Syndrome Encoding and Decoding of BCH Codes in Sublinear Time

Excerpted from Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data

Yevgeniy Dodis[*] Rafail Ostrovsky[†] Leonid Reyzin[‡] Adam Smith[§]

April 19, 2006

We show that the standard decoding algorithm for BCH codes can be modified to run in time polynomial in the length of the syndrome. This works for BCH codes over any field $GF(q)$, which include Hamming codes in the binary case and Reed-Solomon for the case $n = q - 1$. BCH codes are handled in detail in many textbooks (e.g., [vL92]); our presentation here is quite terse. For simplicity, we only discuss primitive, narrow-sense BCH codes here; the discussion extends easily to the general case.

The algorithm discussed here has been revised due to an error pointed out by Ari Trachtenberg. Its implementation is available [HJR].

We'll use a slightly non-standard formulation of BCH codes. Let $n = q^m - 1$ (in the common binary case, $q = 2$). We will work in two finite fields: $GF(q)$ and a larger extension field $\mathcal{F} = GF(q^m)$. BCH codewords, formally defined below, are then vectors in $GF(q)^n$. In most common presentations, one indexes the n positions of these vectors by discrete logarithms of the elements of $\mathcal{F}^*$: position i, for $1 \le i \le n$, corresponds to α^i, where α generates the multiplicative group $\mathcal{F}^*$. However, there is no inherent reason to do so: they can be indexed by elements of $\mathcal{F}$ directly rather than by their discrete logarithms. Thus, we say that a word has value p_x at position x, where $x \in \mathcal{F}^*$. If one ever needs to write down the entire n-character word in an ordered fashion, one can choose arbitrarily a convenient ordering of the elements of $\mathcal{F}$ (e.g., by using some standard binary representation of field elements); for our purposes this is not necessary, as we do not store entire n-bit words explicitly, but rather represent them by their supports: $\mathsf{supp}(v) = \{(x, p_x) \mid p_x \neq 0\}$. Note that for the binary case, we can define $\mathsf{supp}(v) = \{x \mid p_x \neq 0\}$, because p_x can take only two values: 0 or 1.

Our choice of representation will be crucial for efficient decoding: in the more common representation, the last step of the decoding algorithm requires one to find the position i of the error from the field element α^i. However, no efficient algorithms for computing discrete logarithm are known if q^m is large (indeed, a lot of cryptography is based on the assumption that such efficient algorithm does not exist). In our representation, the field element α^i will in fact be the position of the error.

[*] `dodis@cs.nyu.edu`. New York University, Department of Computer Science, 251 Mercer St., New York, NY 10012 USA.

[†] `rafail@cs.ucla.edu`. University of California, Los Angeles, Department of Computer Science, Box 951596, 3732D BH, Los Angeles, CA 90095 USA.

[‡] `reyzin@cs.bu.edu`. Boston University, Department of Computer Science, 111 Cummington St., Boston MA 02215 USA.

[§] `adam.smith@weizmann.ac.il`. Weizmann Institute of Science, Faculty of Mathematics and Computer Science, Rehovot 76100, Israel. Currently supported by the Louis L. and Anita M. Perlman Postdoctoral Fellowship. The research reported here was done while the author was a student at the Computer Science and Artificial Intelligence Laboratory at MIT.

Definition 1. The (narrow-sense, primitive) BCH code of designed distance δ over $GF(q)$ (of length $n \geq \delta$) is given by the set of vectors of the form $\left(c_x\right)_{x\in\mathcal{F}^*}$ such that each c_x is in the smaller field $GF(q)$, and the vector satisfies the constraints $\sum_{x\in\mathcal{F}^*} c_x x^i = 0$, for $i = 1,\ldots,\delta-1$, with arithmetic done in the larger field $\mathcal{F}$.

