplus \delta_1$, $T \oplus \delta_2$ and $T \oplus \delta_1 \oplus \delta_2$. If the operating system cannot comply, she simply chooses different values of δ_1 and δ_2 and she restarts.

Once Eve holds the three sectors, she copies Alice's ciphertext C in sector $T \oplus \delta_1$ and reads some plaintext $P^{(1)}$ in the sector. Then she writes $P^{(1)}$ in sector $T \oplus \delta_2$ and recovers some ciphertext $C^{(2)}$. Finally, she moves $C^{(2)}$ in sector $T \oplus \delta_1 \oplus \delta_2$ and reads a plaintext $P^{(3)}$. We claim that $P^{(3)}$ is in fact Alice's plaintext P.

In order to prove our claim, we denote by PPP, CCC, $PPP^{(1)}$, $CCC^{(1)}$, $PPP^{(2)}$, $CCC^{(2)}$, $PPP^{(3)}$ and $CCC^{(3)}$ the intermediate values in the encryption/decryption calls. Then we check that the following relations hold:

1. $CCC^{(1)} = CCC$,
2. $PPP^{(1)} = PPP \oplus \delta_1$,
3. $PPP^{(2)} = PPP^{(1)}$,
4. $CCC^{(2)} = CCC \oplus \delta_1 \oplus \delta_2$,
5. $CCC^{(3)} = CCC^{(2)} = CCC \oplus \delta_1 \oplus \delta_2$,
6. $PPP^{(3)} = PPP$.

The final equality implies that $P^{(3)} = P$ and concludes the proof of the claim.

As a consequence, Eve has attained her goal and obtained the decryption of Alice's sensitive information. We note that the very same attack also applies to the parallelizable mode EME.

4.2 Attack of cautious implementations

With cautious implementations, Eve can no longer obtain ownership of sectors whose tweaks satisfy the correct relation. As a consequence, the attack becomes harder. However, by creating files with many disk-sectors, Eve can obtain a large set $\mathcal{T}$ of random looking tweaks. If she could find a subset (of odd size) T_1, T_2, $\dots T_{2k+1}$ of elements in $\mathcal{T}$ satisfying the relation:

$$T = T_1 \oplus T_2 \oplus \cdots \oplus T_{2k+1},$$

she could apply a generalization of the attack presented in subsection 4.1. Indeed, when decrypting a ciphertext with tweak value T_{2i+1} and re-encrypting with tweak value T_{2i+2}, we simply offset the intermediate value by an exclusive-or with $T_{2i+1} \oplus T_{2i+2}$. Thus, by alternatively decrypting and encrypting with T_1, T_2, $\dots$, T_{2k}, we offset the original value of CCC by $T_1 \oplus T_2 \oplus \cdots T_{2k} = T \oplus T_{2k+1}$. As a consequence, the final decryption with tweak T_{2k+1} reveals Alice's plaintext.

In order to evaluate the efficiency of the proposed attack, we need to explain how to find a good subset in $\mathcal{T}$. Of course, as soon as $\mathcal{T}$ has more that $n+1$ elements (where n is the tweak size in bits), we expect to find a solution with high probability. Indeed, in a set $\mathcal{T}$ with $n+1$ elements, there are 2^{n+1} possible subsets among which 2^n have odd size. Since T is a n–bit value, we expect that the exclusive-or of at least one of the subsets of odd size will be equal to T, with constant probability[1]. However, to compute such a subset, simple

[1] More precisely, the expected probability tends to $1 - e^{-1} \approx 0.63$ as n tends to infinity.

algorithms such as exhaustive search (with runtime 2^n) or basic time-memory tradeoffs (with runtime $2^{n/2}$) are too slow to be effective in our case. Indeed, if the adversary is able to perform computation of the order of 2^n (or even $2^{n/2}$), he can try exhaustive search attacks against the key of the elementary block cipher.

