Recovering a Code's Length and Synchronization from a Noisy Intercepted Bitstream

Mathieu Cluzeau
INRIA

Matthieu Finiasz
ENSTA

***Abstract*—We focus on the problem of recovering the length and synchronization of a linear block code from an intercepted bitstream. We place ourselves in an operational context where the intercepted bitstream contains a realistic noise level. We present two algorithms, one due to Valembois and the other one brand new. They are both useful in different contexts, able to verify if a given length/synchronization is correct. Using them, we were able to practically recover the synchronization of several codes.**

INTRODUCTION

Most digital communications are both encoded and encrypted. For this reason, in order to be able to perform a cryptanalysis, it is usually necessary to decode the intercepted data. Most of the time, this data is encoded using a standardized algorithm and it is thus assumed that the attacker can decode as efficiently as the legitimate recipient. However, it can happen that non-standard techniques are used. In this case, the code reconstruction problem needs to be addressed. In this article, we only focus on communications encoded using linear block codes, thus reconstruction consists in recovering a generator/parity check matrix of the code. The first step of this reconstruction consists in recovering the code's length and synchronization. Most other articles dealing with code reconstruction [6], [8], [9], [14] consider this information known and only deal with the problem of finding a parity check matrix from noisy codewords. It appears that the most efficient techniques to recover the code's length/synchronization can also be used to reconstruct the code.

This article is composed of three main sections. First, we show that looking for words in the dual code is sufficient to decide if a specific length/synchronization is correct. Then, in the second part, we present two very different techniques for searching words in the dual code. Eventually, we present some experimental results and give estimates for the maximum noise level allowing to recover a code's length and synchronization.

Previous works. This article is not the first to deal with the problem of finding the length/synchronization of a linear block code bitstream. For instance, a technique based on rank computation is presented in [1], [2]. This technique consists in computing, for all possible lengths and synchronizations, the rank of the matrix formed with the noisy codewords. If the noise level is low enough and the length and synchronization are correct, this matrix will not be of full rank. Finding the correct length/synchronization then simply consists in finding the minimum of these ranks. When the level of noise starts to increase, it is necessary to compute the rank of submatrices and hope to find "low noise zones". We compare our experimental results with those taken from [1] in Section III.

Another approach to this problem consists in using information set decoding techniques. The first to propose this idea was Planquette [12] but with very limited applications to block code reconstruction. Valembois later rediscovered this technique and applied it efficiently to the reconstruction of small block codes. The technique we present in Section II-B is directly inspired from his work but applied to the problem of finding the length/synchronization of a code.

I. DECIDING WHETHER A GIVEN LENGTH AND SYNCHRONIZATION IS CORRECT

In this section, we consider that the received bitstream was transmitted through a binary symmetric channel with cross-over probability τ. When trying to recover the length/synchronization of a code C, the first step is to be able to decide whether a given length/synchronization is correct or not. One must thus split the input bitstream into words of the given length, starting at the given synchronization, and then decide if the words obtained are indeed noisy codewords (that is, elements of a vector space with a small amount of noise). Of course, as the target vector space (the code C we are looking for) is unknown, this problem is hard. A simpler way to look at it is to consider the dual problem: instead of looking for a vector space directly, we can look for elements of its orthogonal (these are, words of the dual of C). Such orthogonal words have a probability higher than $\frac{1}{2}$ to be orthogonal to a noisy codeword (if the noise level is lower than $\frac{1}{2}$ of course). In order to decide if a length/synchronization is correct, one can thus look for dual words: as we will see, if such a word can be found then the length is correct (with a probability close to one) and the synchronization is probably not far from correct.

Suppose the correct length/synchronization is (n_0, s_0) and the length/synchronization we are testing is (n, s). After splitting the input bitstream into words of length n, we build a matrix $\mathcal{G}$ such that each line of $\mathcal{G}$ is a word. This matrix is of size $M \times n$ where $M = \lfloor \frac{\ell - s}{n} \rfloor$ if ℓ is the length of the intercepted bitstream. Looking for a word of the dual consists in finding a word h of length n such that $\mathcal{G} \times h$ is of *low weight*. We distinguish three different cases.

