Algebraic Algorithms for Cryptology

Clément PERNET

M1 MoSIG / Info / AM

Organization of the course

Content: computer algebra fundations for cryptology

- Computational aspects of integer arithmetic, finite groups, and finite fields.
  - algorithms and complexity analysis
  - software implementations
- Application to error correcting codes

Organization of the course

Content: computer algebra fundations for cryptology

- Computational aspects of integer arithmetic, finite groups, and finite fields.
  - algorithms and complexity analysis
  - software implementations
- Application to error correcting codes

- $11 \times 1.5h$ of CTD (mix of plenary lecture and tutorial)
- 2 TP (lab session) as home-work

Grading: average of the TP grades

Outline

Algebraic **Computing**

Computing: Algorithms, Complexity, Implementations

Security in cryptology relies on one-way functions: easy to compute, but hard to invert

Easy: cost analysis, fast software implementations

Hard: complexity theory and reductions, fast implementation of expensive attacks

Algebraic Computing

Computing: Algorithms, Complexity, Implementations

Security in cryptology relies on one-way functions: easy to compute, but hard to invert

Easy: cost analysis, fast software implementations

Hard: complexity theory and reductions, fast implementation of expensive attacks

Algebra: finite fields, finite groups, integer and polynomial arithmetic

A good source of one way functions:

- integer multiplication/factorization,
- exponentiation / discrete logarithm in a group, e.g. $(\mathbb{F}_q)^*$
- algebraic coding theory, etc

Outline

Computational cost / complexity

How to guess the cost of the execution of an algorithm on a given instance?

- in time
- in space

Computational cost / complexity

How to guess the cost of the execution of an algorithm on a given instance?

- in time
- in space

Defining a cost model (simplifying assumptions)

- Define units: which operation has cost 1 ? Which data can be stored in space 1 ?

$=$

Computational cost / complexity

How to guess the cost of the execution of an algorithm on a given instance?

- in time
- in space

Defining a cost model (simplifying assumptions)

- Define units: which operation has cost 1 ? Which data can be stored in space 1 ?
- Cost only depends on the input size (or a parameter related to it):
  - uniform across all instances
  - worst case analysis, (sometimes average case analysis)

$$C(n) =$$

Computational cost / complexity

How to guess the cost of the execution of an algorithm on a given instance?

- in time
- in space

Defining a cost model (simplifying assumptions)

- Define units: which operation has cost 1 ? Which data can be stored in space 1 ?
- Cost only depends on the input size (or a parameter related to it):
  - uniform across all instances
  - worst case analysis, (sometimes average case analysis)
- Asymptotic analysis : mostly care about large instances

$$C(n) = O(n^2)$$

Asymptotics

Landau notation:

- $f(n) = O(g(n))$ iff $f(n) \leq Kg(n)\ \forall\ n \geq n_0$ for some $K > 0$ and $n_0 \geq 0$
- $f(n) = \Omega(g(n))$ iff $g(n) = \mathcal{O}(f(n))$
- $f(n) = \Theta(g(n))$ iff $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$

Equivalently, $f(n) = O(g(n))$ if $f(n)/g(n)$ is bounded by a constant for all n sufficiently large.

Asymptotics

Landau notation:

- $f(n) = O(g(n))$ iff $f(n) \leq Kg(n)\ \forall\ n \geq n_0$ for some $K > 0$ and $n_0 \geq 0$
- $f(n) = \Omega(g(n))$ iff $g(n) = \mathcal{O}(f(n))$
- $f(n) = \Theta(g(n))$ iff $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$

Equivalently, $f(n) = O(g(n))$ if $f(n)/g(n)$ is bounded by a constant for all n sufficiently large.

Example

$$
\begin{aligned}
2n^3 - 3n^2 \log n + 5n + 12 &= \Theta(n^3) \\
n + 1 &= O(\frac{1}{1000} n) \\
n \log n &= O(n^2)
\end{aligned}
$$

$$
\begin{aligned}
n^2 + 100000 n^{1.9} &= \Omega(n^2) \\
(3n + 1) \log^2 n &\neq O(n \log n) \\
2^n &\neq O(n^k) \text{ for any } k \in \mathbb{Z}
\end{aligned}
$$

Asymptotics

poly-logarithmic notations (*soft-O*)

$f(n) = \tilde{\mathcal{O}}(g(n))$ iff $f(n) = \mathcal{O}(g(n)\log^e g(n))$ for some $e > 0$

Asymptotics

poly-logarithmic notations (*soft-O*)

$f(n) = \tilde{\mathcal{O}}(g(n))$ iff $f(n) = \mathcal{O}(g(n)\log^e g(n))$ for some $e > 0$

Example

$$n \times \log n \times \log\log n = \tilde{\mathcal{O}}(n)$$

$\rightsquigarrow$ Quasi-linear cost.

Magnitudes

Linear or Exp time ?

Size of an integer n represented in base 2 : $s = \lceil \log_2 n \rceil$ bits.

$$n = \Theta(2^s) = \Theta(exp(s))$$

$\rightsquigarrow$ any algorithm working on an integer n with cost linear in n takes actually an exponential time in the input size.

Orders of magnitude in practice

Nowadays' computers are quite fast

Speed of a PC: 3GHz $\rightsquigarrow 3 \times 10^9 \times 4 \times 2$ `int64_t` mult. per sec.

- Video projector is at 3m of the screen: $300\,000 km/s \rightsquigarrow 10^{-8}$s
- 240 multiplications done before the light reaches the screen

Orders of magnitude in practice

Nowadays' computers are quite fast

Speed of a PC: 3GHz $\rightsquigarrow 3 \times 10^9 \times 4 \times 2$ `int64_t` mult. per sec.

- Video projector is at 3m of the screen: $300\,000 km/s \rightsquigarrow 10^{-8}$s
- 240 multiplications done before the light reaches the screen

- Age of the universe : 15 billion $\times 365 \times 24 \times 3600 \approx 5.10^{17} s \approx 2^{59} s$

Orders of magnitude in practice

Nowadays' computers are quite fast

Speed of a PC: 3GHz $\rightsquigarrow 3 \times 10^9 \times 4 \times 2$ `int64_t` mult. per sec.

- Video projector is at 3m of the screen: $300\,000 km/s \rightsquigarrow 10^{-8}$s
- 240 multiplications done before the light reaches the screen

- Age of the universe : 15 billion $\times 365 \times 24 \times 3600 \approx 5.10^{17} s \approx 2^{59} s$
- Number of electrons in the universe : $\approx 10^{64} \approx 2^{213}$

Orders of magnitude in practice

Nowadays' computers are quite fast

Speed of a PC: 3GHz $\rightsquigarrow 3 \times 10^9 \times 4 \times 2$ `int64_t` mult. per sec.

- Video projector is at 3m of the screen: $300\,000 km/s \rightsquigarrow 10^{-8}$s
- 240 multiplications done before the light reaches the screen

- Age of the universe : 15 billion $\times 365 \times 24 \times 3600 \approx 5.10^{17} s \approx 2^{59} s$
- Number of electrons in the universe : $\approx 10^{64} \approx 2^{213}$
- Costs for algorithms working with 128 bit integers

Cost	s	s^2	s^3	s^4	$n = 2^s$
Nb of ops	128	16 384	$2 \cdot 10^6$	$3 \cdot 10^8$	10^{39}
Time on a 2.5Ghz PC	$5.3ns$	$0.68\mu s$	$87.4\mu s$	$11.2ms$	$2^{93.5} s$

$\rightsquigarrow 2^{93.5} s \approx 2^{34.5} \times \approx 2.4 \times 10^{10} \times$ the age of the universe !

Outline

Outline

Integer arithmetic

Fixed precision 32, 64 bits : word size integers

`uint32_t:` $[0..2^{32}-1]$

`int32_t:` $[-2^{31}+1..2^{31}-1]$

`uint64_t:` $[0..2^{64}-1]$

`int64_t:` $[-2^{63}+1..2^{63}-1]$

Atomic cost:

- add, mul, sub: ≈ 1 clock cycle;
- div, mod : ≈ 10 clock cycles

Integer arithmetic

Fixed precision 32, 64 bits (24, 53): word size integers

`uint32_t:` $[0..2^{32}-1]$ `uint64_t:` $[0..2^{64}-1]$

`int32_t:` $[-2^{31}+1..2^{31}-1]$ `int64_t:` $[-2^{63}+1..2^{63}-1]$

Atomic cost:

- add, mul, sub: ≈ 1 clock cycle;
- div, mod : ≈ 10 clock cycles

Alternatively, one can store integers on floating point types:

`float:` $[-2^{23}+1..2^{23}-1]$

`double:` $[-2^{52}+1..2^{52}-1]$

$\rightsquigarrow$ faster on most CPUs, but slightly smaller representation capacity

Integer arithmetic

Fixed precision 32, 64 bits (24, 53): word size integers

`uint32_t`: $[0..2^{32}-1]$ `uint64_t`: $[0..2^{64}-1]$

`int32_t`: $[-2^{31}+1..2^{31}-1]$ `int64_t`: $[-2^{63}+1..2^{63}-1]$

Atomic cost:

- add, mul, sub: ≈ 1 clock cycle;
- div, mod : ≈ 10 clock cycles

Alternatively, one can store integers on floating point types:

`float`: $[-2^{23}+1..2^{23}-1]$

`double`: $[-2^{52}+1..2^{52}-1]$

$\rightsquigarrow$ faster on most CPUs, but slightly smaller representation capacity

$\rightsquigarrow$ used for small integers; small finite fields/rings, ...

Integer arithmetic

Multi-precision

- No native hardware support
- Software emulation: C/C++ libraries GMP/MPIR:
  $\rightsquigarrow$ vectors of 64 bits unsigned words (called limbs)

Basic arithmetic no longer have unit cost:
$\rightsquigarrow$ depend on the number of limbs

$$s = \#\text{of limbs} = (\log_2 n)/64 = \log_{2^{64}} n$$

Multiprecision Integer arithmetic

Addition		$\mathcal{O}(s)$
Multip.	Classic	$\mathcal{O}(s^2)$
Division		$\mathcal{O}(s^2)$
GCD	Euclidean Alg.	$\mathcal{O}(s^2)$

Multiprecision Integer arithmetic

Addition			$\mathcal{O}(s)$
Multip.	Classic	$s < 32$ words	$\mathcal{O}(s^2)$
	Karatsuba	$32 < s < 256$	$\mathcal{O}(s^{1.585})$
	Toom-Cook		$\mathcal{O}(s^{1.465})$
	FFT	$s > 10000$ words	$\mathcal{O}(s \log s) = \tilde{\mathcal{O}}(s)$
Division			$\mathcal{O}(s^2)$
			$\mathcal{O}(\text{Mult}(s)) = \tilde{\mathcal{O}}(s)$
GCD	Euclidean Alg.		$\mathcal{O}(s^2)$
	Fast Euclid. Alg.		$\mathcal{O}(M(s) \log s) = \tilde{\mathcal{O}}(s)$

A swiss army knife for computing costs of divide and conquer algorithms

Theorem (Master Theorem)

Consider a divide and conquer algorithm, dividing the input in b parts of equal size, and making a recursive calls. Define $\alpha = \log_b a$. If its cost satisfies

$$\begin{cases} C(n) &= aC(\frac{n}{b}) + f(n) \\ C(1) &= c \end{cases}$$

then

1. *If* $f(n) = \mathcal{O}\left(n^{\alpha-\epsilon}\right)$ *for some* $\epsilon > 0$ *then* $C(n) = \Theta(n^{\alpha})$
2. *If* $f(n) = \Theta(n^{\alpha})$ *then* $C(n) = \Theta(n^{\alpha} \log n)$
3. *If* $f(n) = \Omega(n^{\alpha+\epsilon})$ *for some* $\epsilon > 0$ *and* $af(n/b) \leq kf(n)$ *with* $k < 1$ *then* $C(n) = \Theta(f(n))$

The Euclidean Division

Theorem

For every $a, b \in \mathbb{Z}$, there is a unique pair $q, r \in \mathbb{Z}$ with $0 \leq r < |b|$ such that $a = bq + r$.

