Modification and Efficient Implementation of the SKHNE Algorithm for Adaptive Data Protection

Dr. MAIRTIN O'DROMA
Department of Electronics and Computer Engineering
University of Limerick
National Technological Park, Limerick
IRELAND

Dr. IVAN GANCHEV
Department of Electronics and Computer Engineering
University of Limerick
National Technological Park, Limerick
IRELAND

Abstract: - A modification and efficient implementation of the SKHNE (Sugiyama, Kasahara, Hirasawa, Namekawa, Euclid) algorithm for a class of BCH codes is proposed. Envisaged as part of an adaptive data protection (ADP) software application it is designed to operate (decode and error correct) for binary "narrow-sense" BCH(8,t) codes providing increasing level of protection against error in accordance with user choices. While such codes cause loss of compression the improvement in error probability is dramatic, e.g. from 10^{-5} to 10^{-41} approximately for BCH(8,10), with a 42% loss of compression. It may function in applications requiring a supplemental inner code or a stand alone data protection shield.

Key-words: - SKHNE algorithm; BCH codes; error protection; archives; CD-ROM; DVD

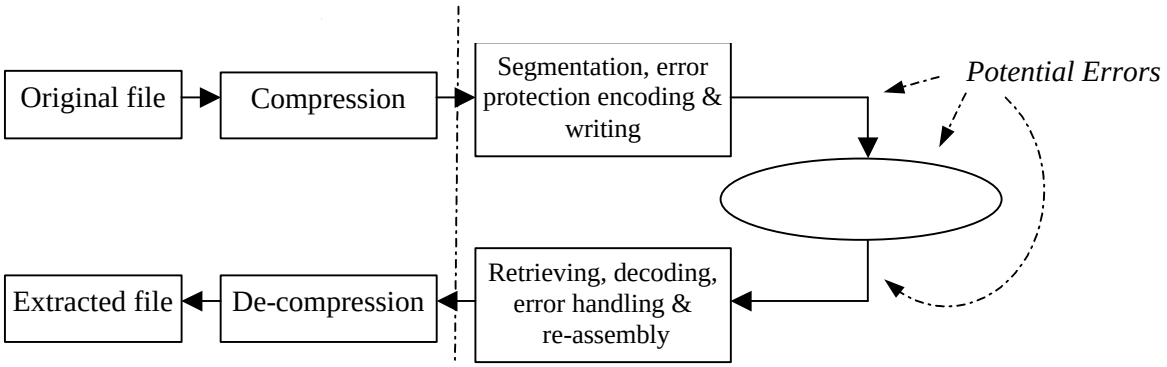
1 Introduction

Coding to protect against errors arising in the electronic archival writing or retrieving process or in the archive itself, fig. 1, is an important field e.g. c.f. [1], [2], [3], [4] and [5]. Some coding schemes are built into the archiving software. Of these some are error detecting only, e.g. the popular CRC-32. Others attempt to recover corrupted data e.g. the ARJ and JAR programs [4], which can be made to repair up to 4 damaged 1kbyte long sections in a 1Mbyte block and 28 kbytes of damaged section(s) on a 1.44 Mbytes volume resp. Other schemes are embedded into the hardware or the firmware. In CD-ROM and DVD firmware implemented (right of 'A', fig.1), powerful, Cross-Interleaved Reed-Solomon Code (CIRC) and Reed-Solomon Product Code (RS-PC) are used resp., e.g. [5], [6], [7] and [8]. The latter can cater for 6.0mm burst error and both add c. 34% and 13% overhead resp., [9]. However regardless of the in-built error protection power in any archive system, requirements will continue to change as a function of service application, user needs and so forth, e.g. [5], [10]. Thus there is room for optional, supplemental, software-based schemes –adaptive data protection (ADP) applications– for multiple error protection, with levels of protection being chosen by the user and traded against increased storage requirements. Such a scheme would be inserted at 'A' in fig. 1. Existing error protection, if present –to the right of 'A'– becomes an outer layer of protection. An ADP has been developed based on Bose-Chaudhuri-Hocquenghem (BCH) codes, [11], [12]. These codes have the attraction of flexibility in the choice of those parameters, which dictate their multiple error-correcting power. Also at block lengths of a few hundred or less, many of them are among the most efficient codes known, [13].

