Communicating with Errors

Someone sends you a message:

> *"'As mmbrof teGreek commniand art of n oft oranzins thsis hihly offesive."*

As you can see, parts of the message have been lost.

Communicating with Errors

Someone sends you a message:

> *"'As mmbrof teGreek commniand art of n oft oranzins thsis hihly offesive."*

As you can see, parts of the message have been lost.

How can we transmit messages so that the receiver can *recover* the original message if there are *errors*?

Communicating with Errors

Someone sends you a message:

> *“’As mmbrof teGreek commniand art of n oft oranzins thsis hihly offesive.”*

As you can see, parts of the message have been lost.

How can we transmit messages so that the receiver can *recover* the original message if there are *errors*?

Today: Use polynomials to share secrets and correct errors.

Review of Polynomials

- "$d+1$ distinct points uniquely determine a degree $\leq d$ polynomial."

Review of Polynomials

- "$d + 1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d + 1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).

Review of Polynomials

- "$d+1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d+1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).
- The results about polynomials hold over *fields*.

Review of Polynomials

- "$d + 1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d + 1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).
- The results about polynomials hold over *fields*.

Why do we use finite fields such as $\mathbb{Z}/p\mathbb{Z}$ (p prime)?

Review of Polynomials

- "$d+1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d+1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).
- The results about polynomials hold over *fields*.

Why do we use finite fields such as $\mathbb{Z}/p\mathbb{Z}$ (p prime)?

- Computations are fast.

Review of Polynomials

- "$d + 1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d + 1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).
- The results about polynomials hold over *fields*.

Why do we use finite fields such as $\mathbb{Z}/p\mathbb{Z}$ (p prime)?

- Computations are fast.
- Computations are *precise*; no need for floating point arithmetic.

Review of Polynomials

- "$d+1$ distinct points uniquely determine a degree $\leq d$ polynomial."
- From the $d+1$ points we can find an *interpolating polynomial* via Lagrange interpolation (or linear algebra).
- The results about polynomials hold over *fields*.

Why do we use finite fields such as $\mathbb{Z}/p\mathbb{Z}$ (p prime)?

- Computations are fast.
- Computations are *precise*; no need for floating point arithmetic.
- As a result, finite fields are *reliable*.

Nuclear Bombs

Think about the password for America's nuclear bombs.

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

For safety, we want to require k government officials to agree before the nuclear bomb password is revealed.

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

For safety, we want to require k government officials to agree before the nuclear bomb password is revealed.

- That is, if k government officials come together, they can access the password.

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

For safety, we want to require k government officials to agree before the nuclear bomb password is revealed.

- That is, if k government officials come together, they can access the password.
- But if $k - 1$ or fewer officials come together, they cannot access the password.

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

For safety, we want to require k government officials to agree before the nuclear bomb password is revealed.

- That is, if k government officials come together, they can access the password.
- But if $k - 1$ or fewer officials come together, they cannot access the password.

In fact, we will design something *stronger*.

Nuclear Bombs

Think about the password for America's nuclear bombs.

- "No one man should have all that power." – Kanye West

For safety, we want to require k government officials to agree before the nuclear bomb password is revealed.

- That is, if k government officials come together, they can access the password.
- But if $k - 1$ or fewer officials come together, they cannot access the password.

In fact, we will design something *stronger*.

- If $k - 1$ officials come together, they know *nothing* about the password.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

Shamir's Secret Sharing Scheme

Work in $\mathsf{GF}(p)$.

1. Encode the secret s as a_0.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$.

Shamir's Secret Sharing Scheme

Work in $\mathsf{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \dots, a_{k-1}$ randomly in $\{0, 1, \dots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Shamir's Secret Sharing Scheme

Work in $\mathsf{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a-1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P. Then evaluate $P(0)$.

Shamir's Secret Sharing Scheme

Work in $\mathsf{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P. Then evaluate $P(0)$.

- k people know the secret.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \dots, a_{k-1}$ randomly in $\{0, 1, \dots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \dots + a-1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P. Then evaluate $P(0)$.

- k people know the secret.

No Information: If $k-1$ officials come together, there are p possible polynomials that go through the $k-1$ shares.

Shamir's Secret Sharing Scheme

Work in $\mathsf{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P. Then evaluate $P(0)$.

- k people know the secret.

No Information: If $k-1$ officials come together, there are p possible polynomials that go through the $k-1$ shares.

