3TH INTERNATIONAL CONFERENCE ON COMBINATORICS, CRYPTOGRAPHY AND COMPUTATION
Iran University of Science &Technology

Proceedings of the 3nd International Conference on Combinatorics, Cryptography and Computation (I4C2018)

A new fast algorithm for decoding the [47,24,11] quadratic residue code

Zeinab Roostaie, Mohammad Gholami
Department of Mathematics, Faculty of Mathematical Sciences

Shahrekord. Iran

zeinab.roostaie@gmail.com ; gholami-m@sci.sku.ac.ir

ABSTRACT

In this paper, we present a new method for decoding [47,24,11] quadratic residue (*QR*) code which is the reduction of the required memory compared to the full lookup table or other known algebraic decoding methods. The idea behind this decoding technique is based on the existence of a one-to-one relation between the syndrome and correctable error patterns. In this approach, errors are directly found from the table and no multiplication operation over a finite field is required. Because of using the cyclic structure of codes, weight of syndrome and a reduced lookup table, the proposed algorithm can be applied on many other cyclic codes.

KEYWORDS: error pattern, finite field, quadratic residue codes, syndrome.

1 INTRODUCTION

In this paper, we present a new method for decoding [47,24,11] quadratic residue (*QR*) code which is the reduction of the required memory compared to the full lookup table or other known algebraic decoding methods. The well-known quadratic residue (*QR*) codes, first discussed in 1958 by prange [1,2], is one of the best-known examples of binary cyclic codes with code rate close to 1/2 and generally large minimum distance. In the past decades, a series of different decoding methods have been developed to decode the *QR* codes[3]- [5], [6]- [13], [18]. Nevertheless, these algebraic decoding techniques require a large number of operations in finite field. These complicated computations lead to a time delay in the decoding procedure and therefore the decoding time will become unrealistic when the code length is large. Recently, a new method, called table lookup decoding algorithm (TLDA)[16], is proposed for decoding of cyclic *QR* codes. In general, the TLDA makes possible a faster decoding time for cyclic codes in comparison with the algebraic decoding algorithm. Moreover, it is practical for hardware or firmware implementations when the code length is not too large. For example, an efficient TLDA to decode the long binary systematic [47,24,11] *QR* code[6] is required so that it can allow for the correction of up to $t = \lfloor (d_{\min}(c) - 1)/2 \rfloor = \lfloor (11-5)/2 \rfloor = 5$ errors. The full lookup table in the conventional table lookup decoding algorithm (CTLDA) needs $\sum_{i=1}^{5} \binom{47}{i} = 1729647$ syndromes and their corresponding error

patterns and it requires $1729647 \times (6bytes + 3bytes) \cong 14.85$ mega bytes (MB) memory stored. Such a large memory required makes the computation very complicated. In another effort, Chen[12], used a lookup table decoding algorithm (LTDA) with a memory size 1.05 MB to decode the binary systematic [47,24,11] QR code. However, the size of the memory required to store this lookup table is still too large. Afterward, Chang[14] recommended a method for decoding of binary *QR* codes that due to the property of the cyclic codes, the required memory size reduced up $1/n$ time of its original size, where n is the code length of the code. In fact, it is verified in [14] that for decoding the [47,24,11] *QR* only needs 36.6 KB memory size, which includes $\sum_{i=1}^{4}\binom{47}{i}/47 = 4164$ syndromes together their corresponding error patterns.

In this paper, using the properties of cyclic codes, weight of syndrome, reduced lookup table, and a code rate close to 1/2, a new algorithm, called reduced lookup table decoding algorithm (RLTDA) is proposed to decode the [47,24,11] *QR* code, which has this property that it only needs 2.6 KB memory size, including 300 syndromes and their corresponding error patterns. Such a small memory size can be easily embedded in the memory of the micro controller chips[19]. In general, the proposed method can be extended for each cyclic code with the parity-check matrix $H=[I_{n-k}|P_k]$ in which k ~n-k i.e. cyclic codes with rate close to 1/2.

The outline of the paper is organized as follows. Section II describes the background and some preliminaries of the binary linear cyclic and *QR* codes, then Section III explains the proposed decoding algorithm. and Section IV is the conclusion.....

2 PRELIMINARIES AND CONSTRUCTIONS

Let n,k be two positive integers. By a (binary) [n,k] linear code, we mean a dimension k linear subspace of $F_2^n = \{\ x_0x_1...x_{n-1} : x_i \in F_2\}$, where $F_2 = \{0,1\}$ is the finite field of order 2. An [n,k] linear code C is said to be cyclic if for each $c = c_0c_1...c_{n-1} \in C$, we have $c' = c_{n-1}c_0c_1...c_{n-2} \in C$, i.e the right cyclic shift of each codeword must also be a codeword. According to the polynomial form, each codeword $c \in C$ can be shown by the code polynomial $c(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1}$ in $F_2[x]$ Now, the following lemmas are given from [15]-[18].