2 BCH Encoding/Decoding Process

2.1 Definition of Terms

Letting $\boldsymbol{G(x)}$ be the characteristic unique generator polynomial for a BCH code. Its polynomial coefficients are required to be elements of a finite Galois field, $GF(\boldsymbol{q})$. The BCH codes are most easily defined in terms of the roots of $\boldsymbol{G(x)}$. Thus a primitive t-error-correcting $BCH(\boldsymbol{m,t})$ code over



*GF(**q**)* of block length $n=(q^m-1)$ has $\alpha^{m_0}, \alpha^{m_0+1}, \ldots, \alpha^{m_0+2t-1}$ as roots of $G(x)$ for any m_0, where α is a primitive element of $GF(q^m)$. Those codes with $m_0=1$ are called "narrow-sense" BCH codes and those defined over *GF(2)* are called binary codes. These only are considered here. The choice of α for $GF(2^m)$ is immaterial, [14]. Of course the encoder and decoder should use the same α .

2.2 BCH Encoding

A systematic *n*-tuple *BCH(**m,t**)* codeword $c(x)$ for the input *k*-tuple message $b(x)$ may be obtained via, e.g. [14]:

$$c(x) = b(x)x^{n-k} + h(x),$$

where $h(x) = \{b(x)\, x^{n-k}\} \text{modulo} \{G(x)\}$, $n = 2^m-1$, and $k=n-\{\text{degree of } G(x)\}$.

The code's minimum Hamming distance (HD) is $d=2t+1$.

2.3 BCH Decoding

The BCH decoding process operates on the minimum distance decoding principle, i.e. choosing as the valid corrected codeword vector, c, that one nearest –HD– to the retrieved data vector, r. It does this by finding that error vector e of minimum Hamming weight such that $c=r \oplus e$ ($\oplus$ denoting modulo-2 sum). Figure 2 shows the block diagram of the binary BCH decoder implemented as a part of the general structure for time domain BCH decoders, [13].

It comprises the following phases:

i) Computing the syndromes and obtaining the syndrome polynomial $S(z)$

The syndromes S_i of r, $=r_j$, $j=0,\ldots,(n-1)$ may be defined, [14], as the values

$$(1) \qquad S_i = \sum_{j=0}^{n-1} r_j \alpha^{ij}, \quad i = 1, 2, \ldots, 2t .$$

If $S_i=0$ for $i=1,\ldots, 2t$, then $c=r$. The syndrome polynomial can be obtained as:

$$(2) \qquad S(z) = S_1 + S_2 z + S_3 z^2 + \ldots + S_{2t} z^{2t-1} = \sum_{i=0}^{2t-1} S_{i+1} z^i .$$

ii) Solving the key equation and finding the error locator polynomial $\Lambda(z)$

If r has n errors in error locations $j_1, j_2, \ldots, j_n$, (with $n \leq 2t$) then its error locator polynomial $L(z)$ and error evaluator polynomial $\Omega(z)$ can be written [14]:

$$(3) \qquad \Lambda(z) = \prod_{k=1}^{n} (1 - \alpha^{j_k} z), \quad \text{and}$$

$$\Omega(z) = \sum_{k=1}^{n} e_{j_k} \alpha^{j_k} \prod_{l=1 \text{and} l \neq k}^{n} (1 - \alpha^{i_l} z).$$