- But this is the same as number of possible secrets.

Shamir's Secret Sharing Scheme

Work in $\mathrm{GF}(p)$.

1. Encode the secret s as a_0.
2. Pick $a_1, \ldots, a_{k-1}$ randomly in $\{0, 1, \ldots, p-1\}$. This defines a polynomial $P(x) := a_{k-1}x^{k-1} + \cdots + a - 1x + a_0$.
3. For the ith government official, give him/her the share $(i, P(i))$.

Correctness: If any k officials come together, they can interpolate to find the polynomial P. Then evaluate $P(0)$.

- k people know the secret.

No Information: If $k-1$ officials come together, there are p possible polynomials that go through the $k-1$ shares.

- But this is the same as number of possible secrets.
- The $k-1$ officials discover nothing new.

Implementation of Secret Sharing

How large must the prime p be?

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.
- Larger than the secret.

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.
- Larger than the secret.

If the secret s has n bits, then the secret is $O(2^n)$.

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.
- Larger than the secret.

If the secret s has n bits, then the secret is $O(2^n)$. So we need $p > 2^n$.

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.
- Larger than the secret.

If the secret s has n bits, then the secret is $O(2^n)$. So we need $p > 2^n$.

The arithmetic is done with $\log p = O(n)$ bit numbers.

Implementation of Secret Sharing

How large must the prime p be?

- Larger than the number of people involved.
- Larger than the secret.

If the secret s has n bits, then the secret is $O(2^n)$. So we need $p > 2^n$.

The arithmetic is done with $\log p = O(n)$ bit numbers.

The runtime is a polynomial in the number of bits of the secret and the number of people, i.e., the scheme is *efficient*.

Sending Packets

You want to send a long message.

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.
- So say you want to send n packets, $m_0, m_1, \ldots, m_{n-1}$.

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.
- So say you want to send n packets, $m_0, m_1, \ldots, m_{n-1}$.
- In information theory, we say that you send the packets across a **channel**.

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.
- So say you want to send n packets, $m_0, m_1, \ldots, m_{n-1}$.
- In information theory, we say that you send the packets across a **channel**.
- What happens if the channel is *imperfect*?

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.
- So say you want to send n packets, $m_0, m_1, \ldots, m_{n-1}$.
- In information theory, we say that you send the packets across a **channel**.
- What happens if the channel is *imperfect*?
- First model: when you use the channel, it can drop any k of your packets.

Sending Packets

You want to send a long message.

- In Internet communication, the message is divided up into smaller chunks called **packets**.
- So say you want to send n packets, $m_0, m_1, \ldots, m_{n-1}$.
- In information theory, we say that you send the packets across a **channel**.
- What happens if the channel is *imperfect*?
- First model: when you use the channel, it can drop any k of your packets.

Can we still communicate our message?

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$?

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$. Remember: n points determine a degree $\leq n-1$ polynomial.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$. Remember: n points determine a degree $\leq n-1$ polynomial.

Then, send $(0, P(0)), (1, P(1)), \ldots, (n+k-1, P(n+k-1))$ across the channel.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$. Remember: n points determine a degree $\leq n-1$ polynomial.

Then, send $(0, P(0)), (1, P(1)), \ldots, (n+k-1, P(n+k-1))$ across the channel.

- Note: If the channel drops packets, the receiver *knows* which packets are dropped.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$. Remember: n points determine a degree $\leq n-1$ polynomial.

Then, send $(0, P(0)), (1, P(1)), \ldots, (n+k-1, P(n+k-1))$ across the channel.

- Note: If the channel drops packets, the receiver *knows* which packets are dropped.

Property of polynomials: If we receive *any* n packets, then we can interpolate to recover the message.

Reed-Solomon Codes

Encode the packets $m_0, m_1, \ldots, m_{n-1}$ as values of a polynomial $P(0), P(1), \ldots, P(n-1)$.

What is $\deg P$? At most $n-1$. Remember: n points determine a degree $\leq n-1$ polynomial.

Then, send $(0, P(0)), (1, P(1)), \ldots, (n+k-1, P(n+k-1))$ across the channel.

- Note: If the channel drops packets, the receiver *knows* which packets are dropped.

Property of polynomials: If we receive *any* n packets, then we can interpolate to recover the message.

If the channel drops at most k packets, we are safe.