A. Correct length/synchronization: $n = n_0$, $s = s_0$

In this case, each line of $\mathcal{G}$ is a noisy codeword. Thus, if h is a word of the dual of C of weight w the weight of the product $\mathcal{G} \times h$ strongly depends on τ and follows a binomial distribution, centered in $\frac{M}{2}\big(1 - (1 - 2\tau)^w\big)$, with a variance of $\sigma^2 = \frac{M}{4}\big(1 - (1 - 2\tau)^{2w}\big)$. This distribution is depicted

in Figure 1 (dashed line). However, if h' is not a word of the dual of C, whatever its weight, the weight of the product $\mathcal{G} \times h'$ will follow a binomial distribution centered in $\frac{M}{2}$ with a variance $\frac{M}{4}$ (plain line in Figure 1).

If these two distributions have a small enough intersection, then it is possible to tell, with high probability, whether a word h is in the dual of C or not.

B. Correct length, incorrect synchronization: $n = n_0$, $s \neq s_0$

In this case, each line of $\mathcal{G}$ is composed of two different codewords: the first $s_0 - s \mod n$ bits belong to one word, the remaining bits to the next one (see Figure 2). We take a word h in the dual of C and cyclicly shift it (to the right) by $s_0 - s$ positions to obtain a word $\bar{h}$.

- If the support of $\bar{h}$ is included in $[s_0 - s, n-1]$ or $[0, s_0 - s - 1]$ then the product $\mathcal{G} \times \bar{h}$ will follow the same distribution as dual words in the previous case (the dotted line in Figure 1).
- If the support of $\bar{h}$ is split among the two intervals, then each bit of the product will be zero with probability $\frac{1}{2}$ (if we assume that the codewords are mutually independent). The weight of $\mathcal{G} \times \bar{h}$ will thus follow the same distribution as for random words.

Once again, if the two distributions are distant enough, it will be possible to decide whether a (low weight) word is in the dual of C or not. However, this will only work for words such that the support of $\bar{h}$ is not split. In practice, this will decrease the probability of finding words in the dual of C. The larger $s_0 - s$, the more this probability will decrease. If the chosen synchronization s is close to the correct synchronization s_0, the behavior of dual word finding algorithms will be nearly the same as in the first case.

C. Incorrect length: $n \neq n_0$

In this third case, as it can be seen in Figure 3, each codeword will have a different offset. There is a high probability that $\mathcal{G}$ is not close to any vector subspace. In practice, for any word h', the product $\mathcal{G} \times h'$ will follow the binomial distribution centered on $\frac{M}{2}$ represented by the plain line in Figure 1. The only case in which some words could follow a different distribution, is if n divides n_0 and each offset of a dual word h by n positions is also in the dual of C. If such an unlikely event occurs, a divisor of n_0 has however been found, which makes finding n_0 a much easier problem.

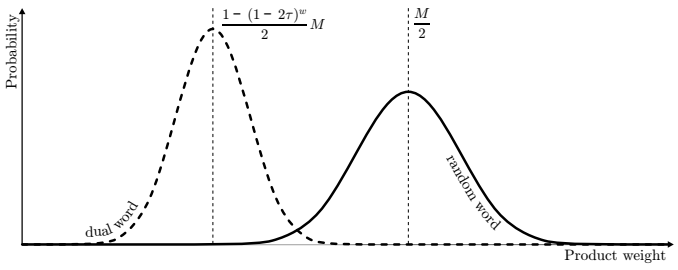

Figure 1. Distribution of the weight of the product $\mathcal{G} \times h$.

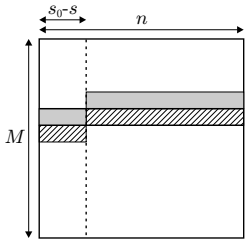

Figure 2. The matrix $\mathcal{G}$ when an incorrect synchronization was chosen.

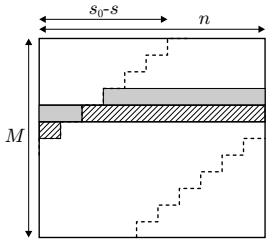

Figure 3. The matrix $\mathcal{G}$ when an incorrect length was chosen.

D. Analysis

What appears from the study of these three cases is that words following the dashed line distribution of Figure 1 can only be found when the *correct length* was chosen. If τ is known, for a given word weight w, the two distributions are known and it is thus possible to compute a threshold T. If we can find a word h such that the weight of $\mathcal{G} \times h$ is below T, then there is a high probability that the length n chosen is equal to n_0. This also means that the offset s is probably not too far from s_0. To find the exact value of s_0, it is necessary to look at the supports of dual words found. As we have seen in I-B the correct offset cannot split the support of any words. We can thus proceed with some kind of dichotomy, but this requires to find several dual words for each tested offset. In the end, when the correct synchronization has been found, we usually have found enough dual words to reconstruct the complete code C.

