### Viewing Messages as Polynomials

A \((n, k, n-k+1)\) code:

- Consider the polynomial of degree \(k-1\)
  \[ p(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0 \]
- **Message**: \((a_{k-1}, \ldots, a_1, a_0)\)
- **Codeword**: \((p(1), p(2), \ldots, p(n))\)

To keep the \(p(i)\) fixed size, we use \(a_i \in GF(p)\)

**Unisolvence Theorem**: Any subset of size \(k\) of \((p(1), p(2), \ldots, p(n))\) is enough to (uniquely) reconstruct \(p(x)\) using polynomial interpolation, e.g., LaGrange's Formula.

### Polynomial-Based Code

\(A (n, k, 2s +1)\) code:

- \(\begin{array}{c|c|c}
  \hline
  & k & 2s \\
  \hline
  n & \hline
  \end{array} \)

Can detect \(2s\) errors

Can correct \(s\) errors

Generally can correct \(\alpha\) erasures and \(\beta\) errors if \(\alpha + 2\beta \leq 2s\)

### Correcting Errors

**Correcting \(s\) errors**:

1. Find \(k + s\) symbols that agree on a polynomial \(p(x)\). These must exist since originally \(k + 2s\) symbols agreed and only \(s\) are in error
2. There are no \(k + s\) symbols that agree on the wrong polynomial \(p'(x)\)
   - Any subset of \(k\) symbols will define \(p'(x)\)
   - Since at most \(s\) out of the \(k+s\) symbols are in error, \(p'(x) = p(x)\)
A Systematic Code

Systematic polynomial-based code
\[ p(x) = a_{k-1}x^{k-1} + \ldots + a_{1}x + a_{0} \]

Message: \((a_{k-1}, \ldots, a_{1}, a_{0})\)
Codeword: \((a_{k-1}, \ldots, a_{1}, a_{0}, p(1), p(2), \ldots, p(2s))\)

This has the advantage that if we know there are no errors, it is trivial to decode.
This will allow us to use the "Parity Check" ideas from linear codes (i.e., \(Hc^T = 0\)) to quickly test for errors.

Reed-Solomon Codes in the Real World

(204,188,17)\(_{256}\): ITU J.83(A)
(128,122,7)\(_{256}\): ITU J.83(B)
(255,223,33)\(_{256}\): Common in Practice
- Note that they are all byte based (i.e., symbols are from \(GF(2^8)\)).

Decoding rate on 600MHz Pentium (approx.):
- (255,251) = 45Mbps
- (255,223) = 4Mbps

Dozens of companies sell hardware cores that operate 10x faster (or more)
- (204,188) = 320Mbps (Altera decoder)

Applications of Reed-Solomon Codes

- Storage: CDs, DVDs, "hard drives",
- Wireless: Cell phones, wireless links
- Satellite and Space: TV, Mars rover, ...
- Digital Television: DVD, MPEG2 layover
- High Speed Modems: ADSL, DSL, ..

Good at handling burst errors.
Other codes are better for random errors.
- e.g., Gallager codes, Turbo codes

RS and “burst” errors

Let’s compare to Hamming Codes (which are "optimal").

<table>
<thead>
<tr>
<th>Code</th>
<th>Code Bits</th>
<th>Check Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS (255, 253, 3)(_{256})</td>
<td>2040</td>
<td>16</td>
</tr>
<tr>
<td>Hamming (2^{11}-1, 2^{11}-11-1, 3)(_{2})</td>
<td>2047</td>
<td>11</td>
</tr>
</tbody>
</table>

They can both correct 1 error, but not 2 random errors.
- The Hamming code does this with fewer check bits
However, RS can fix 8 contiguous bit errors in one byte
- Much better than lower bound for 8 arbitrary errors

\[ \log\left(1 + \binom{n}{1} + \ldots + \binom{n}{8}\right) > 8\log(n-7) \approx 88 \text{ check bits} \]
Galois Fields

The polynomials
\[ \mathbb{Z}_p[x] \mod p(x) \] of degree \( n-1 \)
where
- \( p(x) \in \mathbb{Z}_p[x] \),
- \( p(x) \) is irreducible, and
- \( \deg(p(x)) = n \)
form a finite field. Such a field has \( p^n \) elements. These fields are called Galois Fields or \( \mathbb{GF}(p^n) \).
The special case \( n = 1 \) reduces to the fields \( \mathbb{Z}_p \).
The multiplicative group of \( \mathbb{GF}(p^n)/(0) \) is cyclic, i.e., contain a generator element. (This will be important later).

\[ \mathbb{GF}(2^n) \]

Hugely practical!
The coefficients are bits \( \{0,1\} \).
For example, the elements of \( \mathbb{GF}(2^n) \) can be represented as a byte, one bit for each term, and \( \mathbb{GF}(2^{64}) \) as a 64-bit word.
- e.g., \( x^6 + x^4 + x + 1 = 01010011 \)
How do we do addition?