Alternative Encoding

The message has packets $m_0, m_1, \ldots, m_{n-1}$.

Alternative Encoding

The message has packets $m_0, m_1, \ldots, m_{n-1}$.

Instead of encoding the messages as values of the polynomial, we can encode it as coefficients of the polynomial.

Alternative Encoding

The message has packets $m_0, m_1, \ldots, m_{n-1}$.

Instead of encoding the messages as values of the polynomial, we can encode it as coefficients of the polynomial.

$$P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0.$$

Alternative Encoding

The message has packets $m_0, m_1, \ldots, m_{n-1}$.

Instead of encoding the messages as values of the polynomial, we can encode it as coefficients of the polynomial.

$$P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0.$$

Then, send $(0, P(0)), (1, P(1)), \ldots, (n+k-1, P(n+k-1))$ as before.

Corruptions

Now you receive the following message:

“As d memkIrOcf tee GVwek tommcnity and X pZrt cf lneTof KVesZ oAcwWizytzoOs this ir higLly offensOvz.”

Corruptions

Now you receive the following message:

> *"As d memkIrOcf tee GVwek tommcnity and X pZrt cf lneTof KVesZ oAcwWizytzoOs this ir higLly offensOvz."*

Instead of letters being *erased*, letters are now corrupted.

Corruptions

Now you receive the following message:

> *"As d memkIrOcf tee GVwek tommcnity and X pZrt cf lneTof KVesZ oAcwWizytzoOs this ir higLly offensOvz."*

Instead of letters being *erased*, letters are now corrupted. These are called **general errors**.

Corruptions

Now you receive the following message:

> *"As d memkIrOcf tee GVwek tommcnity and X pZrt cf lneTof KVesZ oAcwWizytzoOs this ir higLly offensOvz."*

Instead of letters being *erased*, letters are now corrupted. These are called **general errors**.

Can we still recover the original message?

Corruptions

Now you receive the following message:

> *"As d memkIrOcf tee GVwek tommcnity and X pZrt cf lneTof KVesZ oAcwWizytzoOs this ir higLly offensOvz."*

Instead of letters being *erased*, letters are now corrupted. These are called **general errors**.

Can we still recover the original message?

In fact, Reed-Solomon codes still do the job!

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.

The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

Let $\text{Encode} : (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^{n+k}$ be the encoding function.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

Let $\text{Encode} : (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^{n+k}$ be the encoding function.
Let $\mathscr{C} := \mathsf{range}(\text{Encode})$ be the set of **codewords**.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \dots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \dots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

Let $\text{Encode} : (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^{n+k}$ be the encoding function.
Let $\mathscr{C} := \mathsf{range}(\text{Encode})$ be the set of **codewords**.
A codeword is a possible encoded message.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \ldots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \ldots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

Let Encode $: (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^{n+k}$ be the encoding function.
Let $\mathscr{C} := \mathsf{range}(\text{Encode})$ be the set of **codewords**.
A codeword is a possible encoded message.

We want the codewords to be far apart.

A Broader Look at Coding

Suppose we want to send a length-n message, $m_0, m_1, \dots, m_{n-1}$. Each packet is in $\mathbb{Z}/p\mathbb{Z}$.
The message $(m_0, m_1, \dots, m_{n-1})$ is in $(\mathbb{Z}/p\mathbb{Z})^n$.

We want to *encode* the message into $(\mathbb{Z}/p\mathbb{Z})^{n+k}$. The encoded message is *longer*, because redundancy recovers errors.

Let $\text{Encode} : (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^{n+k}$ be the encoding function.
Let $\mathscr{C} := \text{range}(\text{Encode})$ be the set of **codewords**.
A codeword is a possible encoded message.

We want the codewords to be far apart. Separated codewords means we can tolerate errors.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Proof of Triangle Inequality:

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Proof of Triangle Inequality:

- Start with s_1.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Proof of Triangle Inequality:

- Start with s_1.
- Change $d(s_1, s_2)$ symbols to get s_2.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Proof of Triangle Inequality:

- Start with s_1.
- Change $d(s_1, s_2)$ symbols to get s_2.
- Change $d(s_2, s_3)$ symbols to get s_3.

Hamming Distance

Given two strings s_1 and s_2, the **Hamming distance** $d(s_1, s_2)$ between two strings is the number of places where they differ.