To explain this definition, let us fix a generator α of the multiplicative group of the large field $\mathcal{F}^*$. For any vector of coefficients $\left(c_x\right)_{x\in\mathcal{F}^*}$, we can define a polynomial

$$c(z) = \sum_{x\in GF(q^m)^*} c_x z^{\mathsf{dlog}(x)}$$

where $\mathsf{dlog}(x)$ is the discrete logarithm of x with respect to α. The conditions of the definition are then equivalent to the requirement (more commonly seen in presentations of BCH codes) that $c(\alpha^i) = 0$ for $i = 1,\ldots,\delta-1$, because $(\alpha^i)^{\mathsf{dlog}(x)} = (\alpha^{\mathsf{dlog}(x)})^i = x^i$.

We can simplify this somewhat. Because the coefficients c_x are in $GF(q)$, they satisfy $c_x^q = c_x$. Using the identity $(x+y)^q = x^q + y^q$, which holds even in the large field $\mathcal{F}$, we have $c(\alpha^i)^q = \sum_{x\neq 0} c_x^q x^{iq} = c(\alpha^{iq})$. Thus, roughly a $1/q$ fraction of the conditions in the definition are redundant: we only need to check that they hold for $i \in \{1,...,\delta-1\}$ such that $q \not| i$.

The syndrome of a word (not necessarily a codeword) $(p_x)_{x\in\mathcal{F}^*} \in GF(q)^n$ with respect to the BCH code above is the vector

$$\mathsf{syn}(p) = p(\alpha^1),\ldots,p(\alpha^{\delta-1}), \quad \text{where} \quad p(\alpha^i) = \sum_{x\in\mathcal{F}^*} p_x x^i.$$

As mentioned above, we do not in fact have to include the values $p(\alpha^i)$ such that $q|i$.

COMPUTING WITH LOW-WEIGHT WORDS. A low-weight word $p \in GF(q)^n$ can be represented either as a long string or, more compactly, as a list of positions where it is nonzero and its values at those points. We call this representation the support list of p and denote it $\mathsf{supp}(p) = \{(x,p_x)\}_{x:p_x\neq 0}$.

Lemma 1. *For a q-ary BCH code C of designed distance δ, one can compute:*

- $\mathsf{syn}(p)$ *from* $\mathsf{supp}(p)$ *in time polynomial in δ, $\log n$, and* $|\mathsf{supp}(p)|$, *and*
- $\mathsf{supp}(p)$ *from* $\mathsf{syn}(p)$ *(when p has weight at most $(\delta-1)/2$), in time polynomial in δ and* $\log n$.

Proof. Recall that $\mathsf{syn}(p) = (p(\alpha),...,p(\alpha^{\delta-1}))$ where $p(\alpha^i) = \sum_{x\neq 0} p_x x^i$. Part (1) is easy, since to compute the syndrome we only need to compute the powers of x. This requires about $\delta \cdot \mathsf{weight}(p)$ multiplications in $\mathcal{F}$. For Part (2), we adapt Berlekamp's BCH decoding algorithm, based on its presentation in [vL92]. Let $M = \{x \in \mathcal{F}^* | p_x \neq 0\}$, and define

$$\sigma(z) \stackrel{\text{def}}{=} \prod_{x\in M}(1-xz) \quad \text{and} \quad \omega(z) \stackrel{\text{def}}{=} \sigma(z)\sum_{x\in M}\frac{p_x xz}{(1-xz)}$$

Since $(1-xz)$ divides $\sigma(z)$ for $x \in M$, we see that $\omega(z)$ is in fact a polynomial of degree at most $|M| = \mathsf{weight}(p) \leq (\delta-1)/2$. The polynomials $\sigma(z)$ and $\omega(z)$ are known as the error locator polynomial and evaluator polynomial, respectively; observe that $\gcd(\sigma(z),\omega(z)) = 1$.