**Addition** over \( \mathbb{Z}_2 \) corresponds to xor.
- Just take the xor of the bit-strings (bytes or words in practice). This is dirt cheap.

Multiplication over \( \mathbb{GF}(2^n) \)

If \( n \) is small enough can use a table of all combinations.
The size will be \( 2^n \times 2^n \) (e.g., 64K for \( \mathbb{GF}(2^8) \)).
Otherwise, use standard shift and add (xor)

**Note:** dividing through by the irreducible polynomial on an overflow by 1 term is simply a test and an xor.

- e.g., \( 0111 \mod 1001 = 0111 \)
  \[ 1011 \mod 1001 = 1011 \text{ xor } 1001 = 0010 \]
  \^ just look at this bit for \( \mathbb{GF}(2^3) \)

Multiplication over \( \mathbb{GF}(2^8) \)

```c
unsigned char mult(unsigned char a, unsigned char b) {
    int p = a; unsigned char r = 0; /* result */
    while(b) {
        if (b & 1) r = r ^ p; b = b >> 1;
        p = p << 1; if (p & 0x100) /* p has x^8 term */
             /* 0x11B = 100011011 = x^8+x^4+x^3+x+1*/
             p = p ^ 0x11B;
    }
    return r;
}
```
Finding inverses over GF($2^n$)

Again, if $n$ is small just store in a table.
- Table size is just $2^n$.
For larger $n$, use long division algorithm.
- This is again easy to do with shift and xors.

Galois Field

GF($2^3$) with irreducible polynomial: $x^3 + x + 1$
$\alpha = x$ is a generator

| $\alpha$ | $x$ | $010$ | $2$ |
| $\alpha^2$ | $x^2$ | $100$ | $3$ |
| $\alpha^3$ | $x + 1$ | $011$ | $4$ |
| $\alpha^4$ | $x^2 + x$ | $110$ | $5$ |
| $\alpha^5$ | $x^2 + x + 1$ | $011$ | $6$ |
| $\alpha^6$ | $x^2 + 1$ | $101$ | $7$ |

Will use this as an example.

Discrete Fourier Transform (DFT)

Another View of polynomial-based codes
$\alpha$ is a primitive $n$th root of unity ($\alpha^n = 1$) - a generator

$$T = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-2} & \alpha^{n-3} & \ldots & \alpha^{n-2(n-1)} \end{pmatrix} \begin{pmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} m_0 \\ m_{k-1} \\ \vdots \\ m_1 \end{pmatrix}$$

Evaluate polynomial $m_k \alpha^{k-1} \ldots + m_1 \alpha + m_0$ at $n$ distinct roots of unity, $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$
Inverse DFT: $m = T^{-1}c$

DFT Example

$\alpha = x$ is 7th root of unity in GF($2^3$)/$x^3 + x + 1$
(i.e., multiplicative group, which excludes additive inverse)
Recall $\alpha = "2", \alpha^2 = "3", \ldots, \alpha^7 = "1"$

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^5 & \alpha^3 \\ 1 & \alpha^3 & \alpha^6 & \alpha & \alpha^5 & \alpha^3 & \alpha^2 \\ 1 & \alpha^4 & \alpha^2 & \alpha^6 & \alpha^3 & \alpha^5 & \alpha \\ 1 & \alpha^5 & \alpha^3 & \alpha^2 & \alpha^6 & \alpha^5 & \alpha^4 \\ 1 & \alpha^6 & \alpha^5 & \alpha^3 & \alpha^2 & \alpha^6 & \alpha^4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 & 3 & 3 & 3 \\ 1 & 4 & 4 & 4 & 4 & 4 & 4 \\ 1 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 6 & 6 & 6 & 6 & 6 & 6 \\ 1 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

Should be clear that $c = T \cdot (m_0, m_1, \ldots, m_{k-1}, 0, \ldots)^T$ is the same as evaluating $p(x) = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1}$ at $n$ points.
Decoding

Why is it hard?

Brute Force: try \( k+2s \) choose \( k+s \) possibilities and solve for each.

Efficient Decoding

I don't plan to go into the Reed-Solomon decoding algorithm, other than to mention the steps.

- Syndrome Calculator
- Error Polynomial
- Berlekamp-Massy
- Error Locations
- Chien Search
- Error Magnitudes
- Forney Algorithm
- Error Corrector

This is the hard part. CD players use this algorithm. (Can also use Euclid's algorithm.)