Lemma 2.1 Among all code polynomials in *C*, there is a unique monic generator polynomial $g(x) = g_0 + g_1x + ... + g_{r-1}x^{r-1} + x^r$ with minimal degree $r<n$.

Every code polynomial *c(x)* in *C* is a multiple of *g(x)*, and can be expressed uniquely as $c(x) = m(x)g(x)$, where $m(x) = m_0 + m_1x + ... + m_{k-1}x^{k-1}$ is the message polynomial corresponding to the massage vector $(m_0, m_1, ..., m_{k-1})$.

Lemma 2.2 The generator polynomial *g(x)* of *C* is a factor of $x^n - 1$ in $F_2[x]$.

Let *C* be a cyclic code with the generator polynomial $g(x) = g_0 + g_1x + ... + g_rx^r$, $g_0 \neq 0$ and $c \in C$ be a codeword transmitted through a noisy channel. Moreover, let $r=c+e$ is the received vector where e is an error pattern induced by the noisy channel which is denoted by $e(x) = e_0 + e_1x + ... + e_{n-1}x^{n-1}$ in the polynomial form. The generator matrix of *C* is as follows**:**

$$G=\begin{vmatrix} g_0 & g_1 & \dots & g_r & 0 & \dots & 0 & 0 \\ 0 & g_0 & g_1 & \dots & g_r & \dots & 0 & 0 \\ . & & . & \dots & . & \dots & . & . \\ . & & . & \dots & . & \dots & . & . \\ . & & . & \dots & . & \dots & . & . \\ 0 & \dots & 0 & g_0 & g_1 & \dots & g_r & 0 \\ 0 & 0 & \dots & 0 & g_0 & \dots & g_{r-1} & g_r \end{vmatrix}$$

In fact, each row of G is the right-cyclic shift of the previous row.
Moreover, the parity-check matrix of code C can be represented by the following matrix.

$$H=\begin{vmatrix} h_k & h_{k-1} & \dots & h_0 & 0 & \dots & 0 & 0 \\ 0 & h_k & h_{k-1} & \dots & h_0 & \dots & 0 & 0 \\ . & & . & \dots & . & \dots & . & . \\ . & & . & \dots & . & \dots & . & . \\ . & & . & \dots & . & \dots & . & . \\ 0 & \dots & 0 & h_k & h_{k-1} & \dots & h_0 & 0 \\ 0 & 0 & \dots & 0 & h_k & \dots & h_1 & h_0 \end{vmatrix}$$

where

$h(x)=h_0+h_1x+\dots+h_kx^k$, $k=n-r$, is the parity-check polynomial defined by $h(x)=\dfrac{x^n-1}{g(x)}$.

Now, let $n=8l\pm1$, where l is an arbitrary positive integer and Q be the quadratic residues modulo n, i.e. the set of nonzero squares modulo n.
For example, the set of quadratic residues (QR) modulo 47 is as follows.

Q_{47}={1, 2, 3, 4, 6, 7, 8, 9, 12, 14, 16, 17, 18, 21, 24, 25, 27, 28, 32, 34, 36, 37, 42}

Let C be a [47,24,11] cyclic code with the following generator polynomial:

$$g(x)=\prod_{i\in Q_{47}}(x-w^i)=1+x+x^2+x^3+x^5+x^6+x^7+x^9+x^{10}+x^{12}+x^{13}+x^{14}+x^{18}+x^{19}+x^{23}.$$

where $w=\beta^{\frac{2^{23}-1}{47}}=\beta^{178481}$ is the primitive 47th root of $x^{47}-1$ and β is the primitive element of the finite field $GF(2^{23})$.
The parity-check polynomial of the [47,24,11] QR code is as follows:

$$h(x)=x^{47}-1\Big/g(x)=x^{24}+x^{20}+x^{19}+x^{16}+x^{15}+x^{13}+x^{12}+x^9+x^6+x^4+x.$$

Now, for a received vector $r(x)$, let $s(x)=s_0+s_1x+\dots+s_{n-k-1}x^{n-k-1}=r(x)h(x) \mod x^n-1$ be the syndrome polynomial in $GF(q)[x]/(x^n-1)$ which can be expressed as syndrome vector $s=(s_0,s_1,\dots,s_{n-k-1})=rH^T$, where H^T is the transpose of H. Clearly, if $r=c+e$, c is a codeword, then $s=eH^T$.

3 THE REDUCED LOOKUP TABLE DECODING ALGORITHM

The [47,24,11] *QR* code has the minimum-distance 11, so it can correct all error patterns of weight less than or equal $t = \lfloor (d_{\min}(C) - 1)/2 \rfloor = \lfloor (11-1)/2 \rfloor = 5$ Based on lookup-table decoding, a decoding method was proposed in [12] for decoding the [47,24,11] *QR* in which the total number of syndromes and their corresponding error patterns is $\sum_{i=1}^{5} \binom{47}{i} = 1729647$. On the other hand, the memory-size required to store each syndrome and error pattern is 3 and 6 bytes, respectively. Thus, the total memory size for the full lookup table (FLT) is $1729647 \times (6bytes + 3bytes) \cong 14.85$. However, such a large memory requirement is not efficient in practice and it is very expensive.