The Euclidean Division

Theorem

For every $a, b \in \mathbb{Z}$, there is a unique pair $q, r \in \mathbb{Z}$ with $0 \leq r < |b|$ such that $a = bq + r$.

Proof by a slow algorithm.

begin

$q_1 \leftarrow 0$;
$r_1 \leftarrow a$;
$ui \leftarrow 1$;
while $r_i \geq 0$ **do**
$r_{i+1} \leftarrow r_i - b$;
$q_{i+1} \leftarrow q_i + 1$;
$i = i + 1$;

return $(q, r) \leftarrow (q_i, r_i)$

□

The Euclidean Division

Theorem

For every $a, b \in \mathbb{Z}$, there is a unique pair $q, r \in \mathbb{Z}$ with $0 \leq r < |b|$ such that $a = bq + r$.

Proof by a slow algorithm.

begin

$q_1 \leftarrow 0$;
$r_1 \leftarrow a$;
$ui \leftarrow 1$;
while $r_i \geq 0$ **do**
$r_{i+1} \leftarrow r_i - b$;
$q_{i+1} \leftarrow q_i + 1$;
$i = i + 1$;

return $(q, r) \leftarrow (q_i, r_i)$

□

- q is the quotient
- r is the remainder, also called the residue of a modulo b, denoted by $r = a \mod b$

The Euclidean Division

Theorem

For every $a, b \in \mathbb{Z}$, there is a unique pair $q, r \in \mathbb{Z}$ with $0 \leq r < |b|$ such that $a = bq + r$.

Proof by a slow algorithm.

begin

$q_1 \leftarrow 0$;
$r_1 \leftarrow a$;
$ui \leftarrow 1$;
while $r_i \geq 0$ **do**
$r_{i+1} \leftarrow r_i - b$;
$q_{i+1} \leftarrow q_i + 1$;
$i = i + 1$;

return $(q, r) \leftarrow (q_i, r_i)$

□

- q is the quotient
- r is the remainder, also called the residue of a modulo b, denoted by $r = a \mod b$
- Cost: nb of iter.: $q \approx a/b \approx 2^{s_a - s_b}$
  $\rightsquigarrow C(s_a, s_b) = \mathcal{O}\left(s_a 2^{s_a - s_b}\right)$

The Euclidean Division

Theorem

For every $a, b \in \mathbb{Z}$, there is a unique pair $q, r \in \mathbb{Z}$ with $0 \leq r < |b|$ such that $a = bq + r$.

Proof by a slow algorithm.

begin

$q_1 \leftarrow 0$;
$r_1 \leftarrow a$;
$ui \leftarrow 1$;
while $r_i \geq 0$ **do**
$r_{i+1} \leftarrow r_i - b$;
$q_{i+1} \leftarrow q_i + 1$;
$i = i + 1$;

return $(q, r) \leftarrow (q_i, r_i)$

□

- q is the quotient
- r is the remainder, also called the residue of a modulo b, denoted by $r = a \mod b$
- Cost: nb of iter.: $q \approx a/b \approx 2^{s_a - s_b}$ $\rightsquigarrow C(s_a, s_b) = \mathcal{O}\left(s_a 2^{s_a - s_b}\right)$
- using elementary school division algorithm $C(s_a, s_b) = \mathcal{O}\left(s_b(s_a - s_b)\right)$

GCD and Euclidean Algorithm

Definition (GCD = Greatest Common Divisor)

The GCD of a and b is the greatest integer g dividing both a and b

Example

- $\mathrm{GCD}(12, 16) = 4$
- $\mathrm{GCD}(12, 17) = 1 \rightsquigarrow 12$ and 17 are *coprime*

GCD and Euclidean Algorithm

Property

- $GCD(-a, b) = GCD(a, b))$
- $GCD(a, b) = GCD(b, a))$
- $GCD(a, b) = GCD(a - b, b)$
- $GCD(a, b) = GCD(a \mod b, b)$

where $a \mod b$ is the remainder of the euclidean division of a by b.

GCD and Euclidean Algorithm

Problem

Given $a, b \in \mathbb{Z}$*, find* $g = GCD(a, b)$

begin

$r_0 = a$;
$r_1 = b$;
while $r_i \neq 0$ **do**
$r_{i+1} = r_{i-1} \mod r_i$; /* $r_{i-1} = r_i q_i + r_{i+1}$ */
$i = i + 1$;

- The last $r_i \neq 0$ is the gcd of a and b

GCD and Euclidean Algorithm

Bezout relation

If $g = \text{GCD}(a, b)$, then there exist $u, v \in \mathbb{Z}$, coprime such that $g = ua + vb$

begin

$r_0 = a$, $u_0 = 1, v_0 = 0$;
$r_1 = b$, $u_1 = 0, v_1 = 1$;
while $r_i \neq 0$ **do**
$r_{i+1} = r_{i-1} \mod r_i$; /* $r_{i-1} = r_i q_i + r_{i+1}$ */
$u_{i+1} = u_{i-1} - q_i u_i$;
$v_{i+1} = v_{i-1} - q_i v_i$;
$i = i + 1$;

- The last $r_i \neq 0$ is the gcd of a and b
- invariant $u_i a + v_i b = r_i$ for all $i \rightsquigarrow$ Bezout coefficients

Outline

Finite ring and fields: $\mathbb{Z}/n\mathbb{Z}$

Integers modulo n

$\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$ equiped with addition et mult. *modulo* n.

- use integer arithmetic
- reduce the results mod n

Finite ring and fields: $\mathbb{Z}/n\mathbb{Z}$

Integers modulo n

$\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$ equiped with addition et mult. *modulo n*.

- use integer arithmetic
- reduce the results mod n

Addition	`c = a + b;` `if (c >= n) c = c - n;`
Opposé	`if (a) c = n - a; else c = a;`
Multiplication	`c = a * b;` `if (c >= n) c = c % n; // c modulo n`
Inverse	...

Modular Inverse

Modulo n any non-zero element does not necessarily have an inverse: $2^{-1} \mod 4$

Computing the modular inverse $a^{-1} \mod n$

$\mathrm{GCD}(a, n) = 1 \Leftrightarrow ua + vn = 1 \Leftrightarrow ua = 1 \mod n \Leftrightarrow a^{-1} = u \mod n.$

Corollary

$\mathbb{Z}/p\mathbb{Z}$ is a field iff p is prime

Corollary

Any finite field is isomorphic to either

- *$\mathbb{Z}/p\mathbb{Z}$ for a prime p or*
- *$\mathbb{Z}/p\mathbb{Z}[X]/(Q)$ where $Q \in \mathbb{Z}/p\mathbb{Z}[X]$ is irreducible*

Outline

The Chinese remainder theorem

Problem (Sunzi Suanjing)

Find n *knowing that* $\begin{cases} n \mod 3 = 2, \\ n \mod 5 = 3, \\ n \mod 7 = 2 \end{cases}$

$\rightsquigarrow$ $n = 23 + 105k$ *for* $k \in \mathbb{Z}$.

$\rightsquigarrow$ *unique integer between* 0 *and* 104

The Chinese remainder theorem

Problem (Sunzi Suanjing)

Find n *knowing that* $\begin{cases} n \mod 3 = 2, \\ n \mod 5 = 3, \\ n \mod 7 = 2 \end{cases}$

$\rightsquigarrow$ $n = 23 + 105k$ *for* $k \in \mathbb{Z}$.

$\rightsquigarrow$ *unique integer between* 0 *and* 104

Theorem

If p, q *are coprime and* x, y *are residues modulo* p *and* q*. Then* $\exists! A \in \mathbb{Z}_+, A < pq$*, such that*

$$\begin{cases} A = x \mod p \\ A = y \mod q \end{cases}$$

The Chinese remainder theorem

Theorem (Alternative formulation)

If p, q are coprime, then there is an isomorphism between the rings

$$\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \equiv \mathbb{Z}/(pq)\mathbb{Z}.$$

The Chinese remainder theorem

Theorem (Alternative formulation)

If p, q are coprime, then there is an isomorphism between the rings

$$\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \equiv \mathbb{Z}/(pq)\mathbb{Z}.$$

Isomorphism:

$$\begin{aligned} f: \quad \mathbb{Z}/(pq)\mathbb{Z} &\to \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \\ n &\mapsto (n \mod p, n \mod q) \end{aligned}$$

$$\begin{aligned} f^{-1}: \quad \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} &\to \mathbb{Z}/(pq)\mathbb{Z} \\ (x, y) &\mapsto xq(q^{-1} \mod p) + yp(p^{-1} \mod q) \mod pq \end{aligned}$$

The Chinese remainder theorem

Theorem

If $m_1, \ldots, m_k$ are pairwise relatively prime,

$$\mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z} \equiv \mathbb{Z}/(m_1 \ldots m_k)\mathbb{Z}.$$

Isomorphism:

$$\begin{array}{rrcl} f: & \mathbb{Z}/(m_1 \ldots m_k)\mathbb{Z} & \to & \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z} \\ & n & \mapsto & (n \mod m_1, \ldots, m \mod m_k) \\ f^{-1}: & \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z} & \to & \mathbb{Z}/(m_1 \ldots m_k)\mathbb{Z} \\ & (x_1, \ldots, x_k) & \mapsto & \sum_{i=1}^{k} x_i \Pi_i Y_i \mod \Pi \end{array}$$

where $\begin{cases} \Pi = \prod_{i=1}^{k} m_i \\ \Pi_i = \Pi / m_i \\ Y_i = \Pi_i^{-1} \mod m_i \end{cases}$

The Chinese Remainder Theorem

Theorem (Alternative formulation)

If $m_1, \ldots, m_k$ are pairwise relatively prime and $a_1, \ldots, a_k$ are residues modulo resp. $m_1, \ldots, m_k$. Then $\exists! A \in \mathbb{Z}_+, A < \prod_{i=1}^{k} m_i$, such that

$$A = a_i \mod m_i \; \forall i = 1 \ldots k.$$

Analogy with the polynomials

Over the ring of polynomials $K[X]$ (for any field K),

$$P(a) = P \mod (X - a)$$

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Analogy with the polynomials