We will in fact work with our polynomials modulo z^δ. In this arithmetic the inverse of $(1-xz)$ is $\sum_{\ell=1}^{\delta}(xz)^{\ell-1}$, that is

$$(1-xz)\sum_{\ell=1}^{\delta}(xz)^{\ell-1} \equiv 1 \mod z^\delta.$$

We are given $p(\alpha^\ell)$ for $\ell = 1, ..., \delta$. Let $S(z) = \sum_{\ell=1}^{\delta-1} p(\alpha^\ell) z^\ell$. Note that $S(z) \equiv \sum_{x\in M} p_x \frac{xz}{(1-xz)} \mod z^\delta$. This implies that

$$S(z)\sigma(z) \equiv \omega(z) \mod z^\delta.$$

The polynomials $\sigma(z)$ and $\omega(z)$ satisfy the following four conditions: they are of degree at most $(\delta-1)/2$ each, they are relatively prime, the constant coefficient of σ is 1, and they satisfy this congruence. In fact, let $w'(z), \sigma'(z)$ be any nonzero solution this congruence, where degrees of $w'(z)$ and $\sigma'(z)$ are at most $(\delta-1)/2$. Then $w'(z)/\sigma'(z) = \omega(z)/\sigma(z)$. (To see why this is so, multiply the initial congruence by $\sigma'()$ to get $\omega(z)\sigma'(z) \equiv \sigma(z)\omega'(z) \mod z^\delta$. Since the both sides of the congruence have degree at most $\delta - 1$, they are in fact equal as polynomials.) Thus, there is at most one solution $\sigma(z), \omega(z)$ satisfying all four conditions, which can be obtained from any $\sigma'(z), \omega'(z)$ by reducing the resulting fraction $\omega'(z)/\sigma'(z)$ to obtain the solution of minimal degree with the constant term of σ equal to 1.

Finally, the roots of $\sigma(z)$ are the points x^{-1} for $x \in M$, and the exact value of p_x can be recovered from $\omega(x^{-1}) = p_x \prod_{y\in M, y\neq x}(1 - yx^{-1})$ (this is only needed for $q > 2$, because for $q = 2$, $p_x = 1$). Note that it is possible that a solution to the congruence will be found even if the input syndrome is not a syndrome of any p with $\mathsf{weight}(p) > (\delta-1)/2$ (it is also possible that a solution to the congruence will not be found at all, or that the resulting $\sigma(z)$ will not split into distinct nonzero roots). Such a solution will not give the correct p. Thus, if there is no guarantee that $\mathsf{weight}(p)$ is actually at most $(\delta-1)/2$, it is necessary to recompute $\mathsf{syn}(p)$ after finding the solution, in order to verify that p is indeed correct.

Representing coefficients of $\sigma'(z)$ and $\omega'(z)$ as unknowns, we see that solving the congruence requires only solving a system of δ linear equations (one for each degree of z, from 0 to $\delta-1$) involving $\delta+1$ variables over $\mathcal{F}$, which can be done in $O(\delta^3)$ operations in $\mathcal{F}$ using, e.g., Gaussian elimination. The reduction of the fraction $\omega'(z)/\sigma'(z)$ requires simply running Euclid's algorithm for finding the g.c.d. of two polynomials of degree less than δ, which takes $O(\delta^2)$ operations in $\mathcal{F}$. Suppose the resulting σ has degree e. Then one can find the roots of σ as follows. First test that σ indeed has e distinct roots by testing that $\sigma(z)|z^{q^m} - z$ (this is a necessary and sufficient condition, because every element of $\mathcal{F}$ is a root of $z^{q^m} - z$ exactly once). This can be done by computing $(z^{q^m} \mod \sigma(z))$ and testing if it equals $z \mod \sigma$; it takes m exponentiations of a polynomial to the power q, i.e., $O((m \log q)e^2)$ operations in $\mathcal{F}$. Then apply an equal-degree-factorization algorithm (e.g., as described in [Sho05]), which also takes $O((m \log q)e^2)$ operations in $\mathcal{F}$. Finally, after taking inverses of the roots of $\mathcal{F}$ and finding p_x (which takes $O(e^2)$ operations in $\mathcal{F}$), recompute $\mathsf{syn}(p)$ to verify that it is equal to the input value.

Because $m \log q = \log(n+1)$ and $e \le (\delta-1)/2$, the total running time is $O(\delta^3 + \delta^2 \log n)$ operations in $\mathcal{F}$; each operation in $\mathcal{F}$ can done in time $O(\log^2 n)$, or faster using advanced techniques.