Here, we introduce a new decoding algorithm, called reduced lookup table decoding algorithm (RLTDA), which uses the properties of cyclic codes, weight of syndromes, reduced lookup table and the code-rate close to 1/2 to reduce lookup time, number of syndromes and corresponding error patterns. In fact, the reduced lookup table used in RLTDA consists of syndrome patterns and corresponding error patterns with at most two errors in the message block of the codeword. All possible 5 errors may de occurred in [47,24,11] *QR* code are described in Table 1.

Table 1: Possible 5 errors in [47,24,11] *QR* code

t	Block parity-check	Block message
1	5	0
2	0	0,1,2,3,4
3	1	0,1,2,3,4
4	2	0,1,2,3
5	3	0,1,2
6	4	0,1
7	0	5

Theorem 3.1 If *w(s)=0*, then no errors happened in the received word.

Theorem 3.2 Let *e* be the error of weight and syndrome-weight at most 5, i.e. $w(e) \le 5$ and $0 \ne w(s) \le 5$ Then all non-zero elements of *e* are located in the parity-check block.

Theorem 3.3 If $w(S) \ge 6$ and syndrome is in the table then at most 2 errors happened in the message block of the received codeword.

Theorem 3.4 If $w(S) \ge 6$ and syndrome is not in the table, it means there are six possible condition:

- One error happened in the parity-check block and at most four errors in the message block.
- two errors happened in the parity-check block and at most three errors in the message block.
- three errors happened in the parity-check block and at most two errors in the message block.
- four errors happened in the parity-check block and one error in the message block.
- At most four errors happened in the message block.
- five errors happened in the message block.

3.1 Algorithm

S0 Let c be transmitted codeword and $r=c+e$ be the received vector having the syndrome $s = rH^T$ of weight $w(s)$.

S1 Let t=5 and k=24.

S2 If $w(s)=0$, then r is a codeword, so go to the end.

S3 If $0 \neq w(s) \leq t$, then decode r as $c=r-(s<<k)$, and go to the end.

S4 If $w(s)>t$, then search syndrome s in Table Table 3 using BSA (binary search algorithm). If s corresponds to an error vector e_i in the table, then set $c=r-e_i$, and go to the end.

S5 For each s_i in Table 4, set $s^{(i)} = s - s_i$ and $w(s^{(i)})$ be the weight of $s^{(i)}$.

S6 If $w(s^{(i)}) \leq 4$, for some $1 \leq i \leq 300$, then set $c = r - (s^{(i)} << k) - e_i$ and go to the end.

S7 Let r' be the n-k bits left-cyclic-shift of r and $s' = r'H^T$ be the syndrome of r' of weight $w(s')$.

S8 If $w(s') = t$, then set $c' = r' - (s' << k)$, else let $s'^{(i)} = s' - s_i$, then if $3 \leq w(s'^{(i)}) \leq 4$ set $c' = r' - (s'^{(i)} << k) - e_i$, else flip the first bit of the massage part of r and then set r', $s' = r'H^T$, $s'^{(i)} = s' - s_i$ and $c' = r' - (s'^{(i)} << k) - e_i$.

S9 Let c be the n-k bits right-cyclic-shift of c' and go to the end.

S10 End.

As seen in Tables 3,4, RLTDA algorithm only needs $\sum_{i=1}^{2} \binom{24}{i} = 300$ syndromes with their corresponding error patterns. Hence, the memory required to store the syndromes with the error patterns is $300 \times (6bytes + 3bytes) \cong 2.6\ KB$. Moreover, if $N=300$ is the number of all syndromes, then the time complexity of finding a target in the RLTDA algorithm using a binary search is at most $O(\log_2 N) = \log_2 300 = 8$ times. However, if a linear search algorithm is used to find the syndromes, then the overall time complexity of the algorithm will be at most $O(N)=300$ times. Therefore, the average searching speed in RLTDA algorithm can be saved up to $300/8 \cong 37.5 times$.

Table 2. number of syndromes and error patterns in the reduced look up table.

t	Number of syndromes	Number of error patterns
1	24	24
2	276	276
Total	300	300

4 CONCLUSION

In this paper, an efficient look up table is suggested to decode the binary systematic [47,24,11] *QR* code up to five errors. This decoding algorithm is based on the properties of cyclic codes, the weight of syndromes and the reduced lookup table. These facts, of course, lead to a substantial reduction of the size of the standard traditional lookup table. In this method, the well-known binary search algorithm is applied to reduce the searching time. In general, the proposed method can be applied for other well-known cyclic codes with rates close to 1/2.

REFERENCES

[1] E. Prange. Some cyclic error-correcting codes with simple decoding algorithms, Air Force Cambridge Research Center-TN-58-156, Cambridge, Ma(1958).