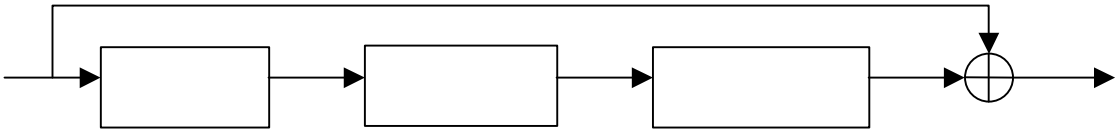
When t or less errors occur in r, then the following key equation holds, [14]:

$$(4) \qquad L(z)S(z) \equiv W(z) \bmod (z^{2t}).$$

iii) Finding error locations and obtaining the error vector e

The reciprocals of the roots of $L(z)$, i.e. $\alpha^{-j_1}, \alpha^{-j_2}, \ldots, \alpha^{-j_n}$ (eq. 3), correspond to the error locations. That is an error is found in position j *iff* $L(\alpha^{-j})=0$. From here e is obtained. The roots themselves may be found by simply checking which elements y of $GF(2^m)$ satisfy the equation $L(y)=0$. If $\alpha^{p(\hat{o})}$ is such a root then the location $j_{\hat{o}}$, $(\hat{o} \leq n)$, of the corresponding error location in r is (counting from the right starting with 0):

$$(5) \qquad j_{\hat{o}} = 2^m - 1 - p(\hat{o})$$



3 Error Correction with the SKHNE Algorithm

3.1 Finding *L(z)*

Berlekamp [15] developed an algorithm to find the minimum-degree ***L(z)***, which satisfies eq.4. Though efficient, it was later improved by Massey [16]. The technique used here adapts and implements a simpler algorithm, reported by Sugiyama, Kasahara, Hirasawa & Namekawa, [17], based on Euclid's algorithm [for finding the greatest common divisor of two polynomials ***a(z)*** and ***b(z)***]. For brevity it is called here the SKHNE algorithm. The useful aspect of the Euclid's algorithm is not in the final answer but in the partial results, as shown in the following. At each iteration ***i*** a set of polynomials $f_i(z)$, $g_i(z)$ and $\varsigma_i(z)$ are generated such that

(6) $$f_i(z)a(z) + g_i(z)b(z) = h_i(z).$$

In noting that eq.6 may also be written

(7) $$g_i(z)b(z) \equiv h_i(z) \bmod a(z)$$

then if *a(z)* is set to z^{2t} and *b(z)* to *S(z)*, this becomes

(8) $$g_i(z)S(z) \equiv h_i(z) \bmod (z^{2t}).$$

At some stage in the iterative process, say stage ***i=n***, when $deg[\varsigma_n(z)]<t$, eq. 8 becomes identical to eq. 4, with $\varsigma_n(z)=\grave{U}(z)$ and $g_n(z)$ being the desired polynomial $\ddot{E}(z)$.

3.3 The SKHNE Steps in 'Finding $\ddot{E}(z)$'

i) Set the initial conditions:
$g_{-1}(z) = 0$, $g_0(z) = 1$,
$h_{-1}(z) = a(z) = z^{2t}$ and
$h_0(z) = b(z) = S(z)$.

ii) Apply Euclid's algorithm. At each iteration ***i*** (***i*** = 1, 2, …) divide $h_{i-2}(z)$ by $h_{i-1}(z)$, obtain the quotient $q_i(z)$, the remainder $h_i(z)$, and construct polynomial $g_i(z)$ via: $g_i(z)=g_{i-2}(z)-q_i(z).g_{i-1}(z)$.

iii) Stop the recursive iterations when $deg[\varsigma_n(z)]<t$, and set $L(z)=g_n(z)$.

Termination (step iii) occurs properly if no more than *t* errors occur in ***r*** because only in this case does eq. 4 hold. Otherwise (i.e. if more than *t* errors occur) an incorrect, though valid, code vector may be produced or the SKHNE algorithm may 'fail'. The latter is detectable. There are two failure modes, [14]: (a) The SKHNE does not terminate properly; this occurs iff $z^t \mid S(z)$. (b) The SKHNE terminates but produces a faulty ***L(z)***; this may happen if ***L(z)*** has ***0*** as a root or it does not split into linear factors.

3.3 The SKHNE Implementation

Table 1 shows the steps in the SKHNE algorithm; it follows a time domain BCH decoder structure [13]. The language used was C++.