One can improve this running time substantially. The error locator polynomial $\sigma()$ can be found in $O(\log \delta)$ convolutions (multiplications) of polynomials over $\mathcal{F}$ of degree $(\delta-1)/2$ each [Bla83, Section 11.7] by exploiting the special structure of the system of linear equations being solved. Each convolution can be performed asymptotically in time $O(\delta \log \delta \log\log \delta)$ (see, e.g., [vzGG03]), the total time required to find σ gets reduced to $O(\delta \log^2 \delta \log\log \delta)$ operation in $\mathcal{F}$. This replaces the δ^3 term in the above running time.

While this is asymptotically very good, Euclidean-algorithm-based decoding [SKHN75], which runs in $O(\delta^2)$ operations in $\mathcal{F}$, will find $\sigma(z)$ faster for reasonable values of δ (certainly for $\delta < 1000$). The algorithm finds σ as follows:

```
set $R_{\text{old}}(z) \leftarrow z^{\delta-1}$, $R_{\text{cur}}(z) \leftarrow S(z)/z$, $V_{\text{old}}(z) \leftarrow 0$, $V_{\text{cur}}(z) \leftarrow 1$.
while $\deg(R_{\text{cur}}(z)) \ge (\delta-1)/2$:
    divide $R_{\text{old}}(z)$ by $R_{\text{cur}}(z)$ to get quotient $q(z)$ and remainder $R_{\text{new}}(z)$;
```

```
        set $V_{\text{new}}(z) \leftarrow V_{\text{old}}(z) - q(z)V_{\text{cur}}(z)$;
        set $R_{\text{old}}(z) \leftarrow R_{\text{cur}}(z), R_{\text{cur}}(z) \leftarrow R_{\text{new}}(z), V_{\text{old}}(z) \leftarrow V_{\text{cur}}(z), V_{\text{cur}}(z) \leftarrow V_{\text{new}}(z)$.
    set $c \leftarrow V_{\text{cur}}(0)$; set $\sigma(z) \leftarrow V_{\text{cur}}(z)/c$ and $\omega(z) \leftarrow z \cdot R_{\text{cur}}(z)/c$
```

In the above algorithm, if $c = 0$, then the correct $\sigma(z)$ does not exist, i.e., $\mathsf{weight}(p) > (\delta - 1)/2$. The correctness of this algorithm can be seen by observing that the congruence $S(z)\sigma(z) \equiv \omega(z) \pmod{z^\delta}$ can have z factored out of it (because $S(z)$, $\omega(z)$ and z^δ are all divisible by z) and rewritten as $(S(z)/z)\sigma(z) + u(z)z^{\delta-1} = \omega(z)/z$, for some $u(z)$. The obtained σ is easily shown to be the correct one (if one exists at all) by applying [Sho05, Theorem 18.7] (to use the notation of that theorem, set $n = z^{\delta-1}, y = S(z)/z, t^* = r^* = (\delta-1)/2, r' = \omega(z)/z, s' = u(z), t' = \sigma(z)$).

The root finding of σ can also be sped up. Asymptotically, detecting if a polynomial over $\mathcal{F} = GF(q^m) = GF(n+1)$ of degree e has e distinct roots and finding these roots can be performed in time $O(e^{1.815}(\log n)^{0.407})$ operations in $\mathcal{F}$ using the algorithm of Kaltofen and Shoup [KS95], or in time $O(e^2 + (\log n)e \log e \log\log e)$ operations in $\mathcal{F}$ using the EDF algorithm of Cantor and Zassenhaus[1]. For reasonable values of e, the Cantor-Zassenhaus EDF algorithm with Karatsuba's multiplication algorithm [KO63] for polynomials will be faster, giving root-finding running time of $O(e^2 + e^{\log_2 3}\log n)$ operations in $\mathcal{F}$. Note that if the actual weight e of p is close to the maximum tolerated $(\delta-1)/2$, then finding the roots of σ will actually take longer than finding σ. $\square$