[2] S. B. Wicker. Error control systems for digital communication and storage, Prentice Hall, 1995.

[3] Y. Chang, T. K. Truong, I. S. Reed, H. Y. Cheng, and C. D. Lee. Algebraic decoding of [71, 36, 11], [79, 40, 15], and [97, 49, 15] quadratic residue codes, IEEE Transaction on Communications, 51 (2003), pp. 1463—1473.

[4] Y. H. Chen, T. K. Truong, Y. Chang, C. D. Lee and S. H. Chen. Algebraic decoding of quadratic residue codes using Berlekamp-massey algorithm, Journal of Information Science and Engineering, 23 (2007), pp. 127--145.

[5] M. Elia. Algebraic decoding of the [23,12,7] Golay codes, IEEE Transaction on Information Theory , 33 (1987), pp. 150--151.

[6] R. He, I. S. Reed, T. K. Truong, and X. Chen. Decoding the [47, 24,11] quadratic residue code, IEEE Transaction on Information Theory, 47 (2001), pp. 1181--1186.

[7] I. S. Reed, X. Yin, and T. K. Truong, Algebraic decoding of the [32,16,8] quadratic residue code, IEEE Transaction on Information Theory, IT-36 (1990), pp. 876—880.

[8] I. S. Reed, T. K. Truong, X. Chen, and X. Yin, The Algebraic decoding of the [41, 21, 9] quadratic residue code, IEEE Transaction on Information Theory, 32 (1986), pp. 561--567.

[9] X. Chen, I.S. Reed, and T.K. Truong. Decoding the [73, 37, 13] quadratic residue code, Proceedings of the IEE, 141 (1994), pp. 253--258.

[10] T. K. Truong, Y. Chang, Y. H. Chen, and C. D. Lee. Algebraic decoding of [103, 52, 19] and [113, 57, 15] quadratic residue codes, IEEE Transaction on Communications, 53 (2005), pp. 749--754.

[11] T. C. Lin, T. K. Truong, H. P. Lee, H. C. Chang. Algebraic decoding of the [41,21,9] quadratic residue code, Information Science, 179 (2009), pp. 3451--3459.

[12] Y. H. Chen, T. K. Troung, C. H. Chien. A lookup table decoding of systematic [47,24,11] quadratic residue code, Information Sciences, 179 (2009), pp. 2470--2477.

[13] T. K. Truong, P. Y. Shih, W. K. Su, C. D. Lee and Y. Chang. Algebraic decoding of the [89,45,17] quadratic residue code, IEEE Transaction on Information Theory, 54 (2008), pp. 5005--5011.

[14] T. C. Lin, H.P. Lee, H. C. Chang, S. I. Chu and T. K. Truong. High speed decoding of the binary [47,24,11] quadratic residue code, Information Sciences, 180 (2010), pp. 4060--4068.

[15] T. C. Lin, H. P. Lee, H. C. Chang, and T. K. Truong. Cyclic weight algorithm of decoding the [47,24,11] quadratic residue code, Information Sciences, 197 (2012), pp. 215-222.

[16] H. P. Lee, H. C. Chang. Decoding of the triple-error-correcting binary quadratic residue codes, Information Sciences, 2 (2014), pp. 7--12.

[17] H. P. Lee, H. C. Chang, T. C. Lin and T. K. Truong. A weight method of decoding the [23,12,7] Golay code using reduced table lookup", International Conference on Communication, Circuits and Systems , ICCCAS (2008), pp. 1--5.

[18] H. P. Lee, H. C. Chang, T. C. Lin and T. K. Truong, A weight method of decoding the binary BCH code, Eighth International Conference on Intelligent Systems Design and Applications, ISDA '08 (2008), pp . 545--549.

Table 3.the error locators of some error patterns required for RLTDA algorithm.