Properties:

- $d(s_1, s_2) \geq 0$, with equality if and only if $s_1 = s_2$.
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$.
- Triangle Inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Proof of Triangle Inequality:

- Start with s_1.
- Change $d(s_1, s_2)$ symbols to get s_2.
- Change $d(s_2, s_3)$ symbols to get s_3.
- So s_1 and s_3 differ by at most $d(s_1, s_2) + d(s_2, s_3)$ symbols. $\square$

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

- Suppose we send the codeword c_{original}.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k + 1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.
- Consider a different codeword c_{other}.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.
- Consider a different codeword c_{other}.
- Then, $d(c_{\text{original}}, c_{\text{other}}) \leq d(c_{\text{original}}, s) + d(s, c_{\text{other}})$.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.
- Consider a different codeword c_{other}.
- Then, $d(c_{\text{original}}, c_{\text{other}}) \leq d(c_{\text{original}}, s) + d(s, c_{\text{other}})$.
- So, $2k+1 \leq k + d(s, c_{\text{other}})$.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k+1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.
- Consider a different codeword c_{other}.
- Then, $d(c_{\text{original}}, c_{\text{other}}) \leq d(c_{\text{original}}, s) + d(s, c_{\text{other}})$.
- So, $2k+1 \leq k + d(s, c_{\text{other}})$.
- So, $d(s, c_{\text{other}}) \geq k+1$.

Hamming Distance & Error Correction

Theorem: A code can recover k general errors if the minimum Hamming distance between any two distinct codewords is at least $2k + 1$.

Proof.

- Suppose we send the codeword c_{original}.
- It gets corrupted to a string s with $d(c_{\text{original}}, s) \leq k$.
- Consider a different codeword c_{other}.
- Then, $d(c_{\text{original}}, c_{\text{other}}) \leq d(c_{\text{original}}, s) + d(s, c_{\text{other}})$.
- So, $2k + 1 \leq k + d(s, c_{\text{other}})$.
- So, $d(s, c_{\text{other}}) \geq k + 1$.
- So s is closer to c_{original} than any other codeword. $\square$

Reed-Solomon Codes Revisited

Given a message $m = (m_0, m_1, \ldots, m_{n-1})$...

Reed-Solomon Codes Revisited

Given a message $m = (m_0, m_1, \ldots, m_{n-1})$...

- Define $P_m(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.

Reed-Solomon Codes Revisited

Given a message $m = (m_0, m_1, \ldots, m_{n-1})$...

- Define $P_m(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send the codeword $(0, P_m(0)), (1, P_m(1)), \ldots, (n+2k-1, P_m(n+2k-1))$.

Reed-Solomon Codes Revisited

Given a message $m = (m_0, m_1, \ldots, m_{n-1})$...

- Define $P_m(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send the codeword $(0, P_m(0)), (1, P_m(1)), \ldots, (n+2k-1, P_m(n+2k-1))$.

What are all the possible codewords?

Reed-Solomon Codes Revisited

Given a message $m = (m_0, m_1, \ldots, m_{n-1})\ldots$

- Define $P_m(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send the codeword $(0, P_m(0)), (1, P_m(1)), \ldots, (n+2k-1, P_m(n+2k-1))$.

What are all the possible codewords?
All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0, P_1(0)), (1, P_1(0)), \ldots, (n+2k-1, P_1(n+2k-1))$
c_2: $(0, P_2(0)), (1, P_2(0)), \ldots, (n+2k-1, P_2(n+2k-1))$

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0,P_1(0)),(1,P_1(0)),\ldots,(n+2k-1,P_1(n+2k-1))$
c_2: $(0,P_2(0)),(1,P_2(0)),\ldots,(n+2k-1,P_2(n+2k-1))$

If $d(c_1,c_2) \leq 2k$, then:

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0, P_1(0)), (1, P_1(0)), \ldots, (n+2k-1, P_1(n+2k-1))$
c_2: $(0, P_2(0)), (1, P_2(0)), \ldots, (n+2k-1, P_2(n+2k-1))$

If $d(c_1, c_2) \leq 2k$, then:
P_1 and P_2 share n points.

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0, P_1(0)), (1, P_1(0)), \ldots, (n+2k-1, P_1(n+2k-1))$
c_2: $(0, P_2(0)), (1, P_2(0)), \ldots, (n+2k-1, P_2(n+2k-1))$

If $d(c_1, c_2) \leq 2k$, then:
P_1 and P_2 share n points.
But n points uniquely determine degree $\leq n-1$ polynomials.