Yet, at Crypto'2002, Wagner presented in [6] an efficient algorithm that addresses this problem. Given 2^t independent lists of elements, $L_1, L_2, \ldots, L_{2^t}$, his algorithm outputs 2^t elements $x_1, x_2, \ldots, x_{2^t}$ (one from each list) such that:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_{2^t} = 0.$$

This algorithm runs in time $O(2^t \cdot 2^{n/(1+t)})$ and requires lists of size $O(2^{n/(1+t)})$.

Wagner's algorithm can be fitted to our need, by first choosing one arbitrary element T' in $\mathcal{T}$, then by constructing the lists in the following way. Divide the rest of $\mathcal{T}$ in 2^t lists of equal length, $L_1, L_2, \ldots, L_{2^t}$. Then, modify one of the lists by xoring it with $T \oplus T'$. Clearly, any solution given by the algorithm of Wagner can be translated into a equation of the form:

$$T_1 \oplus T_2 \oplus \cdots \oplus T_{2^t} = T \oplus T',$$

which can be used by Eve to perform our attack.

The next question is to determine the optimal value of t and see how efficient is the attack derive from this algorithm. A similar problem was already addressed by Wagner in [6] in order to attack the blind signature scheme of Schnorr. The optimal choice is $t = \sqrt{n}$ which leads to a sub-exponential attack with a running time $O(2^{2\sqrt{n}})$. With a typical block cipher size, i.e. $n = 128$ the optimal choice is $t = 11$ and it leads to a very practical attack, where Eve needs to allocate about 2^{22} sectors[2] and to perform a computation of the order of 2^{22} operations. As a result, Eve obtains a relation with 2048 terms and can recover Alice's plaintext after 2049 encryption/decryption steps.

In fact, by slightly modifying Wagner's algorithm, it is possible to obtain much better performance. The modification is straightforward, however, in order to verify that it works, it should be analyzed carefully. We perform this analysis in appendix. The idea of the modification, is to considerer lists $L_1, L_2, \ldots, L_{2^t}$ which are no longer independent. In that case, we can still apply the algorithm of Wagner and hope that it will work. Of course, since the lists are no longer independent, we need fewer elements, moreover, by removing multiple copies of some computations, we greatly speed up the algorithm. In fact, if some care is taken, this really works and we obtain an even better version of Wagner's algorithm. Details are given in the appendix, and for the sake of simplicity, we only give the raw results in the present section.

Once the independence condition is removed, we can build the lists as follows. The first list L_1 is simply $\mathcal{T}$ minus the element T', all lists from L_2 to L_{2^t-1} are copies of L_1. Finally, the last list L_{2^t} is a copy of L_1 where each element is

[2] 2 Gbytes with 512 bytes sectors.

t	List size	File size (512 bytes sectors)	Number of En/Decryption	Runtime	Number of solutions found
11	4096	2 Mbytes	2049	0.3 s	93
10	8192	4 Mbytes	1025	0.5 s	101
9	16384	8 Mbytes	513	1.1 s	18
8	65536	32 Mbytes	257	4.2 s	631
7	131072	64 Mbytes	129	7.7 s	2
6	1048576	512 Mbytes	65	60.5 s	5
5	8388608	4 Gbytes	33	480.6 s	67

Table 7. Possible trade-offs for $n = 128$ with dependent lists.

xored with $T \oplus T'$. As before, any solution can be translated into a equation of the form:

$$T_1 \oplus T_2 \oplus \cdots \oplus T_{2^t} = T \oplus T',$$

and can be used by Eve to perform the attack. Due to the improved performance, several different tradeoffs are now possible. To choose between these tradeoffs, Eve needs to take into account two parameters, the size of the temporary files she can create to mount the attack and the number of encryption/decryption she can perform. Note, that in certain contexts, reading and writing the encrypted disks sector can be easily done in software, while in other contexts low-level access to the hardware need to be performed. In the second case, Eve might want to limit the number of accesses. At one extreme, Eve can choose $t = 11$, create lists of 4096 sectors[3] and perform at most 2049 encryption/decryption steps. At the other, Eve can choose $t = 5$, create lists of 8 388 608 sectors[4] and perform at most 33 encryption/decryption steps. We have implemented the algorithm, set ourselves a typical challenge (which is precisely described in the appendix) and run the algorithm for all values of t in the range from 5 to 11. The tradeoffs we used, the number of solutions found and the runtimes on a 1.5Ghz Pentium 4 are presented in table 7. One particular solution for $t = 5$ in given in table 8 at the end of the appendix. Since the runtimes range from a few seconds to a few minutes, we conclude that even careful implementations of disk-sector encryption using the EMD (or EME) mode of operation are utterly insecure.