e_i	l_i	e_i	l_i	e_i	l_i	e_i	l_i	e_i	l_i	e_i	l_i	e_i	l_i
e_1	[47]	e_2	[46]	e_3	[45]	e_4	[44]	e_5	[43]	e_6	[42]	e_7	[41]
e_8	[40]	e_9	[39]	e_{10}	[38]	e_{11}	[37]	e_{12}	[36]	e_{13}	[35]	e_{14}	[34]
e_{15}	[33]	e_{16}	[32]	e_{17}	[31]	e_{18}	[30]	e_{19}	[29]	e_{20}	[28]	e_{21}	[27]
e_{22}	[26]	e_{23}	[25]	e_{24}	[24]								
e_{25}	[46, 47]	e_{26}	[45, 47]	e_{27}	[44, 47]	e_{28}	[43, 47]	e_{29}	[42, 47]	e_{30}	[41, 47]	e_{31}	[40, 47]
e_{32}	[39, 47]	e_{33}	[38, 47]	e_{34}	[37, 47]	e_{35}	[36, 47]	e_{36}	[35, 47]	e_{37}	[34, 47]	e_{38}	[33, 47]
e_{39}	[32, 47]	e_{40}	[31, 47]	e_{41}	[30, 47]	e_{42}	[29, 47]	e_{43}	[28, 47]	e_{44}	[27, 47]	e_{45}	[26, 47]
e_{46}	[25, 47]	e_{47}	[24, 47]	e_{48}	[45, 46]	e_{49}	[44, 46]	e_{50}	[43, 46]	e_{51}	[42, 46]	e_{52}	[41, 46]
e_{53}	[40, 46]	e_{54}	[39, 46]	e_{55}	[38, 46]	e_{56}	[37, 46]	e_{57}	[36, 46]	e_{58}	[35, 46]	e_{59}	[34, 46]
e_{60}	[33, 46]	e_{61}	[32, 46]	e_{62}	[31, 46]	e_{63}	[30, 46]	e_{64}	[29, 46]	e_{65}	[28, 46]	e_{66}	[27, 46]
e_{67}	[26, 46]	e_{68}	[25, 46]	e_{69}	[24, 46]	e_{70}	[44, 45]	e_{71}	[43, 45]	e_{72}	[42, 45]	e_{73}	[41, 45]
e_{74}	[40, 45]	e_{75}	[39, 45]	e_{76}	[38, 45]	e_{77}	[37, 45]	e_{78}	[36, 45]	e_{79}	[35, 45]	e_{80}	[34, 45]
e_{81}	[33, 45]	e_{82}	[32, 45]	e_{83}	[31, 45]	e_{84}	[30, 45]	e_{85}	[29, 45]	e_{86}	[28, 45]	e_{87}	[27, 45]
e_{88}	[26, 45]	e_{89}	[25, 45]	e_{90}	[24, 45]	e_{91}	[43, 44]	e_{92}	[42, 44]	e_{93}	[41, 44]	e_{94}	[40, 44]
e_{95}	[39, 44]	e_{96}	[38, 44]	e_{97}	[37, 44]	e_{98}	[36, 44]	e_{99}	[35, 44]	e_{100}	[34, 44]	e_{101}	[33, 44]
e_{102}	[32, 44]	e_{103}	[31, 44]	e_{104}	[30, 44]	e_{105}	[29, 44]	e_{106}	[28, 44]	e_{107}	[27, 44]	e_{108}	[26, 44]
e_{109}	[25, 44]	e_{110}	[24, 44]	e_{111}	[42, 43]	e_{112}	[41, 43]	e_{113}	[40, 43]	e_{114}	[39, 43]	e_{115}	[38, 43]
e_{116}	[37, 43]	e_{117}	[36, 43]	e_{118}	[35, 43]	e_{119}	[34, 43]	e_{120}	[33, 43]	e_{121}	[32, 43]	e_{122}	[31, 43]
e_{123}	[30, 43]	e_{124}	[29, 43]	e_{125}	[28, 43]	e_{126}	[27, 43]	e_{127}	[26, 43]	e_{128}	[25, 43]	e_{129}	[24, 43]
e_{130}	[41, 42]	e_{131}	[40, 42]	e_{132}	[39, 42]	e_{133}	[38, 42]	e_{134}	[37, 42]	e_{135}	[36, 42]	e_{136}	[35, 42]
e_{137}	[34, 42]	e_{138}	[33, 42]	e_{139}	[32, 42]	e_{140}	[31, 42]	e_{141}	[30, 42]	e_{142}	[29, 42]	e_{143}	[28, 42]
e_{144}	[27, 42]	e_{145}	[26, 42]	e_{146}	[25, 42]	e_{147}	[24, 42]	e_{148}	[40, 41]	e_{149}	[39, 41]	e_{150}	[38, 41]
e_{151}	[37, 41]	e_{152}	[36, 41]	e_{153}	[35, 41]	e_{154}	[34, 41]	e_{155}	[33, 41]	e_{156}	[32, 41]	e_{157}	[31, 41]
