Searching for Low Weight Codewords in Linear Binary Codes

Somitra Kumar Sanadhya, Indian Statistical Institute, Kolkata
Joint work with Palash Sarkar

In this work we revisit the known algorithms for searching for low weight codewords in linear binary codes. We propose some improvements on them and also propose a new efficient heuristic.

1. Introduction

The general problem of determining the least weight of a linear binary code is known to be NP-Complete [1]. Even the relatively easier problem of determining whether there exists a codeword with weight less than a given bound does not have any polynomial time algorithm known till date. The motivation to study such a property in linear codes stems not only from the theoretical important and challenging open questions in coding theory but also due to its practical ramifications in cryptography.

The general problem being NP-Complete, researchers have developed several algorithms to quickly find low weight codewords which are possibly close to the least weight of a given code. Some such algorithms were proposed in [4], [5] and [3]. We briefly explain these methods and then propose some improvements to them. We also propose a new method to search for low weight codewords. We use some of the algorithms discussed in this work to find low weight codeword for the linearized version of 23-step SHA-256 hash function. We hope to do detailed comparison of all the algorithms using some standard codes in coming days.

2. Overview of Existing Algorithms

Let G be the generator matrix and H be the parity check matrix of a code $\mathcal{C}$. Let the length of a codeword in $\mathcal{C}$ be n and the dimension of the code be k. Then the order of G is $(k \times n)$ and the order of H is $((n-k) \times n)$.

The following three algorithms are available in the literature for finding low weight codewords in a linear code.

(1) Leon's Algorithm [4]: This algorithm uses the generator matrix of the code.
(2) Canteaut and Chabaud's Algorithm [3]: This algorithm also uses the generator matrix of the code. It is an improvement of the Leon's algorithm.
(3) Stern's Algorithm [5]: This algorithm predates the Canteaut-Chabaud algorithm. It uses parity check matrix of the code.

Due to space restrictions, we do not describe the algorithms above. Please refer to the corresponding paper for details on the particular algorithm.

3. Our Improvements to the Implementation of the Algorithms

We next describe our implementation level improvements to the algorithms described previously. These improvements are discussed with respect to the generator matrix based approach. We applied them for the check matrix based approach as well.

3.1. **Word Oriented Approach.** We use a 32-bit word oriented approach in the implementation of all the algorithms discussed. One principal cost of the algorithms is in computing the linear combination of rows. The step requiring linear combination of rows is implemented by a *for* loop which runs as many times as the length of the column. We represent 32 columns as one 32-bit word. With this word oriented approach, the linear combination of one row with another becomes 32 times faster because a word can be linearly combined with another word in one computer clock cycle.

In our implementation of the Canteaut-Chabaud algorithm, the swap of a column from the redundant set with a column from the information set is not done explicitly. Instead, only the part of the matrix restricted to the redundant set is modified so as to reflect the effect of the swap

and the row operations which would be required to obtain identity matrix in the first part after the swap. We implement all these operation over 32-bit words.

3.2. **Increasing the Number of Row Combinations.** We experiment with obtaining a larger number of columns L from the redundant set such that the weight of the punctured code vanishes on them. As this number is increased, the probability for obtaining vanishing punctured code reduces. To be still able to obtain sufficient candidate row combinations of $2p$ rows to be tried on the full code $\mathcal{C}$, we need more rows to be linearly combined. Increasing the number of rows to combine makes sense even when the size of the punctured code is not increased. In this latter case, the quality of candidate row combinations may improve on the full code $\mathcal{C}$.

4. Our Improvements to the Techniques

We now discuss our improvements to the techniques of the algorithms described earlier.

4.1. **Using Wagner's Generalized Birthday Attack .** The algorithms discussed earlier use two lists, each of which is prepared by linear combinations of p rows of the generator (or check) matrix. The problem with increasing the number of rows to combine, for example p in the previous case, is that it is not practically feasible to attempt $\binom{k/2}{p}$ combinations when p is more than 3 or 4 for moderate values of k. To handle larger values of p, we incorporate a technique from [6]. The method in [6], which was earlier used in a special case in [2], is an extension of the birthday paradox in cryptography.

This time we create four lists of linear combinations of p rows on four disjoint sets of L columns from the redundant set. Using the tree based matching techniques of [6], we then find matching pairs restricted to the L columns from these four lists. Each matching pair gives us a combination of $4p$ rows which vanish on a punctured code of length L. Each of these matches is a good candidate for low weight codeword for the full code $\mathcal{C}$.

We can extend the approach outlined above and create 8, 16, $\ldots 2^t$ lists and then use a tree based method described in [6] to find matches in the lists. This will produce combinations of $2^t p$ rows, each of which vanishes on a punctured code of size L. This method can be useful when the length of the full code $\mathcal{C}$ is quite large. We investigated creation of up to 4 lists in the redundant set since the example codes we considered were not larger than length 512.