Over the ring of polynomials $K[X]$ (for any field K),

$$P(a) = P \mod (X - a)$$

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Polynomials	**Integers**
Evaluation:	
$y = P \mod (X - a)$	$y = N \mod m$
$y = P(a)$	$y =$ "Evaluation" of N in m
Interpolation:	
$P = \sum_{i=1}^{k} y_i \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	$N = \sum_{i=1}^{k} y_i \prod_{j \neq i} m_j (\prod_{j \neq i} m_j)^{-1[m_i]}$

Outline

Outline

Definition (informally)

A group $(G, *, 1)$**:** is a set G with an associative law $*$ such that

- 1 is a neutral element $x * 1 = 1 * x = x$
- every element of G is invertible: $\forall x \in G \, \exists y \in G, x * y = y * x = 1$
- Examples: $(\mathbb{Z}, +, 0); (\mathbb{Q} \setminus \{0\}, \times, 1)$

Definition (informally)

A group $(G, *, 1)$**:** is a set G with an associative law $*$ such that

- 1 is a neutral element $x * 1 = 1 * x = x$
- every element of G is invertible: $\forall x \in G \, \exists y \in G, x * y = y * x = 1$
- Examples: $(\mathbb{Z}, +, 0); (\mathbb{Q} \setminus \{0\}, \times, 1)$

A ring $(R, +, \times, 0, 1)$ is

- a group $(R, +, 0)$
- with an associative law $\times$ with neutral element 1.
- such that $0 \times x = 0$
- Examples: $(\mathbb{Z}, +, \times, 0, 1); (\mathbb{Z}/n\mathbb{Z}, +, \times, 0, 1); (\mathbb{Z}[X], +, \times, 0, 1)$

Groups, Rings, Fields

Definition (informally)

A group $(G, *, 1)$**:** is a set G with an associative law $*$ such that

- 1 is a neutral element $x * 1 = 1 * x = x$
- every element of G is invertible: $\forall x \in G \, \exists y \in G, x * y = y * x = 1$
- Examples: $(\mathbb{Z}, +, 0); (\mathbb{Q} \setminus \{0\}, \times, 1)$

A ring $(R, +, \times, 0, 1)$ is

- a group $(R, +, 0)$
- with an associative law $\times$ with neutral element 1.
- such that $0 \times x = 0$
- Examples: $(\mathbb{Z}, +, \times, 0, 1); (\mathbb{Z}/n\mathbb{Z}, +, \times, 0, 1); (\mathbb{Z}[X], +, \times, 0, 1)$

A field $(F, +, \times, 0, 1)$ is

- a ring $(F, +, \times, 0, 1)$
- where every element except 0 has an inverse for $\times$
- equivalently such that $(F \setminus \{0\}, \times, 1)$ is a group.
- Examples: $(\mathbb{Q}, +, \times, 0, 1); (\mathbb{Z}/p\mathbb{Z}, +, \times, 0, 1)$ for p prime

An example of a finite ring: $\mathbb{Z}/n\mathbb{Z}$

$\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$ equiped with addition and mult. *modulo* n.

- $(\mathbb{Z}/n\mathbb{Z}, +, \times, 0, 1)$ is a ring
- not necessarily a field: e.g. $n = pq$
  - $\rightsquigarrow pq = 0 \mod n$
  - $\rightsquigarrow$ if p is invertible, then $p^{-1}pq = q = 0 \mod n$
  - $\rightsquigarrow$ neither p nor q have an inverse $\mod n$

An example of a finite ring: $\mathbb{Z}/n\mathbb{Z}$

$\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$ equiped with addition and mult. *modulo* n.

- $(\mathbb{Z}/n\mathbb{Z}, +, \times, 0, 1)$ is a ring
- not necessarily a field: e.g. $n = pq$
  - $\rightsquigarrow pq = 0 \mod n$
  - $\rightsquigarrow$ if p is invertible, then $p^{-1}pq = q = 0 \mod n$
  - $\rightsquigarrow$ neither p nor q have an inverse $\mod n$

Theorem

$(\mathbb{Z}/n\mathbb{Z}, +, \times, 0, 1)$ *is a field iff* n *is prime.*

Constructive proof.

By the Extended Euclidean Algorithm □

Multiplicative group of a ring

If $(R, +, \times, 0, 1)$ is a ring, not all elements of R are invertible for $\times$.

Definition (Multiplicative group of a ring R)

In a ring $(R, +, \times)$, the subset of the invertible elements w.r.t. $\times$ is a group, called the multiplicative subgroup of R and denoted by R^*.

Multiplicative group of a ring

If $(R, +, \times, 0, 1)$ is a ring, not all elements of R are invertible for $\times$.

Definition (Multiplicative group of a ring R)

In a ring $(R, +, \times)$, the subset of the invertible elements w.r.t. $\times$ is a group, called the multiplicative subgroup of R and denoted by R^*.

- If R is a field, any non-zero element is invertible, $\rightsquigarrow R^* = R \setminus \{0\}$
- $(\mathbb{Z}/n\mathbb{Z})^* = \{x \in \mathbb{Z}/n\mathbb{Z} \text{ s.t. } \mathrm{GCD}(x, n) = 1\}$

Outline

Lagrange, Euler, Fermat

Definition

finite group: *a group with a finite number of elements*

order of an element x***:*** $o(x) = \#\{x^i, i \in \mathbb{Z}\}$

order of a finite group: $o(G) = \#G$

cyclic group: *a finite group generated by a single element:* $G = \{g^i, i \in \mathbb{Z}\}$ *for some* $g \in G$

Lagrange, Euler, Fermat

Definition

finite group: *a group with a finite number of elements*

order of an element x***:*** $o(x) = \#\{x^i, i \in \mathbb{Z}\}$

order of a finite group: $o(G) = \#G$

cyclic group: *a finite group generated by a single element:* $G = \{g^i, i \in \mathbb{Z}\}$ *for some* $g \in G$

Theorem (Lagrange)

For any finite group $(G, \times, 1)$ *and any* $a \in G$*, we have* $a^{\#G} = 1$.

Corollary

The order of any element divides that of the its group: $\forall a \in G,\ o(a) | \#G$

Lagrange, Euler, Fermat

Definition

finite group: *a group with a finite number of elements*

order of an element x***:*** $o(x) = \#\{x^i, i \in \mathbb{Z}\}$

order of a finite group: $o(G) = \#G$

cyclic group: *a finite group generated by a single element:* $G = \{g^i, i \in \mathbb{Z}\}$ *for some* $g \in G$

Theorem (Lagrange)

For any finite group $(G, \times, 1)$ *and any* $a \in G$*, we have* $a^{\#G} = 1$.

Corollary

The order of any element divides that of the its group: $\forall a \in G,\ o(a) | \#G$

Theorem (Lagrange-v2)

If H *is a sub-group of* G*, then* $\#H | \#G$

Lagrange, Euler, Fermat

Definition

finite group: *a group with a finite number of elements*

order of an element x***:*** $o(x) = \#\{x^i, i \in \mathbb{Z}\}$

order of a finite group: $o(G) = \#G$

cyclic group: *a finite group generated by a single element:* $G = \{g^i, i \in \mathbb{Z}\}$ *for some* $g \in G$

Theorem (Lagrange)

For any finite group $(G, \times, 1)$ *and any* $a \in G$*, we have* $a^{\#G} = 1$.

Corollary

The order of any element divides that of the its group: $\forall a \in G,\ o(a) | \#G$

Theorem (Lagrange-v2)

If H *is a sub-group of* G*, then* $\#H | \#G$

Property

Any sub-group H *of a cyclic group* G *is cyclic.*

Euler totient function

Definition

- *Euler Totient:* $\varphi(n) = \#(\mathbb{Z}/n\mathbb{Z})^*$
- *Hence* $\varphi(n) = \#\{x \in \mathbb{Z}/n\mathbb{Z}, \mathit{GCD}(x, n) = 1\}$

Euler totient function

Definition

- *Euler Totient:* $\varphi(n) = \#(\mathbb{Z}/n\mathbb{Z})^*$
- *Hence* $\varphi(n) = \#\{x \in \mathbb{Z}/n\mathbb{Z}, \textit{GCD}(x, n) = 1\}$

Property

- $\varphi(p) = (p-1)$ *for* p *prime*
- $\varphi(p^k) = (p-1)p^{k-1}$ *for* p *prime*
- $\varphi(mn) = \varphi(m)\varphi(n)$ *for* $\textit{GCD}(m, n) = 1$

Euler totient function

Definition

- *Euler Totient:* $\varphi(n) = \#(\mathbb{Z}/n\mathbb{Z})^*$
- *Hence* $\varphi(n) = \#\{x \in \mathbb{Z}/n\mathbb{Z}, \textit{GCD}(x,n) = 1\}$

Property

- $\varphi(p) = (p-1)$ *for* p *prime*
- $\varphi(p^k) = (p-1)p^{k-1}$ *for* p *prime*
- $\varphi(mn) = \varphi(m)\varphi(n)$ *for* $\textit{GCD}(m,n) = 1$

Example: $n = \prod_{i=1}^{k} p_i^{\alpha_i}$ (prime factor decomposition)

$$\varphi(n) = \prod_{i=1}^{k} p_i^{\alpha_i - 1}(p_i - 1)$$

Appliction to finite groups

Property

The number of generators in a cyclic group of order n is $\varphi(n)$

Proof.

If g is a generator. Then,

$$h \text{ is a generator } \Leftrightarrow h = g^i \text{ and } g = h^k \Leftrightarrow h = h^{ik} \mod n \Leftrightarrow ik = 1 \mod n.$$

□

Euler, Fermat

Theorem (Euler)

Let $a, n \in \mathbb{Z}$. If GCD$(a, n) = 1$, then $a^{\varphi(n)} = 1 \mod n$.

Theorem (Fermat)

If p is prime, then $a^p = a \mod p \; \forall a \in \mathbb{Z}/p\mathbb{Z}$.

Théorème RSA

Theorem

For $n = pq$ with p and q prime, then

$$\forall k \in \mathbb{Z}\ \forall a \in \mathbb{Z}/n\mathbb{Z}\ \ a^{1+k\varphi(n)} = a \mod n$$

Théorème RSA

Theorem

For $n = pq$ with p and q prime, then

$$\forall k \in \mathbb{Z}\ \forall a \in \mathbb{Z}/n\mathbb{Z}\ \ a^{1+k\varphi(n)} = a \mod n$$

Proof.

$\varphi(n) = (p-1)(q-1)$

- If a is invertible $\rightsquigarrow$ Fermat: $a^{\varphi(n)} = 1 \mod n$
- If $a = 0 \mod n \rightsquigarrow$ trivial
- Otherwise:

  modulo p: a invertible $\rightsquigarrow$ Euler $(a^{p-1})^{q-1} = 1 \mod p \rightsquigarrow a^{1+k\varphi(n)} = a \mod p$
  modulo q: $a = 0 \mod q \rightsquigarrow a^{1+k\varphi(n)} = 0 = a \mod q$

  Chinese Remainder Theorem $\rightsquigarrow a^{\varphi(n)} = 1 \mod n$

□

Outline

Algebraic extensions

Consider a field $(K, +, \times)$, and a polynomial $P \in K[X]$ of degree d.