A DUAL VIEW OF THE ALGORITHM. Readers may be used to seeing a different, evaluation-based formulation of BCH codes, in which codewords are generated as follows. Let $\mathcal{F}$ again be an extension of $GF(q)$, and let n be the length of the code (note that $|\mathcal{F}^*|$ is not necessarily equal to n in this formulation). Fix distinct $x_1, x_2, \ldots, x_n \in \mathcal{F}$. For every polynomial c over the large field $\mathcal{F}$ of degree at most $n - \delta$, the vector $(c(x_1), c(x_2), \ldots c(x_n))$ is a codeword if and only if every coordinate of the vector happens to be in the smaller field: $c(x_i) \in GF(q)$ for all i. In particular, when $\mathcal{F} = GF(q)$, then every polynomial leads to a codeword, thus giving Reed-Solomon codes.

The syndrome in this formulation can be computed as follows: given a vector $y = (y_1, y_2, \ldots, y_n)$ find the interpolating polynomial $P = p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \cdots + p_0$ over $\mathcal{F}$ of degree at most $n-1$ such that $P(x_i) = y_i$ for all i. The syndrome is then the negative top $\delta - 1$ coefficients of P: $\mathsf{syn}(y) = (-p_{n-1}, -p_{n-2}, \ldots, -p_{n-(\delta-1)})$. (It is easy to see that this is a syndrome: it is a linear function that is zero exactly on the codewords.)

When $n = |\mathcal{F}| - 1$, we can index the n-component vectors by elements of $\mathcal{F}^*$, writing codewords as $(c(x))_{x\in\mathcal{F}^*}$. In this case, the syndrome of $(y_x)_{x\in\mathcal{F}^*}$ defined as the negative top $\delta - 1$ coefficients of P such that $\forall x \in F^*, P(x) = y_x$ is equal to the syndrome defined following Definition 1 as $\sum_{x\in\mathcal{F}} y_x x^i$ for $i = 1, 2, \ldots, \delta - 1$. [2] Thus, when $n = |\mathcal{F}| - 1$, the codewords obtained via the evaluation-based definition are *identical* to the codewords obtain via Definition 1, because codewords are simply elements with the zero syndrome, and the syndrome maps agree.

This is an example of a remarkable duality between evaluations of polynomials and their coefficients: the syndrome can be viewed either as the evaluation of a polynomial whose coefficients are given by the vector, or as the coefficients of the polynomial whose evaluations are given by a vector.

[1]See [Sho05, Section 21.3], and substitute the most efficient known polynomial arithmetic. For example, the procedures described in [vzGG03] take time $O(e \log e \log\log e)$ instead of time $O(e^2)$ to perform modular arithmetic operations with degree-e polynomials.

[2] This statement can be shown as follows: because both maps are linear, it is sufficient to prove that they agree on a vector $(y_x)_{x\in\mathcal{F}^*}$ such that $y_a = 1$ for some $a \in \mathcal{F}^*$ and $y_x = 0$ for $x \neq a$. For such a vector, $\sum_{x\in\mathcal{F}} y_x x^i = a^i$. On the other hand, the interpolating polynomial $P(x)$ such that $P(x) = y_x$ is $-ax^{n-1} - a^2x^{n-2} - \cdots - a^{n-1}x - 1$ (indeed, $P(a) = -n = 1$; furthermore, multiplying $P(x)$ by $x - a$ gives $a(x^n - 1)$, which is zero on all of $\mathcal{F}^*$; hence $P(x)$ is zero for every $x \neq a$).

The syndrome decoding algorithm above has a natural interpretation in the evaluation-based view. Our presentation is an adaptation of Welch-Berlekamp decoding as presented in, e.g., [Sud01, Chapter 10].

