- RS codes are very widely used in:
(1) mass storage systems,
eg. CD, DVD, distributed online storage.

(2) bar codes

(3) deep space & satellite communications.

Reed-Solomon Code

- View the message as a polynomial over a finite field.
Send the evaluations over the channel.

Encoding: (1) Break the N-bit message into k blocks each of size m-bits.
View these blocks as elements

$d_0, d_1, \ldots, d_{k-1}$ in the field $\mathbb{F}_{2^b}$.

(2) Define $P(x) := d_0 + d_1 x + \ldots + d_{k-1} x^{k-1}$
$\in \mathbb{F}_{2^b}[x]$.

(3) Pick n distinct points $e_0, \ldots, e_{n-1} \in \mathbb{F}_{2^b}$.
Send the code $(c_0, c_1, \ldots, c_{n-1}) :=$
$(P(e_0), P(e_1), \ldots, P(e_{n-1}))$.

▷ The encoding is a linear map from $\{0,1\}^N = (\mathbb{F}_{2^b})^k$ to $(\mathbb{F}_{2^b})^n = \{0,1\}^{bn}$.

▷ It can be computed in $\tilde{O}(nb)$ time.

– The code $\bar{c} := (c_0, \ldots, c_{n-1})$ gets transmitted over the erroneous channel.

– If there are no errors, then Bob can interpolate P from $\bar{c}$, assuming $2^b \geq n \geq k$.

Decoding RS

- How does Bob decode m from a corrupted version $\bar{c}'$ of $\bar{c}$?

- Let there be t errors: Say, the values $P(e_{i_1}), \cdots, P(e_{i_t})$ are wrong.

- (Peterson 1960) The main idea is to consider the error locator polynomial $Q(x) := \prod_{j \in [t]} (x - e_{i_j})$.

$\Rightarrow (c_j - c_j') \cdot Q(e_j) = 0 \quad , \forall 0 \leq j \leq n-1.$

$\Rightarrow P(e_j) \cdot Q(e_j) = c_j' \cdot Q(e_j)$

$\Rightarrow R(e_j) = c_j' \cdot Q(e_j)$

where, $R(x) := P \cdot Q \in \mathbb{F}_{2^b}[x]$.

- We do not know Q & R.

- But, we do know their degree bounds : $\deg R = k-1+t$ &
$\deg Q = t$.

$\Rightarrow$ The #unknowns is $(k-1+t)+1+t$
$= k+2t$.

▷ The linear system :
$R(e_j) = c_j' \cdot Q(e_j)$, $\forall 0 \le j \le n-1$,
would give us $R(x), Q(x)$ if
$n \geq k+2t$.

▷ The original message is $P(x) := R/Q$.

Correctness :

- It is clear that decoding works, as long as $2^b \geq n \geq k+2t$.

- Time taken is $\tilde{O}(n^w \cdot b)$.

Distance

- Let us fix the parameters:
$$b = \lg N, \quad k = \frac{N}{\lg N}, \quad n = N.$$

- Then, RS decoder works when
$$t \leq \frac{n-k}{2} = \frac{N}{2} \cdot \left(1 - \frac{1}{\lg N}\right).$$

▷ RS code is of <u>length</u> $N \cdot \lg N$ & corrects up to $\frac{N}{2} \cdot \left(1 - \frac{1}{\lg N}\right)$ <u>errors</u>.

around 50% correction

- $(2t+1)$ is called the <u>distance</u> of the code.

- Intuitively, it is the <u>minimum Hamming distance</u> between any two distinct codewords!

Crossing the 50% barrier

- When the error bound $t \geq N/2$, then there are many messages corresponding to a corrupted codeword.

- Could we find all of them?

- (Madhu Sudan, 1995) found an efficient way to find them —

List Decoding.

- Consider the scenario:

$$d_0, \ldots, d_{k-1} \xrightarrow{RS} \underset{\| \quad \|}{c_0, \ldots, c_{n-1}} \xrightarrow{\text{channel}} c_0', \ldots, c_{n-1}'$$
$$P(e_0) \quad P(e_{n-1})$$

($\geq T$ of the c_i's are correct)

- Now consider a bivariate "error locator" polynomial $Q(x, y)$ of degree D_x & D_y s.t.

$$Q(e_j, c_j') = 0, \quad \forall\, 0 \le j \le n-1.$$

[If $(1+D_x)(1+D_y) \geq n$ then such a nonzero Q exists, and can be computed by linear algebra.]

- Consider $R(x) := Q(x, P(x))$.
  It has deg $\le D_x + (k-1)\cdot D_y$.
  We know that $R(e_j) = 0$ for T many j's in $[0, n-1]$.

$\Rightarrow$ If $T > D_x + (k-1)D_y$, then $R(x) = 0$,
hence, $(y - P(x)) \mid Q(x,y)$.

Lemma: If $n < (1+D_x)(1+D_y)$ & $D_x + (k-1)D_y < T$, then a curve Q fitting $\{(e_j, c_j') \mid j\}$ has $(y - P(x))$ as a factor.

- Finally, the decoding algorithm is:

1) Fix the parameters:
$D_x = \sqrt{nk}$, $D_y = \sqrt{n/k}$ & $T = 2\sqrt{nk}$.

2) Compute $Q(x,y)$ with degree D_x, D_y s.t. $\forall\, 0 \le j \le n-1 : Q(e_j, c_j') = 0$.

3) Factor $Q(x,y)$ & collect its factors of the form $y - f(x)$ with $\deg f \le k-1$.
[They can be at most D_y many.]

4) Output the list of such $\{f\}$.

▷ This list-decoding algorithm is in randomized poly-time.
It works up to $(n - 2\sqrt{nk})$ many bit errors!

– Next, we'll do bivariate poly. factoring.