e_{158}	[30, 41]	e_{159}	[29, 41]	e_{160}	[28, 41]	e_{161}	[27, 41]	e_{162}	[26, 41]	e_{163}	[25, 41]	e_{164}	[24, 41]
e_{165}	[39, 40]	e_{166}	[38, 40]	e_{167}	[37, 40]	e_{168}	[36, 40]	e_{169}	[35, 40]	e_{170}	[34, 40]	e_{171}	[33, 40]
e_{172}	[32, 40]	e_{173}	[31, 40]	e_{174}	[30, 40]	e_{175}	[29, 40]	e_{176}	[28, 40]	e_{177}	[27, 40]	e_{178}	[26, 40]
e_{179}	[25, 40]	e_{180}	[24, 40]	e_{181}	[38, 39]	e_{182}	[37, 39]	e_{183}	[36, 39]	e_{184}	[35, 39]	e_{185}	[34, 39]
e_{186}	[33, 39]	e_{187}	[32, 39]	e_{188}	[31, 39]	e_{189}	[30, 39]	e_{190}	[29, 39]	e_{191}	[28, 39]	e_{192}	[27, 39]
e_{193}	[26, 39]	e_{194}	[25, 39]	e_{195}	[24, 39]	e_{196}	[37, 38]	e_{197}	[36, 38]	e_{198}	[35, 38]	e_{199}	[34, 38]
e_{200}	[33, 38]	e_{201}	[32, 38]	e_{202}	[31, 38]	e_{203}	[30, 38]	e_{204}	[29, 38]	e_{205}	[28, 38]	e_{206}	[27, 38]
e_{207}	[26, 38]	e_{208}	[25, 38]	e_{209}	[24, 38]	e_{210}	[36, 37]	e_{211}	[35, 37]	e_{212}	[34, 37]	e_{213}	[33, 37]
e_{214}	[32, 37]	e_{215}	[31, 37]	e_{216}	[30, 37]	e_{217}	[29, 37]	e_{218}	[28, 37]	e_{219}	[27, 37]	e_{220}	[26, 37]
e_{221}	[25, 37]	e_{222}	[24, 37]	e_{223}	[35, 36]	e_{224}	[34, 36]	e_{225}	[33, 36]	e_{226}	[32, 36]	e_{227}	[31, 36]
e_{228}	[30, 36]	e_{229}	[29, 36]	e_{230}	[28, 36]	e_{231}	[27, 36]	e_{232}	[26, 36]	e_{233}	[25, 36]	e_{234}	[24, 36]
e_{235}	[34, 35]	e_{236}	[33, 35]	e_{237}	[32, 35]	e_{238}	[31, 35]	e_{239}	[30, 35]	e_{240}	[29, 35]	e_{241}	[28, 35]
e_{242}	[27, 35]	e_{243}	[26, 35]	e_{244}	[25, 35]	e_{245}	[24, 35]	e_{246}	[33, 34]	e_{247}	[32, 34]	e_{248}	[31, 34]
e_{249}	[30, 34]	e_{250}	[29, 34]	e_{251}	[28, 34]	e_{252}	[27, 34]	e_{253}	[26, 34]	e_{254}	[25, 34]	e_{255}	[24, 34]
e_{256}	[32, 33]	e_{257}	[31, 33]	e_{258}	[30, 33]	e_{259}	[29, 33]	e_{260}	[28, 33]	e_{261}	[27, 33]	e_{262}	[26, 33]
e_{263}	[25, 33]	e_{264}	[24, 33]	e_{265}	[31, 32]	e_{266}	[30, 32]	e_{267}	[29, 32]	e_{268}	[28, 32]	e_{269}	[27, 32]
e_{270}	[26, 32]	e_{271}	[25, 32]	e_{272}	[24, 32]	e_{273}	[30, 31]	e_{274}	[29, 31]	e_{275}	[28, 31]	e_{276}	[27, 31]
e_{277}	[26, 31]	e_{278}	[25, 31]	e_{279}	[24, 31]	e_{280}	[29, 30]	e_{281}	[28, 30]	e_{282}	[27, 30]	e_{283}	[26, 30]
e_{284}	[25, 30]	e_{285}	[24, 30]	e_{286}	[28, 29]	e_{287}	[27, 29]	e_{288}	[26, 29]	e_{289}	[25, 29]	e_{290}	[24, 29]
e_{291}	[27, 28]	e_{292}	[26, 28]	e_{293}	[25, 28]	e_{294}	[24, 28]	e_{295}	[26, 27]	e_{296}	[25, 27]	e_{297}	[24, 27]
e_{298}	[25, 26]	e_{299}	[24, 26]	e_{300}	[24, 25]								