Suppose $n = |F| - 1$ and $x_1, ..., x_n$ are the non-zero elements of the field. Let $y = (y_1, y_2, \ldots, y_n)$ be a vector. We are given its syndrome $\mathsf{syn}(y) = (-p_{n-1}, -p_{n-2}, \ldots, -p_{n-(\delta-1)})$, where $p_{n-1}, \ldots, p_{n-(\delta-1)}$ are the top coefficients of the interpolating polynomial P. Knowing only $\mathsf{syn}(y)$, we need to find at most $(\delta - 1)/2$ locations x_i such that correcting all the corresponding y_i will result in a codeword. Suppose that codeword is given by a degree-$(n-\delta)$ polynomial c. Note that c agrees with P on all but the error locations. Let $\rho(z)$ be the polynomial of degree at most $(\delta - 1)/2$ whose roots are exactly the error locations. (Note that $\sigma(z)$ from the decoding algorithm above is the same $\rho(z)$ but with coefficients in reverse order, because the roots of σ are the inverses of the roots of ρ.) Then $\rho(z) \cdot P(z) = \rho(z) \cdot c(z)$ for $z = x_1, x_2, \ldots, x_n$. Since $x_1, .., x_n$ are all the nonzero field elements, $\prod_{i=1}^{n}(z - x_i) = z^n - 1$. Thus,

$$\rho(z) \cdot c(z) \quad = \quad \rho(z) \cdot P(z) \bmod \prod_{i=1}^{n}(z - x_i) \quad = \quad \rho(z) \cdot P(z) \bmod (z^n - 1) \,.$$

If we write the left-hand side as $\alpha_{n-1}x^{n-1} + \alpha_{n-2}x^{n-2} + \cdots + \alpha_0$, then the above equation implies that $\alpha_{n-1} = \cdots = \alpha_{n-(\delta-1)/2} = 0$ (because the degree if $\rho(z) \cdot c(z)$ is at most $n - (\delta + 1)/2$). Because $\alpha_{n-1}, \ldots, \alpha_{n-(\delta-1)/2}$ depend on the coefficients of ρ as well as on $p_{n-1}, \ldots, p_{n-(\delta-1)}$, but not on lower coefficients of P, we obtain a system of $(\delta - 1)/2$ equations for $(\delta - 1)/2$ unknown coefficients of ρ. A careful examination shows that it is essentially the same system as we had for $\sigma(z)$ in the algorithm above. The lowest-degree solution to this system is indeed the correct ρ, by the same argument which was used to prove the correctness of σ in Lemma 1. The roots of ρ are the error-locations. For $q > 2$, the actual corrections that are needed at the error locations (in other words, the light vector corresponding to the given syndrome) can then be recovered by solving the linear system of equations implied by the value of the syndrome.

Acknowledgments

We thank Ari Trachtenberg for finding an error in the preliminary version of the algorithms described here; Ronny Roth for discussions about efficient BCH decoding; and Kevin Harmon for his implementation work.

The work of the Y.D. was partly funded by the National Science Foundation under CAREER Award No. CCR-0133806 and Trusted Computing Grant No. CCR-0311095, and by the New York University Research Challenge Fund 25-74100-N5237. The work of the L.R. was partly funded by the National Science Foundation under Grant No. CCR-0311485. The work of the A.S. was partly funded by US A.R.O. grant DAAD19-00-1-0177 and by a Microsoft Fellowship.

References

[Bie05] Juergen Bierbrauer. *Introduction to Coding Theory*. Chapman & Hall/CRC, 2005.

[Bla83] Richard E. Blahut. *Theory and practice of error control codes*. Addison Wesley Longman, Reading, MA, 1983. 512 p.

[HJR] Kevin Harmon, Soren Johnson, and Leonid Reyzin. An implementation of syndrome encoding and decoding for binary BCH codes, secure sketches and fuzzy extractors. Available at `http://www.cs.bu.edu/~reyzin/code/fuzzy.html`.

[KO63] A.A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.

[KS95] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 398–406, Las Vegas, Nevada, 29 May–1 June 1995.

[Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005. Available from `http://shoup.net`.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control*, 27(1):87–99, 1975.

[Sud01] Madhu Sudan. Lecture notes for an algorithmic introduction to coding theory. Course taught at MIT, December 2001.

[vL92] J.H. van Lint. *Introduction to Coding Theory*. Springer-Verlag, 1992.

[vzGG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.