- $K[X]/(P)$ is the set of equivalence classes of $K[X]$ modulo P.
- This is the set of the $P \in K[X]$ with degree $< d$ equipped with the following laws

  Addition: $S + T = S(X) +_{K[X]} T(X) \mod P$
  Multiplication: $S \times T = S(X) \times_{K[X]} T(X) \mod P$

- $(K[X]/(P), +, \times)$ is thus a commutative ring, called the *quotient ring* of $K[X]$ by P.

Galois fields

Algebraic extensions

Consider a field $(K, +, \times)$, and a polynomial $P \in K[X]$ of degree d.

- $K[X]/(P)$ is the set of equivalence classes of $K[X]$ modulo P.
- This is the set of the $P \in K[X]$ with degree $< d$ equipped with the following laws

  Addition: $S + T = S(X) +_{K[X]} T(X) \mod P$
  Multiplication: $S \times T = S(X) \times_{K[X]} T(X) \mod P$

- $(K[X]/(P), +, \times)$ is thus a commutative ring, called the *quotient ring* of $K[X]$ by P.

Property

$K[X]/(P)$ is a field iff P is irreducible over $K[X]$.

Galois fields

Algebraic extensions

Consider a field $(K, +, \times)$, and a polynomial $P \in K[X]$ of degree d.

- $K[X]/(P)$ is the set of equivalence classes of $K[X]$ modulo P.
- This is the set of the $P \in K[X]$ with degree $< d$ equipped with the following laws

  Addition: $S + T = S(X) +_{K[X]} T(X) \mod P$
  Multiplication: $S \times T = S(X) \times_{K[X]} T(X) \mod P$

- $(K[X]/(P), +, \times)$ is thus a commutative ring, called the *quotient ring* of $K[X]$ by P.

Property

$K[X]/(P)$ is a field iff P is irreducible over $K[X]$.

Proof.

For all $S \in K[X]/(P)$, GCD$(S, P) = 1$ hence $\exists U, V, US + VP = 1$ thus S is invertible and $U = S^{-1} \mod P$. □

Extension fields (Galois fields)

Example

Over $(\mathbb{Z}/2\mathbb{Z})[X]$, let $P = (X+1)(X^2+X+1)$ (non-irreducible).

- Then $(\mathbb{Z}/2\mathbb{Z})[X]/(P)$ is not a field: $X+1$ is not invertible since $(X+1)(X^2+X+1) = 0$

Extension fields (Galois fields)

Example

Over $(\mathbb{Z}/2\mathbb{Z})[X]$, let $P = (X+1)(X^2+X+1)$ (non-irreducible).

- Then $(\mathbb{Z}/2\mathbb{Z})[X]/(P)$ is not a field: $X+1$ is not invertible since $(X+1)(X^2+X+1) = 0$
- But $(\mathbb{Z}/2\mathbb{Z})[X]/(X^2+X+1)$ is a field since X^2+X+1 is irreducible. Its elements are $\{0, 1, X, X+1\}$

Extension fields (Galois fields)

Example

Over $(\mathbb{Z}/2\mathbb{Z})[X]$, let $P = (X+1)(X^2+X+1)$ (non-irreducible).

- Then $(\mathbb{Z}/2\mathbb{Z})[X]/(P)$ is not a field: $X+1$ is not invertible since $(X+1)(X^2+X+1) = 0$
- But $(\mathbb{Z}/2\mathbb{Z})[X]/(X^2+X+1)$ is a field since X^2+X+1 is irreducible. Its elements are $\{0, 1, X, X+1\}$

Remark

This is a new finite field, with 4 elements (not of the form $\mathbb{Z}/p\mathbb{Z}$ since $p = 4$ is not prime)

Finite fields

Property

Any finite field has a p^k elements where p is prime and $k \in \mathbb{Z}_{>0}$.
p is called the characteristic of the field.

Finite fields

Property

Any finite field has a p^k elements where p is prime and $k \in \mathbb{Z}_{>0}$.
p is called the characteristic of the field.

Up to an isomorphism, all the finite fields are thus

- either the $\mathbb{Z}/p\mathbb{Z}$ with p a prime number
- or the $\mathbb{F}_p[X]/(Q)$ with p a prime number and Q an irreducible polynomial of degree k over $\mathbb{F}_p[X]$.

Finite fields

Property

Any finite field has a p^k elements where p is prime and $k \in \mathbb{Z}_{>0}$.
p is called the characteristic of the field.

Up to an isomorphism, all the finite fields are thus

- either the $\mathbb{Z}/p\mathbb{Z}$ with p a prime number
- or the $\mathbb{F}_p[X]/(Q)$ with p a prime number and Q an irreducible polynomial of degree k over $\mathbb{F}_p[X]$.

Notation

$\mathbb{F}_q$ denotes the finite field with q elements (q is necessarily of the form $q = p^k$ with p prime and $k \in \mathbb{Z}_{>0}$)

- $\mathbb{F}_p = \mathbb{Z}_p$ when p is prime
- $\mathbb{F}_{p^k} = \mathbb{Z}_p[X]/(Q)$ for p prime and $k = \deg Q$

Multiplicative group of a finite field

Property

The multiplicative group $G = (\mathbb{F}_{p^k})^*$ is cyclic

Multiplicative group of a finite field

Property

The multiplicative group $G = (\mathbb{F}_{p^k})^*$ is cyclic

Proof.

Let $q = p^k$. Let e, be the smallest positive integer s.t. $\forall x \in G\, x^e = 1$.
Thus $X^e - 1$ has $q - 1$ roots in $\mathbb{F}_{p^k}$.
Thus $e \geq q - 1$.
Hence there exists an element $g \in G$ of order e generating all elements of G. □

Multiplicative group of a finite field

Property

The multiplicative group $G = (\mathbb{F}_{p^k})^*$ is cyclic

Proof.

Let $q = p^k$. Let e, be the smallest positive integer s.t. $\forall x \in G\, x^e = 1$.
Thus $X^e - 1$ has $q - 1$ roots in $\mathbb{F}_{p^k}$.
Thus $e \geq q - 1$.
Hence there exists an element $g \in G$ of order e generating all elements of G. □

Definition

The generators of the cyclic group $(\mathbb{F}_{p^k})^*$ are called **primitive elements**.

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \quad \forall 0 < i < p^k - 1 \end{cases}$$

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \quad \forall 0 < i < p^k - 1 \end{cases}$$

- $\mathbb{F}_{p^k}$ is $\mathbb{F}_p$ to which a primitive $(p^k - 1)$-th root of unity has been added (and all elements induced by the $+$ and $\times$ laws). **Denoted by** $\mathbb{F}_p(\alpha)$.

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \quad \forall 0 < i < p^k - 1 \end{cases}$$

- $\mathbb{F}_{p^k}$ is $\mathbb{F}_p$ to which a primitive $(p^k - 1)$-th root of unity has been added (and all elements induced by the $+$ and $\times$ laws). **Denoted by** $\mathbb{F}_p(\alpha)$.
- Let $f_\alpha = X^k - f_{k-1} - \cdots - f_0$ be the **minimal polynomial** of α: the monic polynomial $f \in \mathbb{F}_p[X]$ with least degree such that $f(\alpha) = 0 (\Leftrightarrow \alpha^k = f_{k-1}\alpha^{k-1} + \cdots + f_0)$.
  Then $\mathbb{F}_p(\alpha) \equiv \mathbb{F}_p[X]/f$

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \ \ \forall 0 < i < p^k - 1 \end{cases}$$

- $\mathbb{F}_{p^k}$ is $\mathbb{F}_p$ to which a primitive $(p^k - 1)$-th root of unity has been added (and all elements induced by the $+$ and $\times$ laws). **Denoted by** $\mathbb{F}_p(\alpha)$.
- Let $f_\alpha = X^k - f_{k-1} - \cdots - f_0$ be the **minimal polynomial** of α: the monic polynomial $f \in \mathbb{F}_p[X]$ with least degree such that $f(\alpha) = 0 (\Leftrightarrow \alpha^k = f_{k-1}\alpha^{k-1} + \cdots + f_0)$.
  Then $\mathbb{F}_p(\alpha) \equiv \mathbb{F}_p[X]/f$
- Reciprocally, all construction of the form $\mathbb{F}_{p^k} \equiv \mathbb{F}_p[X]/f$ does not necessarily imply that X generates $(\mathbb{F}_{p^k})^*$.

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \ \ \forall 0 < i < p^k - 1 \end{cases}$$

- $\mathbb{F}_{p^k}$ is $\mathbb{F}_p$ to which a primitive $(p^k - 1)$-th root of unity has been added (and all elements induced by the $+$ and $\times$ laws). **Denoted by** $\mathbb{F}_p(\alpha)$.
- Let $f_\alpha = X^k - f_{k-1} - \cdots - f_0$ be the **minimal polynomial** of α: the monic polynomial $f \in \mathbb{F}_p[X]$ with least degree such that $f(\alpha) = 0 (\Leftrightarrow \alpha^k = f_{k-1}\alpha^{k-1} + \cdots + f_0)$.
  Then $\mathbb{F}_p(\alpha) \equiv \mathbb{F}_p[X]/f$
- Reciprocally, all construction of the form $\mathbb{F}_{p^k} \equiv \mathbb{F}_p[X]/f$ does not necessarily imply that X generates $(\mathbb{F}_{p^k})^*$.
- Those f which satisfy this property (X generates $(\mathbb{F}_{p^k})^*$) are called **primitive polynomials**

Primitive elements and polynomials

- A primitive element $\alpha \in \mathbb{F}_{p^k}^*$ has order $p^k - 1$;
- it is a **primitive** $(p^k - 1)$**-th root of unity**:

$$\begin{cases} \alpha^{p^k-1} & = & 1 \\ \alpha^i & \neq & 1 \ \ \forall 0 < i < p^k - 1 \end{cases}$$

- $\mathbb{F}_{p^k}$ is $\mathbb{F}_p$ to which a primitive $(p^k - 1)$-th root of unity has been added (and all elements induced by the $+$ and $\times$ laws). **Denoted by** $\mathbb{F}_p(\alpha)$.
- Let $f_\alpha = X^k - f_{k-1} - \cdots - f_0$ be the **minimal polynomial** of α: the monic polynomial $f \in \mathbb{F}_p[X]$ with least degree such that $f(\alpha) = 0 (\Leftrightarrow \alpha^k = f_{k-1}\alpha^{k-1} + \cdots + f_0)$.
  Then $\mathbb{F}_p(\alpha) \equiv \mathbb{F}_p[X]/f$
- Reciprocally, all construction of the form $\mathbb{F}_{p^k} \equiv \mathbb{F}_p[X]/f$ does not necessarily imply that X generates $(\mathbb{F}_{p^k})^*$.
- Those f which satisfy this property (X generates $(\mathbb{F}_{p^k})^*$) are called **primitive polynomials**

The Galois fields in practice

Essentially 2 types of implementations:

- polynomial
- logarithmic

The polynomial representation

Simply using the arithmetic of $\mathbb{F}_p[X]$ modulo Q:

- Every element is a polynomial of degree $< k$ with coeffs over $\mathbb{F}_p$
  $\rightsquigarrow$ array of size k of elements of $\mathbb{Z}/p\mathbb{Z}$
  - see representation of $\mathbb{Z}/p\mathbb{Z}$ for the type of the coefficients (`uint64_t, float, double, ...`)
  - Case of $p = 2$: bit-packing technique (see next slide)

The Galois fields in practice

Essentially 2 types of implementations:

- polynomial
- logarithmic

The polynomial representation

Simply using the arithmetic of $\mathbb{F}_p[X]$ modulo Q:

- Every element is a polynomial of degree $< k$ with coeffs over $\mathbb{F}_p$
  $\rightsquigarrow$ array of size k of elements of $\mathbb{Z}/p\mathbb{Z}$
  - see representation of $\mathbb{Z}/p\mathbb{Z}$ for the type of the coefficients (`uint64_t, float, double, ...`)
  - Case of $p = 2$: bit-packing technique (see next slide)
- Addition: remains of degree $< k \rightsquigarrow$ just arithmetic over $\mathbb{Z}/p\mathbb{Z}$
- Mutliplication: $S \times T \mod Q \rightsquigarrow$ euclidean division by Q.