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0, P_1(0)), (1, P_1(0)), \ldots, (n+2k-1, P_1(n+2k-1))$
c_2: $(0, P_2(0)), (1, P_2(0)), \ldots, (n+2k-1, P_2(n+2k-1))$

If $d(c_1, c_2) \leq 2k$, then:
P_1 and P_2 share n points.
But n points uniquely determine degree $\leq n-1$ polynomials.
So $P_1 = P_2$.

Hamming Distance of Reed-Solomon Codes

Codewords: All possible sets of $n+2k$ points, which come from a polynomial of degree $\leq n-1$.

What is the minimum Hamming distance between distinct codewords?

Consider two codewords:

c_1: $(0, P_1(0)), (1, P_1(0)), \ldots, (n+2k-1, P_1(n+2k-1))$
c_2: $(0, P_2(0)), (1, P_2(0)), \ldots, (n+2k-1, P_2(n+2k-1))$

If $d(c_1, c_2) \leq 2k$, then:
P_1 and P_2 share n points.
But n points uniquely determine degree $\leq n-1$ polynomials.
So $P_1 = P_2$.

The minimum Hamming distance is $2k+1$.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \ldots, m_{n-1})$.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \ldots, m_{n-1})$.
- Define $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \ldots, m_{n-1})$.
- Define $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send codeword $(0, P(0)), (1, P(1)), \ldots, (n+2k-1, P(n+2k-1))$.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \ldots, m_{n-1})$.
- Define $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send codeword $(0, P(0)), (1, P(1)), \ldots, (n+2k-1, P(n+2k-1))$.
- The codeword suffers at most k corruptions.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \dots, m_{n-1})$.
- Define $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send codeword $(0, P(0)), (1, P(1)), \dots, (n+2k-1, P(n+2k-1))$.
- The codeword suffers at most k corruptions.
- Receiver decodes by searching for the closest codeword to the received message.

General Errors with Reed-Solomon Codes

Reed-Solomon with $n+2k$ packets gives a code with minimum Hamming distance $\geq 2k+1$ between distinct codewords.

By our theorem, this can correct k general errors.

What is the decoding algorithm?

- Take your message $m = (m_0, m_1, \ldots, m_{n-1})$.
- Define $P(x) = m_{n-1}x^{n-1} + \cdots + m_1 x + m_0$.
- Send codeword $(0, P(0)), (1, P(1)), \ldots, (n+2k-1, P(n+2k-1))$.
- The codeword suffers at most k corruptions.
- Receiver decodes by searching for the closest codeword to the received message.

Can we avoid exhaustive search?

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets.

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets. These packets are potentially corrupted!

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets. These packets are potentially corrupted!

Suppose there are errors at the values $e_1, \ldots, e_k$.

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets. These packets are potentially corrupted!

Suppose there are errors at the values $e_1, \ldots, e_k$. The **error locator polynomial** is:

$$E(x) = (x - e_1) \cdots (x - e_k).$$

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets. These packets are potentially corrupted!

Suppose there are errors at the values $e_1, \ldots, e_k$. The **error locator polynomial** is:

$$E(x) = (x - e_1) \cdots (x - e_k).$$

The roots of E are the locations of the errors.

Berlekamp-Welch Decoding Algorithm

Berlekamp and Welch patented an *efficient* decoding algorithm for Reed-Solomon codes.

Let $R_0, R_1, \ldots, R_{n-2k+1}$ be the received packets. These packets are potentially corrupted!

Suppose there are errors at the values $e_1, \ldots, e_k$. The **error locator polynomial** is:

$$E(x) = (x - e_1) \cdots (x - e_k).$$

The roots of E are the locations of the errors.

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n+2k-1$, we have:

$$P(i)E(i) = R_i E(i).$$

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$. Both sides are zero.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$. Both sides are zero.
- Case 2: i is not an error.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$. Both sides are zero.
- Case 2: i is not an error. Then, $P(i) = R_i$. □

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$. Both sides are zero.
- Case 2: i is not an error. Then, $P(i) = R_i$. □

Multiplying by the error locator polynomial "nullifies" the corruptions.

Berlekamp-Welch Lemma

Key Lemma: For all $i = 0, 1, \ldots, n + 2k - 1$, we have:

$$P(i)E(i) = R_i E(i).$$

Proof.