5 Conclusion

In this paper, we have shown that the modes of operation EMD and EME proposed by Rogaway in [5] for application such as disk-sector encryption are not secure. A question of practical interest is to correct these modes of operations or to find secure alternatives. Indeed, there is a real need for tweakable block ciphers with large blocksize in practical applications.

[3] 2 Mbytes with 512 bytes sectors.
[4] 4 Gbytes with 512 bytes sectors.

As a side bonus, we remarked that Wagner's algorithm as described in [6] can also be applied to lists which are not independent. This special case of the algorithm could be used for many application and when applicable, it performs even better that the general case.

References

1. J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In L. Knudsen, editor, *Advances in Cryptology – Eurocrypt'2002*, volume 2332 of *Lectures Notes in Computer Science*, pages 384–397. Springer, 2002.
2. S. Halevi. An Observation regarding Jutla's modes of operation. Crytology ePrint archive, Report 2001/015, available at `http://eprint.iacr.org`.
3. C. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *Advances in Cryptology – Eurocrypt'01*, volume 2045 of *Lectures Notes in Computer Science*. Springer-Verlag, 2001.
4. M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. In M. Yung, editor, *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lectures Notes in Computer Science*, pages 31–46. Springer, 2002.
5. P. Rogaway. The EMD mode of operation (a tweaked, wide-blocksize, strong PRP), September 26th, 2002. Crytology ePrint archive, Report 2002/148, available at `http://eprint.iacr.org`.
6. D. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lectures Notes in Computer Science*, pages 288–303. Springer, 2002.