Bit-packing for binary fields

If $p = 2$:

- 1 bit = $\mathbb{F}_2$
- 1 byte = $(\mathbb{F}_2)^8 \equiv \mathbb{F}_{2^8}$
- 1 `uint64_t` = $(\mathbb{F}_2)^{64} \equiv \mathbb{F}_{2^{64}}$, etc

Bit-packing for binary fields

If $p = 2$:

- 1 bit = $\mathbb{F}_2$
- 1 byte $= (\mathbb{F}_2)^8 \equiv \mathbb{F}_{2^8}$
- 1 `uint64_t` $= (\mathbb{F}_2)^{64} \equiv \mathbb{F}_{2^{64}}$, etc

For instance $\mathbb{F}_{2^8}$

- `char a`: the binary repr. of `a` is the coefficient vector of $P \in \mathbb{F}_2[X]$ of degree ≤ 7 s. t. $P(2) = a$

a	0	1	2	3	4	5	...
in binary	000000000	000000001	00000010	00000011	00000100	00000101	...
represents	0	1	x	$x+1$	x^2	x^2+1	...

Bit-packing for binary fields

If $p = 2$:

- 1 bit = $\mathbb{F}_2$
- 1 byte $= (\mathbb{F}_2)^8 \equiv \mathbb{F}_{2^8}$
- 1 `uint64_t` $= (\mathbb{F}_2)^{64} \equiv \mathbb{F}_{2^{64}}$, etc

For instance $\mathbb{F}_{2^8}$

- `char a`: the binary repr. of `a` is the coefficient vector of $P \in \mathbb{F}_2[X]$ of degree ≤ 7 s. t. $P(2) = a$

a	0	1	2	3	4	5	...
in binary	000000000	000000001	00000010	00000011	00000100	00000101	...
represents	0	1	x	$x+1$	x^2	x^2+1	...

- addition: bitwise XOR: `a` $\wedge$ `b`

Bit-packing for binary fields

If $p = 2$:

- 1 bit = $\mathbb{F}_2$
- 1 byte $= (\mathbb{F}_2)^8 \equiv \mathbb{F}_{2^8}$
- 1 `uint64_t` $= (\mathbb{F}_2)^{64} \equiv \mathbb{F}_{2^{64}}$, etc

For instance $\mathbb{F}_{2^8}$

- `char a`: the binary repr. of `a` is the coefficient vector of $P \in \mathbb{F}_2[X]$ of degree ≤ 7 s. t. $P(2) = a$

a	0	1	2	3	4	5	...
in binary	000000000	000000001	00000010	00000011	00000100	00000101	...
represents	0	1	x	$x+1$	x^2	x^2+1	...

- addition: bitwise XOR: `a` $\wedge$ `b`
- mult: iterated application of `mulByX`

```
char mulByX (char a){
  char b = a<<1;
  if (a & 128) b ^= 29
  return b;
}
```

Logarithmic representation (Zech-log)

- Choose a generator g of $(\mathbb{F}_q)^*$
- Each element $a \neq 0$ is represented by its discrete log. i s.t. $a = g^i$.
- $a = 0$ is represented by a special value (e.g. $q - 1$)

Logarithmic representation (Zech-log)

- Choose a generator g of $(\mathbb{F}_q)^*$
- Each element $a \neq 0$ is represented by its discrete log. i s.t. $a = g^i$.
- $a = 0$ is represented by a special value (e.g. $q - 1$)
- multiplication: $a \times b = g^i \times g^j = g^{i+j}$
  $\rightsquigarrow$ addition of the indices $\mod q - 1$
  $\rightsquigarrow$ requires to store conversion tables $i \mapsto g^i$ and $j = g^i \mapsto i$

Logarithmic representation (Zech-log)

- Choose a generator g of $(\mathbb{F}_q)^*$
- Each element $a \neq 0$ is represented by its discrete log. i s.t. $a = g^i$.
- $a = 0$ is represented by a special value (e.g. $q - 1$)
- multiplication: $a \times b = g^i \times g^j = g^{i+j}$
  $\rightsquigarrow$ addition of the indices $\mod q - 1$
  $\rightsquigarrow$ requires to store conversion tables $i \mapsto g^i$ and $j = g^i \mapsto i$
- addition: $g^i + g^j = g^i \times (1 + g^{j-i})$
  $\rightsquigarrow$ requires to also store $k \mapsto \ell$ s.t. $g^\ell = 1 + g^k$

Logarithmic representation (Zech-log)

- Choose a generator g of $(\mathbb{F}_q)^*$
- Each element $a \neq 0$ is represented by its discrete log. i s.t. $a = g^i$.
- $a = 0$ is represented by a special value (e.g. $q - 1$)
- multiplication: $a \times b = g^i \times g^j = g^{i+j}$
  $\rightsquigarrow$ addition of the indices $\mod q - 1$
  $\rightsquigarrow$ requires to store conversion tables $i \mapsto g^i$ and $j = g^i \mapsto i$
- addition: $g^i + g^j = g^i \times (1 + g^{j-i})$
  $\rightsquigarrow$ requires to also store $k \mapsto \ell$ s.t. $g^\ell = 1 + g^k$

Exercise

Write the algorithm for the addition, using a precomputed table

Logarithmic representation (Zech-log)

- Choose a generator g of $(\mathbb{F}_q)^*$
- Each element $a \neq 0$ is represented by its discrete log. i s.t. $a = g^i$.
- $a = 0$ is represented by a special value (e.g. $q - 1$)
- multiplication: $a \times b = g^i \times g^j = g^{i+j}$
  $\rightsquigarrow$ addition of the indices $\mod q - 1$
  $\rightsquigarrow$ requires to store conversion tables $i \mapsto g^i$ and $j = g^i \mapsto i$
- addition: $g^i + g^j = g^i \times (1 + g^{j-i})$
  $\rightsquigarrow$ requires to also store $k \mapsto \ell$ s.t. $g^\ell = 1 + g^k$

Exercise

Write the algorithm for the addition, using a precomputed table

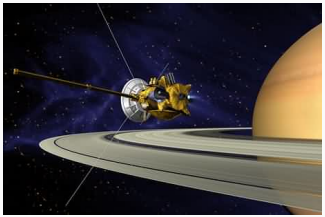
Choosing a good generator

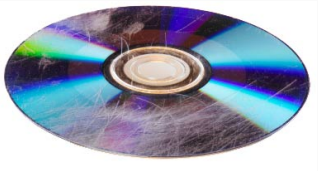
X is a simpler generator to compute with.
$\rightsquigarrow$ the polyn. Q such that $(\mathbb{F}_p[X]/(Q))^*$ is generated by X are the **primitive polynomials**

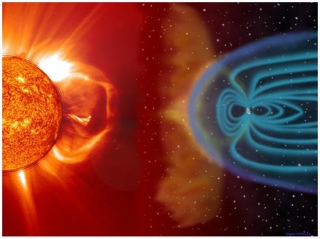
Outline

Outline

Errors everywhere







Error models

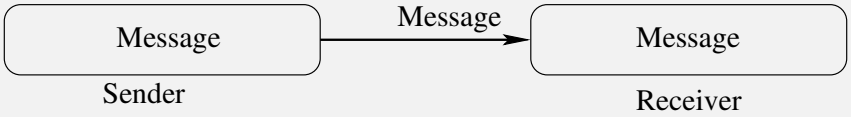
Communication channel

• Radio transmission	electromagnetic interferences
• Ethernet, DSL	electromagnetic interferences
• CD/DVD Audio/Video/ROM	scratches, dust
• RAM	cosmic radiations
• HDD	magnetic field, crash

Error models

Communication channel

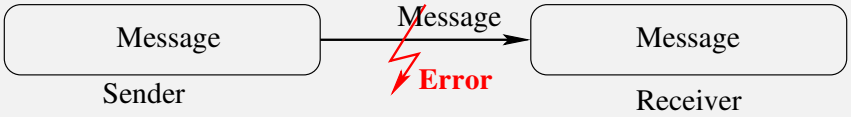
- Radio transmission — electromagnetic interferences
- Ethernet, DSL — electromagnetic interferences
- CD/DVD Audio/Video/ROM — scratches, dust
- RAM — cosmic radiations
- HDD — magnetic field, crash



Error models

Communication channel

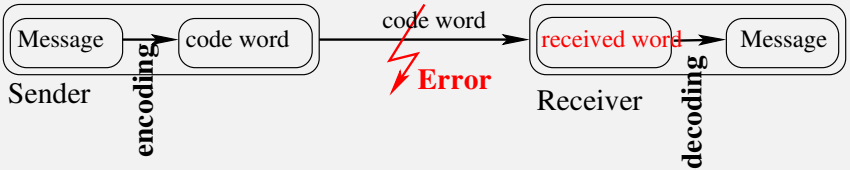
- Radio transmission — electromagnetic interferences
- Ethernet, DSL — electromagnetic interferences
- CD/DVD Audio/Video/ROM — scratches, dust
- RAM — cosmic radiations
- HDD — magnetic field, crash



Error models

Communication channel

• Radio transmission	electromagnetic interferences
• Ethernet, DSL	electromagnetic interferences
• CD/DVD Audio/Video/ROM	scratches, dust
• RAM	cosmic radiations
• HDD	magnetic field, crash



Generalities on coding theory

Goals:

Detect: require retransmission (integrity certificate)

Correct: i.e. when no interraction possible

Tool: Adding redundancy

Generalities on coding theory

Goals:

Detect: require retransmission (integrity certificate)

Correct: i.e. when no interraction possible

Tool: Adding redundancy

Example (NATO phonetic alphabet)

A $\rightarrow$ Alfa, B $\rightarrow$ Bravo, C $\rightarrow$ Charlie, D $\rightarrow$ Delta ...