- Case 1: i is an error. Then, $E(i) = 0$. Both sides are zero.
- Case 2: i is not an error. Then, $P(i) = R_i$. □

Multiplying by the error locator polynomial "nullifies" the corruptions.

Problem: We do not know the locations of the errors.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n + 2k - 1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n + 2k - 1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Define $Q(x) := P(x)E(x)$.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Define $Q(x) := P(x)E(x)$.

Then, $\deg Q = \deg E + \deg P = n + k - 1$.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Define $Q(x) := P(x)E(x)$.

Then, $\deg Q = \deg E + \deg P = n + k - 1$.

So $Q(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$ for $n+k$ unknown coefficients $b_0, b_1, \ldots, b_{n+k-1}$.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Define $Q(x) := P(x)E(x)$.

Then, $\deg Q = \deg E + \deg P = n + k - 1$.

So $Q(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$ for $n+k$ unknown coefficients $b_0, b_1, \ldots, b_{n+k-1}$.

We have $n + 2k$ unknown coefficients.

Berlekamp-Welch Decoding

$$P(i)E(i) = R_i E(i) \qquad \text{for } i = 0, 1, \ldots, n+2k-1.$$

Since $\deg E = k$, then $E(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ for k unknown coefficients $a_0, a_1, \ldots, a_{k-1}$.

Note: Leading coefficient is one!

Define $Q(x) := P(x)E(x)$.

Then, $\deg Q = \deg E + \deg P = n + k - 1$.

So $Q(x) = b_{n+k-1}x^{n+k-1} + \cdots + b_1 x + b_0$ for $n+k$ unknown coefficients $b_0, b_1, \ldots, b_{n+k-1}$.

We have $n+2k$ unknown coefficients. But we also have $n+2k$ equations!

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

Equations, again:

$$b_{n+k-1} i^{n+k-1} + \cdots + b_1 i + b_0 = R_i(i^k + a_{k-1} i^{k-1} + \cdots + a_1 i + a_0).$$

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

Equations, again:

$$b_{n+k-1} i^{n+k-1} + \cdots + b_1 i + b_0 = R_i (i^k + a_{k-1} i^{k-1} + \cdots + a_1 i + a_0).$$

The equations are linear in the unknown variables.

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

Equations, again:

$$b_{n+k-1} i^{n+k-1} + \cdots + b_1 i + b_0 = R_i (i^k + a_{k-1} i^{k-1} + \cdots + a_1 i + a_0).$$

The equations are linear in the unknown variables.

Solve the linear system using methods from linear algebra.

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

Equations, again:

$$b_{n+k-1} i^{n+k-1} + \cdots + b_1 i + b_0 = R_i (i^k + a_{k-1} i^{k-1} + \cdots + a_1 i + a_0).$$

The equations are linear in the unknown variables.

Solve the linear system using methods from linear algebra.
Gaussian elimination.

The Equations Are Linear

Unknowns: $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{n+k-1}$.
Equations: $Q(i) = R_i E(i)$ for $i = 0, 1, \ldots, n+2k-1$.

Equations, again:

$$b_{n+k-1} i^{n+k-1} + \cdots + b_1 i + b_0 = R_i (i^k + a_{k-1} i^{k-1} + \cdots + a_1 i + a_0).$$

The equations are linear in the unknown variables.

Solve the linear system using methods from linear algebra.
Gaussian elimination.

Note: Linear algebra works over fields.

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

$$P(x) = \frac{Q(x)}{E(x)}.$$

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

$$P(x) = \frac{Q(x)}{E(x)}.$$

We have recovered the polynomial P, and therefore the message.

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

$$P(x) = \frac{Q(x)}{E(x)}.$$

We have recovered the polynomial P, and therefore the message.

The Berlekamp-Welch decoding algorithm is more efficient.

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

$$P(x) = \frac{Q(x)}{E(x)}.$$

We have recovered the polynomial P, and therefore the message.

The Berlekamp-Welch decoding algorithm is more efficient.

- Solving a linear system is much faster than exhaustive search of codewords.

Recovering the Encoding Polynomial

Solve a linear system, recover the coefficients of E and Q.

Note that $Q(x) = P(x)E(x)$, so we recover:

$$P(x) = \frac{Q(x)}{E(x)}.$$

We have recovered the polynomial P, and therefore the message.