We will now see two different algorithms to search for words in the dual of C. For each of these algorithms it is possible to estimate the number of tries needed to find one word. It is thus possible to know that a length/synchronization pair is incorrect after a certain number of unsuccessful tries.

II. Finding Words in the Dual of C

A. Exhaustive Search of Words of a Given Weight

The first thing to note when looking for words in the dual of a code is that words of (very) low weight are easier to find. First, the two distributions of Figure 1 are more distant from one another, secondly, it can be easy to exhaustively test all words of weight w. This is exactly what our first algorithm does, using the Chose-Joux-Mitton algorithm [7].

The straightforward exhaustive search technique consists in going through all words of weight w and for each of them, computing the weight of their product with $\mathcal{G}$. If one of these weights is below a threshold T, then the corresponding word

Table I
EXAMPLES OF PARAMETERS AND COMPLEXITIES OF THE CHOSE-JOUX-MITTON ALGORITHM FOR SOME DUAL WORD WEIGHTS.

w	ℓ_1	ℓ_2	time	memory
6	2	1	$O(n^3)$	$O(n^2)$
8	2	2	$O(n^4)$	$O(n^2)$
10	3	2	$O(n^5)$	$O(n^3)$
12	3	3	$O(n^6)$	$O(n^3)$

likely belongs to the dual of C. In order to improve this technique, we can use a birthday technique. We build a list of all products $\mathcal{G} \times h_{\frac{w}{2}}$ where $h_{\frac{w}{2}}$ is a word of weight $\frac{w}{2}$ and look for "near collisions". These "near collisions" can be efficiently found by selecting a small window of size ℓ (typically $\ell = 32$ bits) on which we look for an exact match and then check if the total weight of the product is below T.

The standard birthday technique finds such collisions in $O(n^{\frac{w}{2}})$ operations and with memory $O(n^{\frac{w}{2}})$. This can be improved using the Chose-Joux-Mitton algorithm which achieves the exact same result in time $O(n^{\frac{w}{2}})$ but with memory $O(n^{\lceil \frac{w}{4} \rceil})$. The algorithm works as follow, with parameters $\ell_1 + \ell_2 = \frac{w}{2}$:

- build the list U of all XORs of ℓ_1 columns and sort it,
- for all $s \in [0, 2^S - 1]$ (with $S = \log_2 \binom{n}{\ell_2}$):
  - for all XORs x of ℓ_2 columns, find all elements of U matching $x \oplus s$ on the S first positions of the window,
  - add the corresponding XORs of $\ell_1 + \ell_2$ columns to a list C_s,
  - each collision in C_s is a possible dual word.

The list C_s will have the same size as U in average, giving us an algorithm with complexity $O(n^{\ell_2 + \max(\ell_1, \ell_2)})$ and memory $O(n^{\ell_1})$.

Probability of success. This algorithm exhaustively searches all dual words of weight w. So, if the sequence contains no noise, the algorithm recovers all such dual words in a single iteration. However, this algorithm finds $\frac{1}{2} \times \binom{w}{\frac{w}{2}}$ times each word of weight w. Thus, instead of using all $s \in [0, 2^S - 1]$, we choose to use only a fraction of this interval so as to improve the time complexity of the algorithm. The optimal fraction $\frac{1}{2^\lambda}$ to use depends on w: the probability of finding each dual word of weight w is $1 - (1 - \frac{1}{2^\lambda})^{\frac{1}{2} \times \binom{w}{\frac{w}{2}}}$. For $w = 6$, the optimal choice is $\lambda = 4$ giving a probability of finding each word of 0.48 thus improving the overall time complexity by a factor 8.

In the presence of noise, the probability of finding each dual word is simply the probability $\mathcal{P}$ of having a collision on the window of size ℓ:

$$\mathcal{P} = \left(\frac{1 + (1 - 2\tau)^w}{2} \right)^\ell .$$

If we know the number of dual words of weight w, we thus have an estimate of the number of dual words that one round of the Chose-Joux-Mitton algorithm should return. As we will see in Table III for LDPC codes, this estimate is very accurate in practice.

Computing the threshold T. We need to select T in order to avoid all false alarms (that is, words not in the dual of C with a product by $\mathcal{G}$ below the threshold) and at the same time miss as few as possible dual words. This will be possible if the threshold can be chosen at more than 3 standard deviations from the center of each distribution. If M is large enough, this will be possible.