Alpha Bravo India Tango Tango Echo Delta India Oscar Uniform Sierra !

Generalities on coding theory

Goals:

Detect: require retransmission (integrity certificate)
Correct: i.e. when no interraction possible

Tool: Adding redundancy

Example (NATO phonetic alphabet)

A $\rightarrow$ Alfa, B $\rightarrow$ Bravo, C $\rightarrow$ Charlie, D $\rightarrow$ Delta ...

Alpha Bravo India Tango Tango Echo Delta India Oscar Uniform Sierra !

Two categories of codes:

stream codes: online processing of the stream of information
block codes: cutting information in blocks and applying the same treatment to each block

Generalities on coding theory

Goals:

Detect: require retransmission (integrity certificate)
Correct: i.e. when no interraction possible

Tool: Adding redundancy

Example (NATO phonetic alphabet)

A $\rightarrow$ Alfa, B $\rightarrow$ Bravo, C $\rightarrow$ Charlie, D $\rightarrow$ Delta ...

Alpha Bravo India Tango Tango Echo Delta India Oscar Uniform Sierra !

Two categories of codes:

stream codes: online processing of the stream of information
block codes: cutting information in blocks and applying the same treatment to each block

Generalities and terminology

- A code is a sub-set $\mathcal{C} \subset \mathcal{E}$ of a set of possible words.
- Often, $\mathcal{E}$ is built from an alphabet Σ: $\mathcal{E} = \Sigma^n$.
- Encoding function: $E : \mathcal{S} \to \mathcal{E}$ such that $E(\mathcal{S}) = \mathcal{C}$.
- Informally, a code is
  - t-detector, if any set error on t symbols can be detected
  - t-corrector, if any set error on t symbols can be corrected

Examples

Parity check

$$\begin{array}{rccc} E: & (\mathbb{Z}/2\mathbb{Z})^3 & \to & (\mathbb{Z}/2\mathbb{Z})^4 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, x_3, s) \end{array}$$

with $s = \sum_{i=1}^{3} x_i \mod 2 \rightsquigarrow \sum_{i=1}^{3} x_i + s = 0 \mod 2$

Examples

Parity check

$$\begin{array}{rccc} E: & (\mathbb{Z}/2\mathbb{Z})^3 & \to & (\mathbb{Z}/2\mathbb{Z})^4 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, x_3, s) \end{array}$$

(x_1, x_2', x_3, s) with $x_2' \neq x_2 \to x_1 + x_2' + x_3 + s = 1 \rightsquigarrow$ Error detected
with $s = \sum_{i=1}^{3} x_i \mod 2 \rightsquigarrow \sum_{i=1}^{3} x_i + s = 0 \mod 2$

Examples

Parity check

$$\begin{array}{rccc} E: & (\mathbb{Z}/2\mathbb{Z})^3 & \to & (\mathbb{Z}/2\mathbb{Z})^4 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, x_3, s) \end{array}$$

$\rightsquigarrow$ Error detected

with $s = \sum_{i=1}^{3} x_i \mod 2 \rightsquigarrow \sum_{i=1}^{3} x_i + s = 0 \mod 2$

Repetition code

- “Say that again?”

Examples

Parity check

$$\begin{array}{rccc} E: & (\mathbb{Z}/2\mathbb{Z})^3 & \to & (\mathbb{Z}/2\mathbb{Z})^4 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, x_3, s) \end{array}$$

$\rightsquigarrow$ Error detected

with $s = \sum_{i=1}^{3} x_i \mod 2 \rightsquigarrow \sum_{i=1}^{3} x_i + s = 0 \mod 2$

Repetition code

- “Say that again?”
- “1” $\to$ “111”$\to$ “110”$\to$ “111” $\to$ “1”

Examples

Parity check

$$\begin{array}{rccc} E: & (\mathbb{Z}/2\mathbb{Z})^3 & \to & (\mathbb{Z}/2\mathbb{Z})^4 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, x_3, s) \end{array}$$

$\rightsquigarrow$ Error detected

with $s = \sum_{i=1}^{3} x_i \mod 2 \rightsquigarrow \sum_{i=1}^{3} x_i + s = 0 \mod 2$

Repetition code

- "Say that again?"
- "1" $\to$ "111"$\to$ "110"$\to$ "111" $\to$ "1"

$$\begin{array}{rccc} E: & \mathbb{Z}/2\mathbb{Z} & \longrightarrow & (\mathbb{Z}/2\mathbb{Z})^r \\ & x & \longmapsto & (\underbrace{x, \ldots, x}_{r \text{ times}}) \end{array},$$

and $\mathcal{C} = \mathrm{Im}(E) \subset (\mathbb{Z}/2\mathbb{Z})^r$

Outline

Linear Codes

Let $\mathcal{E} = \mathbb{F}_q^n$ over a finite field $\mathbb{F}_q$.
A linear code $\mathcal{C}$ is a subspace of $\mathcal{E}$.

- length: n
- dimension: $k = \dim(\mathcal{C})$
- Rate (of information): k/n

Encoding function: $E : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ s.t. $\mathcal{C} = \mathrm{Im}(E) \subset \mathbb{F}_q^n$

Linear Codes

Let $\mathcal{E} = \mathbb{F}_q^n$ over a finite field $\mathbb{F}_q$.
A linear code $\mathcal{C}$ is a subspace of $\mathcal{E}$.

- length: n
- dimension: $k = \dim(\mathcal{C})$
- Rate (of information): k/n

Encoding function: $E : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ s.t. $\mathcal{C} = \mathrm{Im}(E) \subset \mathbb{F}_q^n$

Example

- Parity code: $k = n - 1$ — 1-detector
- r-repetition code: $k = r/r = 1$ — $r-1$-detector, $\lfloor \frac{r-1}{2} \rfloor$-corrector

Distance of a code

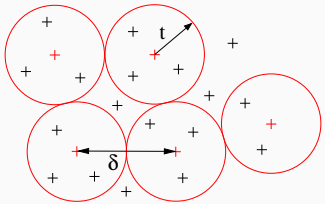
- Hamming weight: $w_H(x) = |\{i, x_i \neq 0\}|$.

Distance of a code

- Hamming weight: $w_H(x) = |\{i, x_i \neq 0\}|$.
- Hamming distance: $d_H(x, y) = w_H(x - y) = |\{i, x_i \neq y_i\}|$

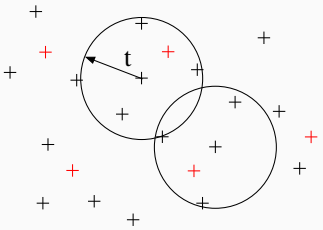
Distance of a code

- Hamming weight: $w_H(x) = |\{i, x_i \neq 0\}|$.
- Hamming distance: $d_H(x, y) = w_H(x - y) = |\{i, x_i \neq y_i\}|$
- Minimum distance of a code $\delta = \min_{x,y \in \mathcal{C}} d_H(x, y)$
  In a linear code: $\delta = \min_{x \in \mathcal{C} \setminus \{0\}} w_H(x))$



Distance of a code

- Hamming weight: $w_H(x) = |\{i, x_i \neq 0\}|$.
- Hamming distance: $d_H(x, y) = w_H(x - y) = |\{i, x_i \neq y_i\}|$
- Minimum distance of a code $\delta = \min_{x,y \in \mathcal{C}} d_H(x, y)$
  In a linear code: $\delta = \min_{x \in \mathcal{C} \setminus \{0\}} w_H(x))$

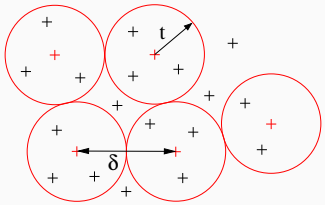


$\mathcal{C}$ is t-corrector if

- $\forall x \in \mathcal{E}\ |\{c \in \mathcal{C}, d_H(x, c) \leq t\}| \leq 1$

Distance of a code

- Hamming weight: $w_H(x) = |\{i, x_i \neq 0\}|$.
- Hamming distance: $d_H(x, y) = w_H(x - y) = |\{i, x_i \neq y_i\}|$
- Minimum distance of a code $\delta = \min_{x,y \in \mathcal{C}} d_H(x, y)$
  In a linear code: $\delta = \min_{x \in \mathcal{C} \setminus \{0\}} w_H(x))$



$\mathcal{C}$ is t-corrector if

- $\forall x \in \mathcal{E}\ |\{c \in \mathcal{C}, d_H(x, c) \leq t\}| \leq 1$
- $\forall c_1, c_2 \in \mathcal{C}\ c_1 \neq c_2 \Rightarrow d_H(c_1, c_2) > 2t$

Perfect codes

Definition

A code is perfect if any detected error can be corrected.

Example

- 4-repetition is not perfect
- 3-repetition is perfect

Perfect codes

Definition

A code is perfect if any detected error can be corrected.

Example

- 4-repetition is not perfect
- 3-repetition is perfect

Property

A code is perfect if the balls of radius t around the codewords form a partition of the ambiant space.

Perfect codes

Definition

A code is perfect if any detected error can be corrected.

Example

- 4-repetition is not perfect
- 3-repetition is perfect

Property

A code is perfect if the balls of radius t around the codewords form a partition of the ambiant space.

Remark

Can be corrected into the wrong code-word. For instance $(b, a, b) \to (b, b, b)$

Generator matrix and parity check matrix

Generator matrix

- The matrix G of the encoding function (depends on a choice of basis):

$$E : x^T \longrightarrow x^T G$$

- Under systematic form: $G = \left[\begin{array}{ccc|c} 1 & & 0 & \\ & \ddots & & \overline{G} \\ 0 & & 1 & \end{array}\right]$

Generator matrix and parity check matrix

Generator matrix

- The matrix G of the encoding function (depends on a choice of basis):

$$E : x^T \longrightarrow x^T G$$

- Under systematic form: $G = \left[\begin{array}{ccc|c} 1 & & 0 & \\ & \ddots & & \overline{G} \\ 0 & & 1 & \end{array}\right]$

Parity check matrix

1. A matrix $H \in K^{(n-k)\times n}$ such that $\ker(H) = \mathcal{C}$:

$$c \in \mathcal{C} \Leftrightarrow Hc = 0$$

2. A basis of $\ker(G^T)$: $HG^T = 0$

Generator matrix and parity check matrix

Exercise

Find G and H of the parity check and of the k-repetition codes.

Generator matrix and parity check matrix

Exercise

Find G and H of the parity check and of the k-repetition codes.

$$G_{par} = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix}, H_{par} = [1 \ \dots \ 1]$$

Generator matrix and parity check matrix

Exercise

Find G and H of the parity check and of the k-repetition codes.

$$G_{par} = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix}, H_{par} = [1 \ \dots \ 1]$$

$$G_{rep} = [1 \ \dots \ 1] = H_{par}, H_{rep} = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix} = G_{par}$$

*The parity check code is the **dual** of the repetition code*

Generator matrix and parity check matrix

Exercise

Find G and H of the parity check and of the k-repetition codes.

$$G_{par} = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix}, H_{par} = [1 \ \dots \ 1]$$

$$G_{rep} = [1 \ \dots \ 1] = H_{par}, H_{rep} = \begin{bmatrix} 1 & & 1 \\ & \ddots & \vdots \\ & 1 & 1 \end{bmatrix} = G_{par}$$

*The parity check code is the **dual** of the repetition code*

Definition

Let $\mathcal{C}$ be a linear code with generating matrix G and parity check matrix H.
The dual code $\mathcal{D}$ of $\mathcal{C}$ is the linear code with generating matrix H and parity check matrix G.