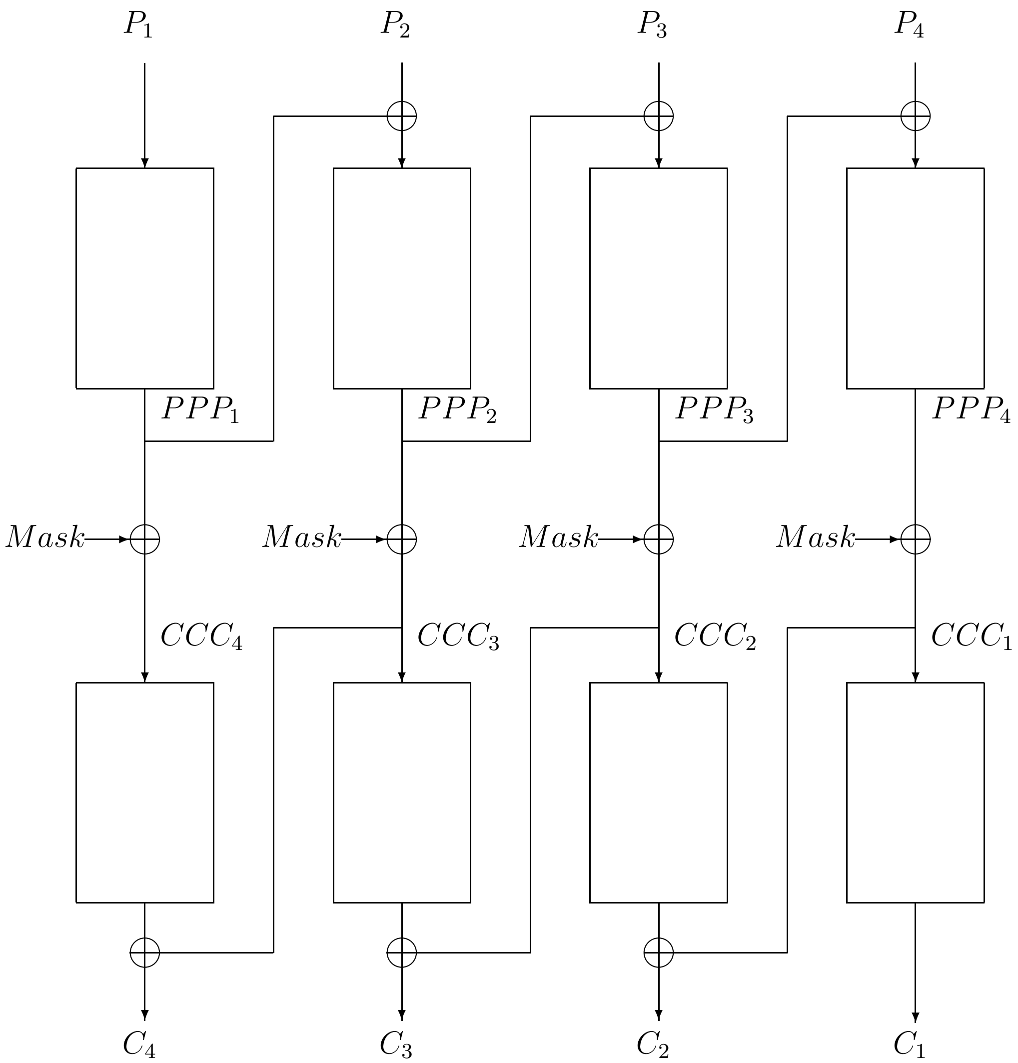

Fig. 1. The EMD mode of operation with 4 blocks

Sector number	Hash value (SHA-1 truncated to 128 bits)
-1	7984B0A0E139CABADB5AFC7756D473FB
0	B6589FC6AB0DC82CF12099D1C2D40AB9
513133	C3953CC09E9A27881177A493460A9D48
545527	2B1DA5A51E2C3A59021F03E4DF3FC186
680726	4D24927467F93AB9D9E9C396C9579544
733840	336B7B518C8B5AACF9874F5F3FDA91DD
736225	CAF64CA5F17DEE1B0D2167E4DBB7DCC0
741657	972861966AD02300242C190D04D96F3A
894941	CBCD0CC6D0B1C32DEAEBFB9DD6C39534
1181280	9728620BB15229A230F0A6A2DF51312E
1322461	347F38B15A6BA92A4952CA3663CB4D2A
1523176	CBCD0E313BEC474FF993BB6EDC396FF3
2658089	FCEAAD0DCAAC6C78059C95DCEC7F7357
2761557	95E4AED645DB35E6FA97F4798E95AC49
3146286	F99617106239ACFF5D7FDD47C4B1107B
3200283	C3953C3615A5BA6F418A1A678D5D7563
3279188	95E4AC21AE84467D82479DD8DC739DB2
3356233	3C46230D14A977CABE345A06119BD354
3364940	CF3AE98EBFDCBFCFE7BF30F941B6390B
3370180	3C46239BD89E4767E6A9DBD15A1A0299
3416711	4D24929BFE4697F0B6899DC6386EB8E0
3622423	FCEAADFB419BCC66BF027686EA0A1411
4332954	67CF00C1B868FA617DE515DE2F80077B
5181376	336B78CC57077035818035A602E754F3
5248208	B67DF5187B3243A42C42039A8CECFB85
5330409	71B0B404746F3090B6BA5989950E20D5
5593763	67CF00577457F735D5E6B01BB9D134F8
6119129	E4C18A2CCDA5627F2E79AC30EB1ACDA6
6601576	F99616B15999B12D0364F4FB700CA182
6643369	71B0B5A54FCC432F2145A4F997A46624
6701136	B67DF530350B470B47AB8B3326A17968
6853675	CAF64C8DBF4784D9B5D1423DFF1BC842
7472044	347F3931E17D76AE0DBFA9D5D7D31754
7631272	CF3AE80E04C44070CD5D2B3C321F67F1