In order for the two "3 standard deviation" bounds to be in the correct order we need:

$$M > \left(\frac{3\sqrt{1 - (1 - 2\tau)^{2w}} + 1}{(1 - 2\tau)^w} \right)^2 . \quad (1)$$

If this inequality is verified, any threshold T between the two "3 standard deviations" bounds can be chosen and should give satisfactory results. In practice we choose to select T in the exact middle of this interval which, as we will see in the last section, gives very good results. This corresponds to:

$$T = \frac{M}{2}\left(1 - \frac{(1 - 2\tau)^w}{2}\right) + 3\frac{\sqrt{M}}{4}\left(\sqrt{1 - (1 - 2\tau)^{2w}} - 1\right).$$

B. Using the Canteaut-Chabaud Algorithm

As we will see in Section III, the previous algorithm can hardly be used for values of w larger than 8. However, for most codes, the minimal distance of their dual will be larger than 8. In order to deal with these codes, we propose to use the algorithm presented by Valembois [8], [14] and based on the Canteaut-Chabaud information set decoding algorithm [5]. The description we give of the algorithm matches the way we implemented it. Here is how this algorithm works:

- select at random an "information set", that is, n lines among the M lines of $\mathcal{G}$
- perform a Gaussian elimination on this information set, swapping and xoring *columns* of $\mathcal{G}$ to obtain a new matrix $\mathcal{G}'$ (see Figure 4) and store the transition matrix P such that $P\mathcal{G} = \mathcal{G}'$
- choose a small window of l lines among the $M - n$ remaining lines of $\mathcal{G}'$
- use the same technique as in the previous algorithm to find all combinations of $2p$ columns xoring to 0 on the l lines of the window
- for each set of $2p$ columns, verify that the xor on the columns of $\mathcal{G}'$ is of weight lower than a threshold T
- each word h of weight $2p$ can be converted to a word of the dual $h' = h \times P$.

This can be implemented very efficiently using the Canteaut-Chabaud algorithm to select the information sets of successive iterations. In practice, this consists in only changing one position in the previous iteration information set so as to make the Gaussian elimination step less costly. Also, in order to optimize the probabilities, it is better to split the columns in two separate sets and look for collisions among words of weight p in each set. The optimal values for the two parameters l and p are chosen in the same way as in [4].

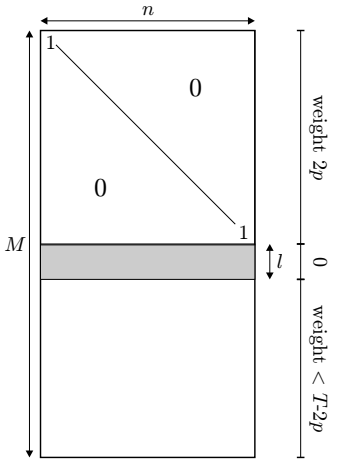

Figure 4. Using the Canteaut-Chabaud algorithm on matrix $\mathcal{G}$.

As for the previous algorithm, it would be interesting to know the probability of success of one iteration of this technique. A given word h of weight w in the dual of C will be found if the product $\mathcal{G} \times h$ is of weight:

- $2p$ on the chosen information set (in 2 sets of weight p),
- 0 on the window of size l,
- and less than $T - 2p$ on the remaining positions.

For a given information set, the probability that the errors in $\mathcal{G}$ are well distributed for the previous conditions is:

$$\mathcal{P}_{\text{cor}} = \tbinom{n}{2p} q^{2p} (1-q)^{n-2p+l} \sum_{i=0}^{T-2p} \tbinom{M-n-l}{i} q^i (1-q)^{M-n-l-i}$$

with $q = \frac{1-(1-2\tau)^w}{2}$. Thanks to this probability, it is possible to compute an estimate (neglecting the dependencies between successive iterations) of the average number of iterations required to find a given word h. However, there are many words in the dual of C and what interests us most is the average number of iterations required to find any one of these words. Unfortunately, this number is much harder to compute as it will depend on the (unknown) distribution of the words of the dual. For this reason, the threshold T will be chosen independently of this value.