Role of the parity check matrix

$$c \in \mathcal{C} \Leftrightarrow Hc = 0$$

- Certificate for detecting errors
- Syndrom: $s_x = Hx = H(c + e) = He$

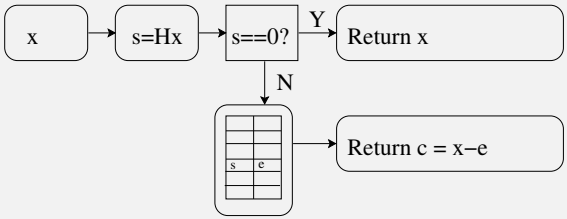
Role of the parity check matrix

$$c \in \mathcal{C} \Leftrightarrow Hc = 0$$

- Certificate for detecting errors
- Syndrom: $s_x = Hx = H(c + e) = He$

A first correction algorithm:

- pre-compute all s_e for $w_H(e) \leq t$ in a table S
- For x received. If $s_x \neq 0$, look for s_x in the table S
- return the corresponding codeword



Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code?
- Generator matrix?
- Minimal distance?
- Is it a perfect code?

Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code? $(n, k) = (7, 4)$
- Generator matrix?
- Minimal distance?
- Is it a perfect code?

Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code? $(n, k) = (7, 4)$
- Generator matrix? $G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- Minimal distance?
- Is it a perfect code?

Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code? $(n, k) = (7, 4)$
- Generator matrix? $G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- Minimal distance? $\delta \leq 3$.
  - If $\delta = 1$, $\exists i, H_i = 0$
  - If $\delta = 2$, $\exists i \neq j, H_i = H_j \rightsquigarrow \delta = 3$
- Is it a perfect code?

Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code? $(n, k) = (7, 4)$
- Generator matrix? $G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- Minimal distance? $\delta \leq 3$.
  - If $\delta = 1$, $\exists i, H_i = 0$
  - If $\delta = 2$, $\exists i \neq j, H_i = H_j \rightsquigarrow \delta = 3$
- Is it a perfect code? $\delta = 3 \rightsquigarrow t = 1$ corrector.
  $|\mathcal{C}| = 2^k \rightsquigarrow \#$ of elts in each ball of radius 1: $2^k(1 + 7) = 2^7 = |K^n| \rightsquigarrow$ perfect

Hamming codes

Let $H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

- Parameters of the corresponding code? $(n, k) = (7, 4)$
- Generator matrix? $G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- Minimal distance? $\delta \leq 3$.
  - If $\delta = 1$, $\exists i, H_i = 0$
  - If $\delta = 2$, $\exists i \neq j, H_i = H_j \rightsquigarrow \delta = 3$
- Is it a perfect code? $\delta = 3 \rightsquigarrow t = 1$ corrector.
  $|\mathcal{C}| = 2^k \rightsquigarrow$ # of elts in each ball of radius 1: $2^k(1 + 7) = 2^7 = |K^n| \rightsquigarrow$ perfect

Generalization

$\forall \ell$: $H(2^\ell - 1, 2^\ell - \ell)$, is 1-corrector, perfect. Example: Minitel, ECC memory: $\ell = 7$

Some bounds

Let $\mathcal{C}$ be a code (n, k, δ) over a field $\mathbb{F}_q$ with q elements.

k and δ can not be simulatneously large for a given n.

Sphere packing:

$$q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n, \text{ with } t = \lfloor \frac{\delta - 1}{2} \rfloor.$$

Some bounds

Let $\mathcal{C}$ be a code (n, k, δ) over a field $\mathbb{F}_q$ with q elements.

k and δ can not be simulatneously large for a given n.

Sphere packing:

$$q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n, \text{ with } t = \lfloor \frac{\delta - 1}{2} \rfloor.$$

Singleton bound:

$$\delta \leq n - k + 1$$

Some bounds

Let $\mathcal{C}$ be a code (n, k, δ) over a field $\mathbb{F}_q$ with q elements.

k and δ can not be simulatneously large for a given n.

Sphere packing:

$$q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n, \text{ with } t = \lfloor \frac{\delta - 1}{2} \rfloor.$$

Singleton bound:

$$\delta \leq n - k + 1$$

Sketch of proof:

- Let H be the parity check matrix $(n-k) \times n$.
- δ is the smallest number of linearly dependent cols of H.
- $n - k + 1 = \text{rank}(H) + 1$ cols are always linearly dependent.

Some bounds

Let $\mathcal{C}$ be a code (n, k, δ) over a field $\mathbb{F}_q$ with q elements.

k and δ can not be simulatneously large for a given n.

Sphere packing:

$$q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n, \text{ with } t = \lfloor \frac{\delta - 1}{2} \rfloor.$$

Singleton bound:

$$\delta \leq n - k + 1$$

Sketch of proof:

- Let H be the parity check matrix $(n-k) \times n$.
- δ is the smallest number of linearly dependent cols of H.
- $n - k + 1 = \text{rank}(H) + 1$ cols are always linearly dependent.

Outline

Evaluation-interpolation codes

Theorem (Interpolation)

For all $x_1, \ldots, x_k$, distincts, and all $y_1, \ldots, y_k$, there is a unique polynomial $f = f_0 + f_1 x + \ldots f_{k-1} x^{k-1}$ of degree $< k$ such that :

$$f(x_j) = y_j, \quad \text{for all } 1 \leq j \leq k.$$

Evaluation-interpolation codes

Theorem (Interpolation)

For all $x_1, \ldots, x_k$, distincts, and all $y_1, \ldots, y_k$, there is a unique polynomial $f = f_0 + f_1 x + \ldots f_{k-1} x^{k-1}$ of degree $< k$ such that :

$$f(x_j) = y_j, \quad \text{for all } 1 \leq j \leq k.$$

Corollary

For some fixed x_i's

- *equivalent representation:* $(y_1, \ldots, y_k) \Leftrightarrow (f_0, \ldots, f_{k-1})$.
- *oversampling:* $(y_1, \ldots, y_k, y_{k+1}, \ldots, y_n) \Leftarrow (f_0, \ldots, f_{k-1})$.

  $\rightsquigarrow$ *adding redundancy*

Reed-Solomon codes

Definition (Reed-Solomon codes)

Let K be a finite field, and $x_1, \ldots, x_n \in K$ distinct elements. The Reed-Solomon code of length n and dimension k is defined by

$$\mathcal{C}(n,k) = \{(f(x_1), \ldots, f(x_n)), f \in K[X]; \deg f < k\}$$

Reed-Solomon codes

Definition (Reed-Solomon codes)

Let K be a finite field, and $x_1, \ldots, x_n \in K$ distinct elements. The Reed-Solomon code of length n and dimension k is defined by

$$\mathcal{C}(n,k) = \{(f(x_1), \ldots, f(x_n)), f \in K[X]; \deg f < k\}$$

Example

$(n,k) = (5,3), f = x^2 + 2x + 1$ over $\mathbb{Z}/19\mathbb{Z}$.

$(1,2,1,0,0) \xrightarrow{Eval} (f(1), f(5), f(8), f(10), f(12)) = (4,5,17,5,7,17)$

$(4,17,5,7,17) \xrightarrow{Interp.} (1,2,1,0,0)$ $\qquad x^2 + 2x + 1$

$(4,17,13,7,17) \xrightarrow{Interp.} (12,8,11,10,1)$ $\qquad x^4 + 10x^3 + 11x^2 + 8x + 12$

Minimal distance of Reed-Solomon codes

Property

$\delta = n - k + 1$ *(Maximum Distance Separable codes)*

Minimal distance of Reed-Solomon codes

Property

$\delta = n - k + 1$ *(Maximum Distance Separable codes)*

Proof.

Singeton bound: $\delta \leq n - k + 1$

□

Minimal distance of Reed-Solomon codes

Property

$\delta = n - k + 1$ *(Maximum Distance Separable codes)*

Proof.

Singeton bound: $\delta \leq n - k + 1$

Let $f, g \in \mathcal{C}$: $\deg f, \deg g < k$.

If $f(x_i) \neq g(x_i)$ for $d < n - k + 1$ values x_i,

Then $f(x_j) - g(x_j) = 0$ for at least $n - d > k - 1$ values x_j.

Now $\deg(f - g) < k$, hence $f = g$. □

Minimal distance of Reed-Solomon codes

Property

$\delta = n - k + 1$ *(Maximum Distance Separable codes)*

Proof.

Singeton bound: $\delta \leq n - k + 1$

Let $f, g \in \mathcal{C}$: $\deg f, \deg g < k$.

If $f(x_i) \neq g(x_i)$ for $d < n - k + 1$ values x_i,

Then $f(x_j) - g(x_j) = 0$ for at least $n - d > k - 1$ values x_j.

Now $\deg(f - g) < k$, hence $f = g$. □

$\rightsquigarrow$ correct up to $\frac{n-k}{2}$ errors.

Decoding via the key equation

c	$= \mathsf{Eval}(f)$	the codeword
y	$= c + e$	the received word
P	$= \mathsf{Interp}((x_i, y_i))$	the interpolant of the received word
Λ	$= \prod_{i \mid e_i \neq 0}(x - x_i)$	the error locator polynomial

Decoding via the key equation

$$
\begin{aligned}
c &= \mathsf{Eval}(f) && \text{the codeword} \\
y &= c + e && \text{the received word} \\
P &= \mathsf{Interp}((x_i, y_i)) && \text{the interpolant of the received word} \\
\Lambda &= \textstyle\prod_{i|e_i \neq 0}(x - x_i) && \text{the error locator polynomial}
\end{aligned}
$$

Property (Key Equation)

$$\Lambda f = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

Decoding via the key equation

c	$= \mathsf{Eval}(f)$	the codeword
y	$= c + e$	the received word
P	$= \mathsf{Interp}((x_i, y_i))$	the interpolant of the received word
Λ	$= \prod_{i\|e_i \neq 0}(x - x_i)$	the error locator polynomial

Property (Key Equation)

$$\Lambda f = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

Proof.

- correct points: $\forall i$ s.t. $e_i = 0 : f(x_i) = P(x_i)$
- erroneous points: $\forall i$ s.t. $e_i \neq 0 : f(x_i) \neq P(x_i)$ but $\Lambda(x_i) = 0$

Decoding via the key equation

$$
\begin{aligned}
c &= \mathsf{Eval}(f) && \text{the codeword} \\
y &= c + e && \text{the received word} \\
P &= \mathsf{Interp}((x_i, y_i)) && \text{the interpolant of the received word} \\
\Lambda &= \textstyle\prod_{i|e_i \neq 0}(x - x_i) && \text{the error locator polynomial}
\end{aligned}
$$

Property (Key Equation)

$$\Lambda f = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

Proof.