Table 8. A solution to our self assigned challenge for $t = 5$

A Revisiting Wagner's generalized birthday algorithm

At Crypto'2002, Wagner presented in [6] a generalized birthday problem which occurs in many subfields of cryptography. This problem is, given 2^t lists of n–bits numbers $L_1, L_2, \ldots, L_{2^t}$ to find $x_1, x_2, \ldots, x_{2^t}$, each drawn from the corresponding list such that:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_{2^t} = 0.$$

Wagner also proposed an algorithm to solve this problem, together with some variants where the $\oplus$ operation is replaced by addition modulo 2^n or modulo q. However, he only considered the case where the lists L_i are built independently at random. We claim that his algorithm still works when the lists L_1 up to L_{2^t} are all copies of a small number of basic lists, or even when all the lists are identical.

A.1 Overview of Wagner's algorithm

The main idea of Wagner's algorithm is to search for solutions that satisfy extra conditions. Of course, this means that many solutions of the initial problem are lost, but on the other hand, the remaining ones can be found quickly. This is done by dividing n into $t+1$ slices each containing (approximately) the same number of bits. Then, the algorithm follows a tree of successive reductions. At the first level of the tree, lists L_{2k} and L_{2k+1} are joined into a new list $L_k^{(1)}$ that contains all numbers of the form $x_{2k} \oplus x_{2k+1}$ which evaluate to zero on the first slice of bits. This basic operation is called a join operation and denoted by $\bowtie$ in [6]. When we want to specify the slice of bits on which the join operation is performed, we add a subscript. With this notation, the first level consists in operations of the form:

$$L_k^{(1)} \leftarrow L_{2k} \bowtie_1 L_{2k+1}.$$

Similarly, the i-th level of Wagner's algorithm consists in operations of the form:

$$L_k^{(i+1)} \leftarrow L_{2k}^{(i)} \bowtie_i L_{2k+1}^{(i)}.$$

At the last level (level number t) the join operation is formed simultaneously of the two slices of bits numbered t and $t+1$. By abuse of notation, we will denote this last join operation by $\bowtie_t$, thus implicitely expanding slice number t to include slice number $t+1$.

Clearly, since all lists in the same level are mutually independent, when the numbers of bits in each slice are equal to the logarithm (in base 2) of the size of the initial lists, all join operations except the last one roughly preserve the sizes of the lists. Moreover, the last join operation outputs a small list of final solutions.

At explained in [6], the join operation can be very efficiently performed by sorting the two lists and by reading the partial collisions from the sorted lists.

A.2 The case of a single list

When the lists are no longer independent, the analysis of the algorithm's behavior is much more intricate. To start this analysis, let us consider the case where all lists are equal to $L^{(0)}$. Of course, in that case, there are many trivial solutions obtained by xoring any expression of the form $x_1 \oplus \cdots \oplus x_{2^{t-1}}$ with itself. However, we would like the algorithm to output a non-trivial solution. The first step of the analysis is to determine whether such a non-trivial solution do exist. To get an expected lower bound on the number of non-trivial solutions, we give an heuristic evaluation of the number of solutions that do not contain any duplicate x_i. Clearly, no such solution may exist when the size S_0 of $L^{(0)}$ is smaller than 2^t, thus, in the sequel we assume that $S_0 > 2^t$. In that case, the number of non-ordered subsets of 2^t chosen from $L^{(0)}$, can be computed as the binomial of S_0 and 2^t. However, the probability that such a subset survives all the join operations is not easy to compute. Indeed, we need to consider all the possible orders of the subset and see if one of those do survive. Therefore, we need to consider many events, which are not independent. As a consequence, exactly computing the number of expected solutions is not straightforward, and we will only perform an heuristic analysis.