Table 4. the table of syndromes of the errors given in Table 3

e_i	s_i	e_i	s_i	e_i	$s_i e_i$	s_i	
e_{1}	11101110110111000110001	e_{2}	00110011011001001010011	e_{3}	01100110110010010100110	e_{4}	11001101100100101001100
e_{5}	01110101111110010101001	e_{6}	11101011111100101010010	e_{7}	00111001001110010010101	e_{8}	01110010011100100101010
e_{9}	11100100111001001010100	e_{10}	00100111000101010011001	e_{11}	01001110001010100110010	e_{12}	10011100010101001100100
e_{13}	11010110011101011111001	e_{14}	01000010001101111000011	e_{15}	10000100011011110000110	e_{16}	11100110000000100111101
e_{17}	00100010110110001001011	e_{18}	01000101101100010010110	e_{19}	10001011011000100101100	e_{20}	11111000000110001101001
e_{21}	00011110111011011100011	e_{22}	00111101110110111000110	e_{23}	01111011101101110001100	e_{24}	11110111011011100011000
e_{25}	11011101101110001100010	e_{26}	10001000000101010010111	e_{27}	00100011010011101111101	e_{28}	10011011001001010011000
e_{29}	00000101001011101100011	e_{30}	11010111111001010100100	e_{31}	10011100101011100011011	e_{32}	00001010001110001100101
e_{33}	11001001110010010101000	e_{34}	10100000111101100000011	e_{35}	01110010100010001010101	e_{36}	00111000101010011001000
e_{37}	10101100111010111110010	e_{38}	01101010101100110110111	e_{39}	00001000110111100001100	e_{40}	11001100000001001111010
e_{41}	10101011011011010100111	e_{42}	01100101101111100011101	e_{43}	00010110110001001011000	e_{44}	11110000001100011010010
e_{45}	11010011000001111110111	e_{46}	10010101011010110111101	e_{47}	00011001101100100101001	e_{48}	01010101101011011110101
e_{49}	11111110111101100011111	e_{50}	01000110100111011111010	e_{51}	11011000100101100000001	e_{52}	00001010010111011000110
e_{53}	01000001000101101111001	e_{54}	11010111100000000000111	e_{55}	00010100011100011001010	e_{56}	01111101010011101100001
e_{57}	10101111001100000110111	e_{58}	11100101000100010101010	e_{59}	01110001010100110010000	e_{60}	10110111000010111010101
e_{61}	11010101011001101101110	e_{62}	00010001101111000011000	e_{63}	01110110110101011000101	e_{64}	10111000000001101111111
e_{65}	11001011011111000111010	e_{66}	00101101100010010110000	e_{67}	00001110101111110010101	e_{68}	01001000110100111011111
e_{69}	11000100000010101001011	e_{70}	10101011010110111101010	e_{71}	00010011001100000001111	e_{72}	10001101001110111110100
e_{73}	01011111111100000110011	e_{74}	00010100101110110001100	e_{75}	10000010001011011110010	e_{76}	01000001110111000111111
e_{77}	00101000111000110010100	e_{78}	11111010100111011000010	e_{79}	10110000101111001011111	e_{80}	00100100111111101100101
e_{81}	11100010101001100100000	e_{82}	10000000110010110011011	e_{83}	01000100000100011101101	e_{84}	00100011011110000110000
e_{85}	11101101101010110001010	e_{86}	10011110110100011001111	e_{87}	01111000001001001000101	e_{88}	01011011000100101100000
e_{89}	00011101011111100101010	e_{90}	10010001101001110111110	e_{91}	10111000011010111100101	e_{92}	00100110011000000011110
e_{93}	11110100101010111011001	e_{94}	10111111111000001100110	e_{95}	00101001011101100011000	e_{96}	11101010100001111010101
e_{97}	10000011101110001111110	e_{98}	01010001110001100101000	e_{99}	00011011111001110110101	e_{100}	10001111101001010001111
e_{101}	01001001111111011001010	e_{102}	00101011100100001110001	e_{103}	11101111010010100000111	e_{104}	10001000001000111011010
e_{105}	01000110111100001100000	e_{106}	00110101100010100100101	e_{107}	11010011011111110101111	e_{108}	11110000010010010001010
e_{109}	10110110001001011000000	e_{110}	00111010111111001010100	e_{111}	10011110000010111111011	e_{112}	01001100110000000111100
e_{113}	00000111100010110000011	e_{114}	10010001000111011111101	e_{115}	01010010111011000110000	e_{116}	00111011110100110011011
e_{117}	11101001101011011001101	e_{118}	10100011100011001010000	e_{119}	00110111110011101101010	e_{120}	11110001100101100101111
e_{121}	10010011111110110010100	e_{122}	01010111001000011100010	e_{123}	00110000010010000111111	e_{124}	11111110100110110000101
e_{125}	10001101111000011000000	e_{126}	01101011000101001001010	e_{127}	01001000001000101101111	e_{128}	00001110010011100100101
e_{129}	10000010100101110110001	e_{130}	11010010110010111000111	e_{131}	10011001100000001111000	e_{132}	00001111000101100000110
e_{133}	11001100111001111001011	e_{134}	10100101110110001100000	e_{135}	01110111101001100110110	e_{136}	00111101100001110101011
e_{137}	10101001110001010010001	e_{138}	01101111100111011010100	e_{139}	00001101111100001101111	e_{140}	11001001001010100011001
e_{141}	10101110010000111000100	e_{142}	01100000100100001111110	e_{143}	00010011111010100111011	e_{144}	11110101000111110110001
e_{145}	11010110001010010010100	e_{146}	10010000010001011011110	e_{147}	00011100100111001001010	e_{148}	01001011010010110111111
e_{149}	11011101110111011000001	e_{150}	00011110001011000001100	e_{151}	01110111000100110100111	e_{152}	10100101011011011110001