4 Conclusion

Addressing the need to provide greater user control over data integrity in traditional storage media, this paper develops a modified SKHNE algorithm, showing how it can be flexibly implemented for binary "narrow-sense" *BCH**(8,t)*** codes such that the user may set the additional level of data protection required especially where degradation of the quality of the storage media is feared.

A reasonable approximation for the error rate, ***AER***, for raw random probability of error ***p***, is

$$AER = \binom{2^m - 1}{t+1} p^{t+1}.$$

For example a *BCH**(8,10)*** code, which will cause a 42% file expansion overhead, yields an AER of 10^{-41} for $p=10^{-5}$.

Table 1: The steps in the SKHNE algorithm

Step	*Action*	*Step*	*Action*
1	Retrieve ***r.*** Calculate S_i (eq. 1). Write *S(z)* as eq.2. If *S(z)=0*, ***r*** is error free: set ***e*** to zero and go to *step 7;* otherwise go to *step2*.	**4**	Check for failure mode (b), ***L(z)*** has ***o*** as a root or does not split into linear factors: if positive then *stop* and issue EM. Otherwise, go to *step 5*.
2	Check $z^t \mid S(z)$: if positive, failure mode (a), *stop* and issue EM; otherwise, go to *step 3*.	**5**	Find: (i) the roots, $a^{p(\hat{o})}$, of ***L(z);*** (ii) error locations $j_{\hat{o}} = 2^m - 1 - p(\hat{o})$
3	Execute 'finding ***L(z)***' algorithm core. If the first remainder $h_i(z)$, which has degree $<t$ is zero then there are more than ***t*** errors in ***r***: s*top* and issue EM. Otherwise go to s*tep 4*.	**6**	Construct error vector ***e*** with 1's in positions $j_{\hat{o}}$ and 0's elsewhere.
		7	Obtain the recovered corrected code vector ***c***, via $e \oplus r = c$. Return to & repeat the process completed.
– *retrieval with another device drive as the errors may not be inherent to the archive*			

[1] X1. *Coding Techniques for digital recorders,*

[2] X2. J. Taylor. *ebook for DVD video and DVD-* McGraw- 1997.

Forum. http://www.dvdforum.org. To date.

[4] X4. R. 76 and JAR 1.02 programs. ARJ.TXT and . To

[5] *Error control systems for* Prentice Hall. 1995.

[6] X6. R.C. Chang and C.B. Shung. A (208,192;8) Reed Solomon decoder for DVD application. *Proc. International Conference* , ICC 98. (ISBN: 0- - 4788 9). Vol. 2. 1998. pp.957-

[

Markarian. Improved decoding technique for the DVD. *Music Technology: The Challenge of Creative DSP*. -17/5.

[8] X8. K. Oh and W. Sung. An efficient Reed- ecoder VLSI with erasure correction. SIPS 97. ISBN 0- -3806 5. 1997. pp. 193-

IEEE Multimedia -Mar. 1999. pp. -84

[10] X10. J.W. Einberger. CD- storage device. *Proc. 9th IEEE Mass Storage Systems*. Boulder, USA. 1988. - 129.

[11] X11. A. Hocquenghem. "Codes correcteurs *Chiffres*, 2, 1959. pp.147 156.

X12. R.C. Bose, and D. K. Ray Chaudhuri. On a class of error correcting binary group codes, *f. Control*, 3, 1960. pp.68 79.

X13. G.C. Clark, Jr. and J. B. Cain. - *Correcting Codes for Digital* Plenum Press. 1981.

X14. O. Pretzel. -*Correcting Codes and* Clarendon Press. 1992.

X15. E. R. Berle amp. On decoding binary -Chaudhuri Hocquenghem codes, *IEEE* , 11. 1965. pp.577-

[16] J.L. Massey. Shift- BCH decoding, *IEEE Trans. Info. Theory* 1969. pp.122-

[17] Y. Sugiyama, M. Kasahara, S. Hirasawa,

equation for decoding Goppa codes, *Inf.* , 27. 1975. pp.87-