We decide to choose the threshold T so as to minimize the probability of having a false alarm (that is, a word not in the dual verifying the weight conditions cited above). There are 2^n possible false candidates, and for each of them the probability of verifying the weight conditions is approximately:

$$\mathcal{P}_{\text{false}} = \frac{1}{2^M} \sum_{i=0}^{T} \tbinom{M}{i}.$$

If we want to avoid false alarms, we thus need $\mathcal{P}_{\text{false}} < 2^{-n}$. This is achieved by choosing:

$$T = \frac{M}{2} - \sqrt{M \frac{n \log 2}{2}}.$$

This choice makes sure that we avoid false alarms but gives us no hint about the probability of finding a word of the dual. However, what is known is that if we increase M (and thus also T), this probability of success will also increase. More details about the choice of M are given in the next section.

III. Practical Experiments

We have previously seen how to test a length and synchronization and how to find words in the dual code of C. Our algorithms thus consist in testing all possible lengths and for each length a number of different synchronizations. In practice, for short codes, we test each length n from 1 to n_0 and $n/8$ synchronizations each time (we test all synchronizations which are a multiple of 8). For larger values of n (especially for LDPC codes), we test a fixed number of synchronizations for each length (every $\frac{n}{8}$ bits for instance). We divide our experiments in two groups which behave very differently in practice for both algorithms presented here.

A. Random Linear Codes

We first consider codes defined by a random generator matrix. For such codes, it appears that our first algorithm based on exhaustive search gives poor results. This algorithm only works well for codes with very low weight words in their dual (weight 6 or 8 at most), which will only be the case for very short random codes (at most $n \approx 40$). Of course, such codes are seldom used in practice. However, the Canteaut-Chabaud-based algorithm makes it possible to find the length/synchronization of longer codes. Table II gives results of simulations on random codes of different lengths with different noise levels. For these lengths, it is important to note that the 10000 iterations are performed in approximately 1s. Thus, if the algorithm is able to find some words, it will also be easy to find the length/synchronization of the code. In practice, for all the length/noise combinations of Table II not containing a zero, the exact length/synchronization can be recovered in a few minutes. Results given in [1] report similar performances for small codes of rate $\frac{1}{2}$. For the particular case of a $(100, 50)$ random code, increasing the number of iterations of the Canteaut-Chabaud algorithm to a few millions (about 10 minutes of computation) allows to deal with an error probability of 0.02 whereas the algorithm from [1] fails. For the longer codes with rates closer to 1 presented in [1], our technique is successful with much higher noise levels.

Table II
Number of words found by 10000 iterations of the Canteaut-Chabaud algorithm on random codes of rate $\frac{1}{2}$. Here, $M = 5n$, ∞ means that too many words were found.

n \ τ	0.001	0.002	0.005	0.01	0.02	0.05
32	14637	27081	42570	42913	19464	210
64	∞	∞	∞	1172189	6310	0
128	∞	∞	∞	2992	0	0
256	∞	∞	0	0	0	0

For lengths longer than 256, this technique can still be successful but only if τ is very small or if particular codes with dual words of very low weight are used. This is for example the case with LDPC codes. Compared to the results

Table III
NUMBER OF WORDS FOUND BY 50 ITERATIONS OF THE CHOSE-JOUX-MITTON ALGORITHM ON LDPC CODES OF RATE $\frac{1}{2}$, LENGTH 1000 AND WEIGHT 6. HERE, $M = 512$. THE FIFTH COLUMN CORRESPONDS TO THE MINIMAL VALUE OF M GIVEN BY EQUATION (1).

τ	words found	expected words per iter.	expected total words found	minimal M	theoretical bound for M
0.01	478	41	492	56	60
0.02	251	7.5	266	70	70
0.03	84	1.5	70	88	82
0.04	15	0.33	16	110	97
0.05	6	0.08	3.9	140	117
0.06	1	0.02	1.0	180	145

presented by Valembois in 2001 [14], we do not obtain any notable improvement. This confirms the threshold effect we had noticed: eight years of computational power improvements have very little influence on the noise-wise performances of this algorithm.

B. LDPC Codes

By nature, LDPC codes [11] have words of very low weight in their dual. This makes it much easier to recover their length/synchronization. Also, our first algorithm was specifically designed for such codes.