- correct points: $\forall i$ s.t. $e_i = 0 : f(x_i) = P(x_i)$
- erroneous points: $\forall i$ s.t. $e_i \neq 0 : f(x_i) \neq P(x_i)$ but $\Lambda(x_i) = 0$

$\rightsquigarrow \forall i \in \{1 \ldots n\} \Lambda(x_i) f(x_i) = \Lambda(x_i) P(x_i)$

Solving the Key equation

$$\Lambda f = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

- Problem: quadratic equation in the unknwon coeffs of f and Λ

Solving the Key equation

$$N = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

- Problem: quadratic equation in the unknwon coeffs of f and Λ
- Simplifying relaxation: set $N = \Lambda f$ *Linearization*

Berlekamp-Welch decoding

Find N of degree $< k + t$ and Λ monic of degree $\leq t$ s.t.

$$N = \Lambda P \mod \prod_{i=1}^{n} (x - x_i)$$

Linear system solving

$N(X) = n_0 + \ldots n_{k+t-1} X^{k+t-1}$ and $\Lambda(X) = \ell_0 + \cdots + \ell_{t-1} X^{t-1} + X^t$.

Unknonwns: $n_0, \ldots n_{k+t-1}, \ell_0, \ldots, \ell_{t-1}$ ($k + 2t$ unknowns)

Equations: each in x_i (n equations)

Berlekamp-Welch decoding

Find N of degree $< k + t$ and Λ monic of degree $\leq t$ s.t.

$$N = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

Linear system solving

$N(X) = n_0 + \ldots n_{k+t-1}X^{k+t-1}$ and $\Lambda(X) = \ell_0 + \cdots + \ell_{t-1}X^{t-1} + X^t$.

Unknonwns: $n_0, \ldots n_{k+t-1}, \ell_0, \ldots, \ell_{t-1}$ ($k + 2t$ unknowns)

Equations: each in x_i (n equations)

$$\left[\begin{array}{cccc|c} 1 & x_1 & x_1^2 & \ldots & x_1^{k+t-1} \\ 1 & x_2 & x_1^2 & \ldots & x_1^{k+t-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{k+t-1} \end{array}\right] \left[\begin{array}{ccc} -P(x_1) & & \\ & \ddots & \\ & & -P(x_n) \end{array}\right] \left[\begin{array}{cccc} 1 & x_1 & \ldots & x_1^t \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & \ldots & x_n^t \end{array}\right] \left[\begin{array}{c} n_0 \\ \vdots \\ n_{k+t-1} \\ \ell_0 \\ \ldots \\ \ell_{t-1} \\ 1 \end{array}\right] = \left[\begin{array}{c} 0 \\ \vdots \\ 0 \end{array}\right]$$

Berlekamp-Welch decoding

Find N of degree $< k + t$ and Λ monic of degree $\leq t$ s.t.

$$N = \Lambda P \mod \prod_{i=1}^{n}(x - x_i)$$

Linear system solving

$N(X) = n_0 + \ldots n_{k+t-1}X^{k+t-1}$ and $\Lambda(X) = \ell_0 + \cdots + \ell_{t-1}X^{t-1} + X^t$.

Unknonwns: $n_0, \ldots n_{k+t-1}, \ell_0, \ldots, \ell_{t-1}$ ($k + 2t$ unknowns)

Equations: each in x_i (n equations)

$$\left[\begin{array}{ccccc|c} 1 & x_1 & x_1^2 & \ldots & x_1^{k+t-1} & \\ 1 & x_2 & x_1^2 & \ldots & x_1^{k+t-1} & \\ \vdots & \vdots & \vdots & \ldots & \vdots & \\ 1 & x_n & x_n^2 & \ldots & x_n^{k+t-1} & \end{array}\right] \left[\begin{array}{ccc} -P(x_1) & & \\ & \ddots & \\ & & -P(x_n) \end{array}\right] \left[\begin{array}{cccc} 1 & x_1 & \ldots & x_1^{t-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & \ldots & x_n^{t-1} \end{array}\right] \left[\begin{array}{c} n_0 \\ \vdots \\ n_{k+t-1} \\ \ell_0 \\ \ldots \\ \ell_{t-1} \end{array}\right] = \left[\begin{array}{c} P(x_1)x_1^t \\ \vdots \\ P(x_n)x_n^t \end{array}\right]$$

Rational fraction reconstruction

Problem (RFR: Rational Fraction Reconstruction)

Given $A, B \in K[X]$ with $\deg B < \deg A = n$, find $f, g \in K[X]$, such that

$$\begin{cases} f & = & gB \mod A \\ \deg f & \leq & d_F, \\ \deg g & \leq & n - d_F - 1, \end{cases} .$$

Theorem

Let $(f_0 = A, f_1 = B, \ldots, f_\ell)$ the sequence of remainders of the extended Euclidean algorithm applied on (A, B) and u_i, v_i the coefficients s.t. $f_i = u_i f_0 + v_i f_1$. Then, at iteration j s.t. $\deg f_j \leq d_F < \deg f_{j-1}$,

1. *(f_j, v_j) is a solution of problem RFR.*
2. *it is minimal: any other solution (f, g) writes*

$$f = qf_j, \quad g = qv_j \quad \text{for } q \in K[X].$$

Reed-Solomon decoding with Extended Euclidean algorithm

Berlekamp-Welch using extended Euclidean algorithm

- Erroneous interpolant: $P = \text{Interp}((y_i, x_i))$
- Error locator polynomial: $\Lambda = \prod_{i|y_i \text{is erroneous}} (X - x_i)$

Find f with $\deg f \leq d_F$ s.t.. f and P match on $\geq n - t$ evaluations x_i.

$$\underbrace{\Lambda f}_{f_j} = \underbrace{\Lambda}_{g_j} P \mod \prod_{i=1}^{n} (X - x_i)$$

and $(\Lambda f, \Lambda)$ is minimal

$\rightsquigarrow$ computed by extended Euclidean Algorithm

$$f = f_j / g_j.$$

Another decoding algorithm: syndrom based

From now on: $K = \mathbb{F}_q, n = q - 1, x_i = \alpha^i$ where α is a primitive n-th root of unity.

$$E(f) = (f(\alpha^0), f(\alpha^1), f(\alpha^2), \ldots, f(\alpha^{n-1})) = DFT_\alpha(f)$$

Another decoding algorithm: syndrom based

From now on: $K = \mathbb{F}_q, n = q-1, x_i = \alpha^i$ where α is a primitive n-th root of unity.

$$E(f) = (f(\alpha^0), f(\alpha^1), f(\alpha^2), \ldots, f(\alpha^{n-1})) = DFT_\alpha(f)$$

Linear recurring sequences

Sequences $(a_0, a_1, \ldots, a_n, \ldots)$ such that

$$\forall j \geq 0 \; a_{j+t} = \sum_{i=0}^{t-1} \lambda_i a_{i+j}$$

generator polynomial: $\Lambda(z) = z^t - \sum_{i=0}^{t-1} \lambda_i z^i$

minimal polynomial: $\Lambda(z)$ of minimal degree

linear complexity of $(a_i)_i$**:** degree t of the minimal polynomial Λ

Blahut theorem

Theorem ([Blahut84], [Prony1795])

The DFT_α of a vector of weight t has linear complexity t.

Blahut theorem

Theorem ([Blahut84], [Prony1795])

The DFT_α of a vector of weight t has linear complexity t.

Skecth of proof

- Let $v = e_i$ be a 1-weight vector. Then $\mathrm{DFT}_\alpha(v) = \mathrm{Ev}_{(\alpha^0,\alpha^1,\ldots,\alpha^n)}(X^i) = ((\alpha^0)^i, (\alpha^1)^i, \ldots, (\alpha^{n-1})^i)$ is linearly generated by $\Lambda(z) = z - \alpha^i$.

Blahut theorem

Theorem ([Blahut84], [Prony1795])

The DFT$_\alpha$ of a vector of weight t has linear complexity t.

Skecth of proof

- Let $v = e_i$ be a 1-weight vector. Then $\mathrm{DFT}_\alpha(v) = \mathrm{Ev}_{(\alpha^0,\alpha^1,\ldots,\alpha^n)}(X^i) = ((\alpha^0)^i, (\alpha^1)^i, \ldots, (\alpha^{n-1})^i)$ is linearly generated by $\Lambda(z) = z - \alpha^i$.
- For $v = \sum_{j=1}^{t} e_{i_j}$, the sequence $\mathrm{DFT}_\alpha(v)$ is generated by $\mathrm{ppcm}_j(z - \alpha^{i_j}) = \prod_{j=1}^{t}(z - \alpha^{i_j})$

Blahut theorem

Theorem ([Blahut84], [Prony1795])

The DFT$_\alpha$ of a vector of weight t has linear complexity t.

Skecth of proof

- Let $v = e_i$ be a 1-weight vector. Then $\mathrm{DFT}_\alpha(v) = \mathrm{Ev}_{(\alpha^0,\alpha^1,\ldots,\alpha^n)}(X^i) = ((\alpha^0)^i, (\alpha^1)^i, \ldots, (\alpha^{n-1})^i)$ is linearly generated by $\Lambda(z) = z - \alpha^i$.
- For $v = \sum_{j=1}^{t} e_{i_j}$, the sequence $\mathrm{DFT}_\alpha(v)$ is generated by $\mathrm{ppcm}_j(z - \alpha^{i_j}) = \prod_{j=1}^{t}(z - \alpha^{i_j})$

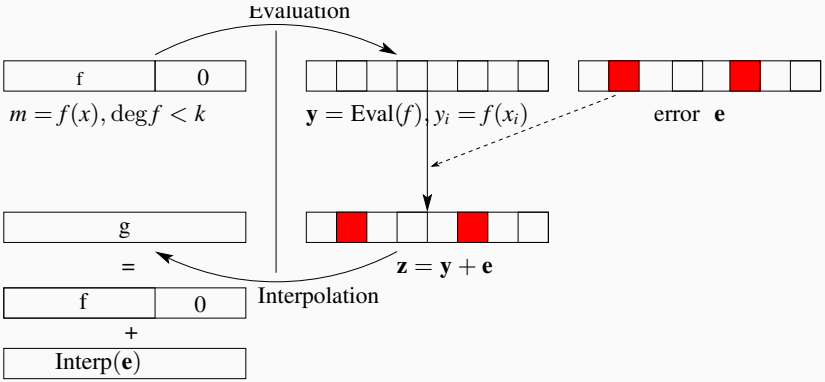
Corollary

The roots of Λ localize the non-zero elements of v: α^{i_j}.

$\rightsquigarrow$ *error locator*

Syndrom Decoding of Reed-Solomon codes

$$\mathcal{C} = \{(f(x_1), \ldots, f(x_n)) | \deg f < k\}$$



Syndrom Decoding of Reed-Solomon codes

$$\mathcal{C} = \{(f(x_1), \ldots, f(x_n)) | \deg f < k\}$$