The Berlekamp-Welch decoding algorithm is more efficient.

- Solving a linear system is much faster than exhaustive search of codewords.
- With more tricks, we can reduce the linear system (with $n+2k$ equations) into a system with only k equations.

Unique Solution?

Is the solution to the linear system unique?

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the “wrong” E and Q?

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the "wrong" E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the “wrong” E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.
Can we solve for the "wrong" E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E, Q) be *any* solution to the linear system.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the “wrong” E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E,Q) be *any* solution to the linear system. So, $Q(i) = R_i E(i)$ for $n+2k$ values of i.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the "wrong" E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E, Q) be *any* solution to the linear system. So, $Q(i) = R_i E(i)$ for $n+2k$ values of i.
- There are at most k errors so $R_i = P(i)$ for at least $n+k$ values of i.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.
Can we solve for the “wrong” E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E, Q) be *any* solution to the linear system. So, $Q(i) = R_i E(i)$ for $n + 2k$ values of i.
- There are at most k errors so $R_i = P(i)$ for at least $n + k$ values of i.
- So $Q(i) = P(i)E(i)$ for $n + k$ values of i.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.
Can we solve for the "wrong" E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E, Q) be *any* solution to the linear system. So, $Q(i) = R_i E(i)$ for $n+2k$ values of i.
- There are at most k errors so $R_i = P(i)$ for at least $n+k$ values of i.
- So $Q(i) = P(i)E(i)$ for $n+k$ values of i. But these are degree $n+k-1$ polynomials.

Unique Solution?

Is the solution to the linear system unique? Not if there are fewer than k errors.

Can we solve for the "wrong" E and Q?

Theorem: Any solutions E and Q have $Q(x)/E(x) = P(x)$.

Proof.

- Let (E, Q) be *any* solution to the linear system. So, $Q(i) = R_i E(i)$ for $n+2k$ values of i.
- There are at most k errors so $R_i = P(i)$ for at least $n+k$ values of i.
- So $Q(i) = P(i)E(i)$ for $n+k$ values of i. But these are degree $n+k-1$ polynomials.
- So $Q(x) = P(x)E(x)$ for all x. □

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets?

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

- We will learn counting soon.

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

- We will learn counting soon.
- There are $\binom{n+2k}{k}$ subsets of $R_0, R_1, \ldots, R_{n+2k-1}$.

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

- We will learn counting soon.
- There are $\binom{n+2k}{k}$ subsets of $R_0, R_1, \ldots, R_{n+2k-1}$.
- For each such subset, try fitting a polynomial of degree $\leq n-1$ which fits the remaining $n+k$ points.

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

- We will learn counting soon.
- There are $\binom{n+2k}{k}$ subsets of $R_0, R_1, \ldots, R_{n+2k-1}$.
- For each such subset, try fitting a polynomial of degree $\leq n-1$ which fits the remaining $n+k$ points.
- It is possible to bound:

$$\binom{n+2k}{k} \geq \left(\frac{n+2k}{k}\right)^k.$$

Comparison with Brute Force

Receive $R_0, R_1, \ldots, R_{n+2k-1}$.

Where are the corrupted packets? Brute force approach:

- We will learn counting soon.
- There are $\binom{n+2k}{k}$ subsets of $R_0, R_1, \ldots, R_{n+2k-1}$.
- For each such subset, try fitting a polynomial of degree $\leq n-1$ which fits the remaining $n+k$ points.
- It is possible to bound:

$$\binom{n+2k}{k} \geq \left(\frac{n+2k}{k}\right)^k.$$

The complexity grows exponentially with k.

Summary

- Two ways to encode information in a polynomial: as values, or as coefficients.
- Secret sharing: Encode secret in polynomial, hand out "shares" of the polynomial to officials.
  - If any k officials come together, they know the secret, but $k-1$ officials know nothing.
- If minimum Hamming distance between distinct codewords is $2k+1$, then correct k general errors.
- Reed-Solomon codes: Interpolate a polynomial through n packets and send values of the polynomial.
  - To correct k erasure errors, send $n+k$.
  - To correct k general errors, send $n+2k$.
- The error locator polynomial E has a root at every error.
- Berlekamp-Welch decoding: $Q(x) = P(x)E(x)$, solve for the coefficients of E and Q using $Q(i) = R_i E(i)$.