In order to perform this heuristic analysis, we remove the simplest dependences and then assume that the remaining objects are independent. Simple dependence between different orders for the same subset clearly arise when joining a list with itself. Indeed, $x \oplus x$ is always 0 and moreover $x \oplus x'$ is always equal to $x' \oplus x$. To remove these dependences, we can make a simple change to the join operation and ensure that we never create $x \oplus x$ and that we create $x \oplus x'$ only once. With this change, every non-ordered subset can occur in $(2^t)!/2^{2^t-1}$ different orders. Moreover, the probability of survival of each order can be estimated as $2^{-2^t \cdot \frac{n}{t+1}}$. Simplifying the product of these two numbers, we estimate the probability of success of a single non-ordered subset by:

$$\frac{1}{(2e)^{2^t} \cdot 2^{(n/t-t)2^t}}.$$

On the other hand, we also know that the probability of success cannot be better than the probability of satisfying the xor condition we want to obtain, i.e., 2^{-n}.

An heuristic analysis of these probabilities shows that S_0 should be at least 2 or 3 times 2^t and should be of the order of $2^{1+n/t}$. As a consequence, the size of the list is about twice the size of one of the lists in the general case of Wagner's algorithm with similar parameters n and t. Moreover, as in the general case, the optimal tradeoff is roughly for $t = n^{1/2}$ and t should not be greater than $n^{1/2}$.

Concerning the performance of the algorithm, note that since all the lists are equal, all the join operation in a single level are identical. Thus, instead of performing 2^t join operations, we only need to perform t join operations, one per level of the tree. As a consequence, the algorithm becomes much faster. Indeed, the runtime is $O(t \cdot 2^{n/t})$ instead of $O(2^{t+n/t})$. With the optimal choice, $t = n^{1/2}$, we have a runtime of $O(t \cdot 2^t)$ instead of $O(2^{2t})$.

A.3 The case of two lists

When all the lists are either L or L', Wagner's algorithm is also very fast. In the general case, with an arbitrary repartition between copies of L, 3 join paths of length t are required. For the sake of simplicity, we only address two of these cases, where the repartition is either 2^{t-1} copies of L and 2^{t-1} copies of L' or $2^t - 1$ copies of L and a single copy of L'. In these two cases, 2 join paths will suffice.

The first repartition is the simplest to deal with, it suffices to apply the join tree for 2^{t-1} copies of a single list twice, on L and on L' independently and finally to join the two resulting lists at the last level.

With the second repartition, let $L^{(0)} = L$ and $L'^{(0)} = L'$, then at each level but the last, we perform two join operations as follow:

$$L^{(i+1)} \leftarrow L^{(i)} \bowtie_i L^{(i)},$$

$$L'^{(i+1)} \leftarrow L^{(i)} \bowtie_i L'^{(i)}.$$

At the last level, we terminate by computing the join of $L^{(t-1)}$ and $L'^{(t-1)}$.

Of course, this fast evaluation of Wagner's trees can be generalized to any small number m of different lists. Then, the maximal number of join paths is bounded by $2m - 1$. Thus, in all these cases, we get a complexity of $O(t \cdot 2^t)$ elementary operations instead of $O(2^{2t})$ for the original algorithm.

A.4 Implementation, Practical challenge

In order to verify that our rough heuristic analysis is indeed correct, we implemented the algorithm and we did set up a practical challenge for ourselves. This challenge is in the spirit of cautious implementations of disk-sector encryption as described in section 4.2. We arbitrarily decided that the tweak values for sector number i would be generated by truncating to 128 bits the result of SHA-1 applied on the character string containing the decimal representation of i. Moreover, we decided to choose to special values of i for the tweak values T and T' associated to Alice's sector and to the final decryption in the attack from section 4.2. Namely, for T we choose $i = -1$ and for T' we choose $i = 0$. The rest of the values range from 1 to the total number of needed sectors. We computed this total number for various values of the parameter t (see table 7) and ran the algorithm. For $t = 5$, the solutions we found (in 8 minutes) are compact enough to include in this paper, one of them is given in table 8. The hash values in the table are written in hexadecimal.

Note: In the implementation, we took care to remove any non-trivial duplicates appearing during the computation. While very few such duplicates do appear, this step is important. Indeed, non-trivial duplicates lead to trivial solutions at the next level. Moreover, a small number of trivial solutions quickly undergoes an exponential growth, fill the available memory and cause the algorithm to fail.