4.2. **Closeness Parameterization of the Punctured Code.** If the length of the punctured code is increased without increasing the number of row combinations, the probability of obtaining a codeword which vanishes on the punctured code is lowered. This means that some good candidate row combinations will not be considered while examining the full code. To handle this, we do not require the codewords to vanish completely on the punctured code, rather we look for a *low weight punctured code*. We implement this by a *closeness parameter*. A row combination which produces a codeword of weight less than *closeness parameter* in $\mathcal{C}'$ is examined as a candidate for low weight code for the full code $\mathcal{C}$.

The same technique can also be used in the multi-list approach discussed in Section 4.1. In that situation, the number of rows to combine have increased but we may not wish to lose on the candidate row combinations which produce a very low weight in the punctured code. We use a measure of *closeness of match* to keep such close matches in the candidate list for the full code.

5. A New Efficient Heuristic

The proposed heuristic works on the parity check matrix of the code. It starts with a random ordering of the columns of the check matrix $H_{(n-k)\times n}$. We attempt to obtain a weight $w = 2w_1$ using the proposed method.

We first partition the columns of the check matrix into two sets, say X and Y. The two sets contain $n/2$ columns each. We then create a list $\mathcal{L}_1$ by linearly combining w_1 columns from X. Each element of the list is a column vector of length $(n-k)$. Since the number of all w_1 combinations of $n/2$ columns is $\binom{n/2}{w_1}$, it may not be feasible to attempt all such combinations exhaustively. Therefore we propose to use a parameter l for the size of this list. We shortly

TABLE 1. Least weight codeword of weight 79 for 23-step linearized version of SHA-256 found using Canteaut-Chabaud algorithm [3] with $p = 2$ and $L = 10$. The 16 words of 32-bit each are listed in hexadecimal.

0-3	c0080020	a0900121	40010212	0100088a
4-7	00100800	80020818	80808d02	40082000
8-11	42000682	92002000	409a0200	10200002
12-15	00590020	28025081	44100100	10520050

describe a suitable value for this parameter. Similarly, we create a list $\mathcal{L}_2$ by linearly combining w_1 columns from Y. We propose to use the same size l for the list $\mathcal{L}_2$ as well.

If we can find a match in the two lists $\mathcal{L}_1$ and $\mathcal{L}_2$, then we have obtained $2w_1$ columns whose linear combination is zero and hence we have a codeword of weight $w = 2w_1$. In this case the heuristic is successful. If it is not, then we restart the search by randomizing the column ordering again. To obtain success, we need to ensure that the two lists have some intersection. Since there are a total of $2^{(n-k)}$ possible column vectors, birthday bound requires the size of the lists l to satisfy the bound $l \times l \geq 2^{(n-k)}$. Thus we need $l \geq 2^{(n-k)/2}$.

The above bound for l may still be large and it may be difficult to prepare lists this long. In such a case, we can extend the two list approach and prepare 4 or 8 or $\ldots 2^t$ lists, and look for a match among these lists by using Wagner's tree-based approach [6]. The advantage of our method is that it does not require any Gaussian to be performed. The cost of the algorithm is only in the preparation and matching of the lists, hence a single iteration of our algorithm is likely to be more efficient than a single iteration of other algorithms. We plan to do analysis of the rate of convergence of our algorithm later.

6. Results and Conclusions

We tested the algorithms of Leon, Stern and Canteaut-Chabaud on a code prepared by linearizing the 23-step SHA-256 hash function. In the linearization, all additions were replaced by XOR and the non-linear 3-input boolean functions f_{IF} and f_{MAJ} were approximated by their middle arguments. In the SHA-256 hash function, one block (512-bit) message is mapped to 8 registers of 32-bits each. Thus, its linearized version corresponds to a code having generator matrix of order 256×512. We tested the algorithms with improvements described in Sections 3 and 4. We found that the algorithm by Canteaut and Chabaud gave the best results. In a code of size 512, the method could produce a codeword of weight 79 which is detailed in hexadecimal in Table 1. We are still implementing our new method described in Section 5.

References

[1] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg, *On the Inherent Intractability of Certain Coding Problems*, IEEE Transactions on Information Theory **24** (1978), 384–386.

[2] Paul Camion and Jacques Patarin, *The Knapsack Hash Function proposed at Crypto'89 can be broken*, Advances in Cryptology - Eurocrypt '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings (Donald W. Davies, ed.), Lecture Notes in Computer Science, vol. 547, Springer, 1991, pp. 39–53.

[3] Anne Canteaut and Florent Chabaud, *A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511*, IEEE Transactions on Information Theory **44** (1998), no. 1, 367–378.

[4] Jeffrey S. Leon, *A probabilistic algorithm for computing minimum weights of large error-correcting codes*, IEEE Transactions on Information Theory **34** (1988), no. 5, 1354–1359.

[5] Jacques Stern, *A Method for Finding Codewords of Small Weight*, Coding Theory and Applications (Gérard D. Cohen and Jacques Wolfmann, eds.), Lecture Notes in Computer Science, vol. 388, Springer, 1988, pp. 106–113.

[6] David Wagner, *A Generalized Birthday Problem*, Advances in Cryptology - Crypto 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings (Moti Yung, ed.), Lecture Notes in Computer Science, vol. 2442, Springer, 2002, pp. 288–303.