e_{153}	11101111010011001101100	e_{154}	01111011000011101010110	e_{155}	10111101010101100010011	e_{156}	11011111001110110101000
e_{157}	00011011111000011011110	e_{158}	01111100100010000000011	e_{159}	10110010010110110111001	e_{160}	11000001001000011111100
e_{161}	00100111110101001110110	e_{162}	00000100111000101010011	e_{163}	01000010100011100011001	e_{164}	11001110010101110001101
e_{165}	10010110100101101111110	e_{166}	01010101011001110110011	e_{167}	00111100010110000011000	e_{168}	11101110001001101001110
e_{169}	10100100000001111010011	e_{170}	00110000010001011101001	e_{171}	11110110000111010101100	e_{172}	10010100011100000010111
e_{173}	01010000101010101100001	e_{174}	00110111110000110111100	e_{175}	11111001000100000000110	e_{176}	10001010011010101000011
e_{177}	01101100100111111001001	e_{178}	01001111101010011101100	e_{179}	00001001110001010100110	e_{180}	10000101000111000110010
e_{181}	11000011111100011001101	e_{182}	10101010110011101100110	e_{183}	01111000101100000110000	e_{184}	00110010100100010101101
e_{185}	10100110110100110010111	e_{186}	01100000100010111010010	e_{187}	00000010111001101101001	e_{188}	11000110001111000011111
e_{189}	10100001010101011000010	e_{190}	01101111100001101111000	e_{191}	00011100111111000111101	e_{192}	11111010000010010110111
e_{193}	11011001001111110010010	e_{194}	10011111010100111011000	e_{195}	00010011100010101001100	e_{196}	01101001001111110101011
e_{197}	10111011010000011111101	e_{198}	11110001011000001100000	e_{199}	01100101001000101011010	e_{200}	10100011011110100011111
e_{201}	11000001000101110100100	e_{202}	00000101110011011010010	e_{203}	01100010101001000001111	e_{204}	10101100011101110110101
e_{205}	11011111000011011110000	e_{206}	00111001111110001111010	e_{207}	00011010110011101011111	e_{208}	01011100101000100010101
e_{209}	11010000011110110000001	e_{210}	11010010011111101010110	e_{211}	10011000010111111001011	e_{212}	00001100000111011110001
e_{213}	11001010010001010110100	e_{214}	10101000001010000001111	e_{215}	01101100111100101111001	e_{216}	00001011100110110100100
e_{217}	11000101010010000011110	e_{218}	10110110001100101011011	e_{219}	01010000110001111010001	e_{220}	01110011111100011110100
e_{221}	00110101100111010111110	e_{222}	10111001010001000101010	e_{223}	01001010001000010011101	e_{224}	11011110011000110100111
e_{225}	00011000001110111100010	e_{226}	01111010010101101011001	e_{227}	10111111010001100010111	e_{228}	11011001111001011110010
e_{229}	00010111001101101001000	e_{230}	01100100010011000001101	e_{231}	10000010101110010000111	e_{232}	10100001100011110100010
e_{233}	11100111111000111101000	e_{234}	01101011001110101111100	e_{235}	10010100010000100111010	e_{236}	01010010000110101111111
e_{237}	00110000011101111000100	e_{238}	11110100101011010110010	e_{239}	10010011110001001101111	e_{240}	01011101000101111010101
e_{241}	00101110011011010010000	e_{242}	11001000100110000011010	e_{243}	11101011101011100111111	e_{244}	10101101110000101110101
e_{245}	00100001000110111100001	e_{246}	11000110010110001000101	e_{247}	10100100001101011111110	e_{248}	01100000111011110001000
e_{249}	00000111100001101010101	e_{250}	11001001010101011101111	e_{251}	10111010001011110101010	e_{252}	01011100110110100100000
e_{253}	01111111111011000000101	e_{254}	00111001100000001001111	e_{255}	10110101010110011011011	e_{256}	01100010011011010111011
e_{257}	10100110101101111001101	e_{258}	11000001110111100010000	e_{259}	00001111000011010101010	e_{260}	01111100011101111101111
e_{261}	10011010100000101100101	e_{262}	10111001101101001000000	e_{263}	11111111110110000001010	e_{264}	01110011000000010011110
e_{265}	11000100110110101110110	e_{266}	10100011101100110101011	e_{267}	01101101011000000010001	e_{268}	00011110000110101010100
e_{269}	11111000111011111011110	e_{270}	11011011110110011111011	e_{271}	10011101101101010110001	e_{272}	00010001011011000100101
e_{273}	01100111011010011011101	e_{274}	10101001101110101100111	e_{275}	11011010110000000100010	e_{276}	00111100001101010101000
e_{277}	00011111000000110001101	e_{278}	01011001011011111000111	e_{279}	11010101101101101010011	e_{280}	11001110110100110111010
e_{281}	10111101101010011111111	e_{282}	01011011010111001110101	e_{283}	01111000011010101010000	e_{284}	00111111000001100011010
e_{285}	10110010110111110001110	e_{286}	01110011011110101000101	e_{287}	10010101100011111001111	e_{288}	10110110101110011101010
e_{289}	11110000110101010100000	e_{290}	01111100000011000110100	e_{291}	11100110111101010001010	e_{292}	11000101110000110101111
e_{293}	10000011101011111100101	e_{294}	00001111011101101110001	e_{295}	00100011001101100100101	e_{296}	01100101010110101101111
e_{297}	11101001100000111111011	e_{298}	01000110011011001001010	e_{299}	11001010101101011011110	e_{300}	10001100110110010010100