All our simulations were done on a computer with 3GB of memory for LDPC codes with parity checks of weight 6. With our implementation, 3GB of memory were sufficient for lengths up to 10 000 but not more. Table III gives the number of words of the dual found in 50 iterations of the Chose-Joux-Mitton algorithm on LDPC codes of length 1000 for different noise levels. The optimal choice of ℓ is not obvious but will probably be around $\frac{w}{2}\log_2 n$. Here we chose $\ell = 30$. Note that one iteration takes approximately 3 seconds. One can thus see that for noise levels of 5-6% a complete length/synchronization recovery will take very long (probably a few days on a single computer). Table III also shows some interesting results:

- for $\tau \leq 0.02$, a single iteration of the algorithm is enough to verify if a length/synchronization is correct. Thus, with such noise levels, recovering the length/synchronization of the code takes a few hours. Moreover, this only requires very few intercepted words.
- for noise levels close to the correction capacity of the LDPC ($\tau = 0.06$ for instance) it is still possible to verify a length/synchronization pair. This was not obvious at first sight for such noise levels.
- in [10], Cluzeau and Tillich take an information theoretic approach to the problem of code reconstruction. They give a theoretical and asymptotic bound on the number of noisy codewords required to reconstruct an LDPC code which is reported in the last column of Table III. It appears that the minimal number of words required for our algorithm is quite close to this bound.
- as compared to the results of Barbier *et al.* [1], [2], our technique can handle significantly higher noise levels while also being much faster. For instance, with parameters similar to those of Table III, their algorithm takes several days to reconstruct an LDPC with noise level of 0.002 (assuming length and synchronization are already known).

IV. CONCLUSION

We were able to recover the length/synchronization of various linear block codes. For random codes, our algorithms were successful only for short lengths (up to 256) with very low noise level (well below the correction capacity of the code). Similar results were obtained by Valembois in the context of code reconstruction bringing us to the conclusion that length/synchronization recovery is possible every time code reconstruction is possible. This is mostly because reconstruction is fast whenever it is possible.

It is the first time that the Chose-Joux-Mitton algorithm is applied to the code reconstruction problem. It gives very good results for LDPC codes. Our experiments on codes with parity checks of weight 6 show that for length up to 10000 we can recover the length/synchronization of the code even for noise levels close to the correction capacity of the LDPC. For longer lengths, more than 3GB of memory are necessary. For parity checks of weight 8, the time complexity increases to $O(n^4)$ and thus lengths beyond 2000 are impractical. To the best of our knowledge, this is the first example of long LDPC code reconstruction in the presence of realistic noise levels. Previous works by Barbier [1] could not deal with crossover probabilities higher than 0.002. Exhaustive search approaches really seem to be the best choice for LDPC reconstruction.

REFERENCES

[1] J. Barbier. *Analyse de canaux de communication dans un contexte non coopératif*. Thèse de doctorat, École Polytechnique, November 2007.

[2] J. Barbier, G. Sicot, and S. Houcke. Algebric approach of the reconstruction of linear and convolutional error correcting codes. In *CCIS 2006*, 2006.

[3] G. Burel and R. Gautier. Blind estimation of encoder and interleaver characteristics in a non cooperative context. In *International Conference on Communications, Internet and Information Technology, CIIT*, 2003.

[4] A. Canteaut. *Attaques de cryptosystèmes à mots de poids faible et construction de fonctions t-résilientes*. PhD thesis, Paris 6, 1996.

[5] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: application to primitive narrow-sense BCH codes of length 511. *IEEE Transaction on Information Theory*, 44(1):367–378, January 1998.

[6] C. Chabot. Recognition of a code in a noisy environment. In *IEEE Conference, ISIT'07*, pages 2210–2215, 2007.

[7] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: an algorithmic point of view. In L.R. Knudsen, editor, *Eurocrypt 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2002.

[8] M. Cluzeau. Block code reconstruction using iterative decoding techniques. In *IEEE Conference, ISIT'06*, pages 2269–2273, Seattle, USA, 2006. IEEE Press.

[9] M. Cluzeau. *Reconnaissance d'un schéma de codage*. PhD thesis, École Polytechnique, 2006.

[10] M. Cluzeau and J.P. Tillich. On the code reverse engineering problem. In *IEEE Conference, ISIT'08*, pages 634–638, Toronto, Canada, 2008. IEEE Press.

[11] R. G. Gallager. *Low Density Parity Check Codes*. MIT Press, 1963.

[12] Guillaume Planquette. *Identification de trains binaires codés*. Phd thesis, Université de Rennes I, December 1996.

[13] G. Sicot and S. Houcke. Blind detection of interleaver parameters. In *ICCASP05*, 2005.

[14] A. Valembois. Detection and recognition of a binary linear code. *Discrete Applied Mathematics*, 111(1-2):199–218, July 2001.