4F5: Advanced Information Theory and Coding
Coding & Cryptology Part

Jossy Sayir
js851@cam.ac.uk

Lent Term 2022

Contents

Acknowledgments

Much of the material in this course is heavily inspired by the lecture notes of my late PhD supervisor Jim Massey, available from http://www.isiweb.ee.ethz.ch/archive/massey_scr/, and further inspired by the notes of my doctoral "older brother" Ueli Maurer for the course "Discrete Mathematics" he currently teaches at ETH Zurich.

I also received valuable feedback from students over the years, allowing me to fix a few typos and mistakes, notably Vyas Raina, Rob Sumner, Jon Carter, Demetris Skottis, Felix Kok and many others. I don't doubt that there are many mistakes left and I may have introduced some new ones while trying to improve the notes, so please email me (my email address is on the front page) about any suspected mistakes, inconsistencies, etc. that you find!

Chapter 1

Practical Number Theory and Algebra

Number theory and algebra were long considered to be the purest of all mathematical disciplines, a bastion of true “mathmos”[1] working on matters so remote and abstract that no engineer or physicist could possibly take interest in them. Things have changed over the second half of the twentieth century, when engineering applications such as coding theory and cryptography started making use of results from both these disciplines in practical contexts. This by no means indicates that number theory or algebra have stopped being pure math disciplines. Novels have been written about Andrew Wiles’ 1995 proof of Fermat’s 1637 “last theorem” (no positive integers a, b, c satisfy $a^n + b^n = c^n$ for $n > 2$). Preda Mihăilescu’s 2002 proof of Catalan’s 1844 conjecture is no less remarkable (the only consecutive powers of positive integers are 8 and 9. In other words, the equation $a^n = b^m + 1$ for $a > 0, b > 0, n > 1, m > 1$ has only one solution $a = 3, b = 2, n = 2, m = 3$). Both number theory and algebra are disciplines of great elegance, where questions may appear trivial at first glance, but turn out to pose tremendous challenges and attract the greatest and bravest mathematicians who often dedicate their lives to one simple looking problem.

Are we about to embark on an excursion into the hardest fields of mathematics as a quick aside in the 4th year course of your Engineering studies? The answer is yes and no at the same time. Yes, we will learn some number theory and algebra. But our focus will be very different from the approach taken by number theory and algebra courses in the math department. We have a purely “geeky” interest in those disciplines. Our aim is to understand the objects they describe to a sufficient level of depth so that we can use them in the context we require, but not beyond. We will need to understand some mathematical proofs in the process and I am sure that you will enjoy and experience this course as one of the most math-oriented of your student days. Remember however that we remain engineers and our interest is practically motivated. Don’t get lost in the mathematical formalism. Try to always interpret every formula, equation and theorem in terms of what it means for simple numerical examples. We will often give definitions and theorems without the level of

[1] `https://en.wiktionary.org/wiki/mathmo`

generality that is provided in math courses, because we know that the cases our definitions exclude currently have no practical relevance in the applications we will later investigate. I consider my job done if you come out of this lecture with sufficient understanding so you can program number theory and algebra "calculators" that could power the coding methods and cryptographic algorithms discussed later in the course.

1.1 Number Theory

1.1.1 Euclid's division theorem and remainders

Number theory deals with the set of integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$. One of the earliest treatises on number theory was Book 7 of Euclid's "Elements" and contained the following fundamental theorem:

Theorem 1.1 (Euclid's division theorem) *For any integers n (the "dividend") and $d \neq 0$ (the "divisor"), there exist unique integers q (the "quotient") and r (the "remainder") such that*

$$n = qd + r \tag{1.1}$$

and

$$0 \leq r < |d|. \tag{1.2}$$

Proof: We will assume without loss of generality that $n > 0$ and $d > 0$. The proof can be repeated along very similar lines for all possible sign combinations. The proof has the steps:

1. we show that there exists at least one pair (q', r') with $r' \geq 0$ such that $n = q'd + r'$ (ignoring the extra condition)
2. we show that if $r' > d$, we can generate a new pair (q'', r'') such that $n = q''d + r''$ and $0 \leq r'' < r'$
3. note that step 1 and step 2 together imply that there exists a pair (q, r) that satisfies both conditions because it means that we can keep generating new pairs with smaller remainders until the remainder satisfies the second condition
4. we show that the pair (q, r) is unique

Step 1 is easily proved by giving an example of a pair that satisfies the condition: $(q', r') = (0, n)$ fits the bill.

Step 2 is a recursive step. Take $(q'', r'') = (q' + 1, r' - d)$. Clearly, $r'' \geq 0$ and $r'' = r' - d < r'$ since $d > 0$. Furthermore, $q''d + r'' = (q' + 1)d + r' - d = q'd + r' = n$ so the pair (q'', r'') satisfies the first condition.

We have now proved that there exists a pair (q, r) that satisfies both conditions.

Now suppose the two pairs (q_1, r_1) and (q_2, r_2) satisfy both conditions. Let us assume without loss of generality that $q_1 \geq q_2$. Then we can rearrange $n = q_1 d + r_1 = q_2 d + r_2$ to yield

$$(q_1 - q_2)d = r_2 - r_1 \tag{1.3}$$

but the expression on the left is either zero or larger than d. The expression on the right cannot be larger than d if $r_1 < d$ and $r_2 < d$. Hence, $(q_1, r_1) = (q_2, r_2)$. □

Example: 25 divided by 7 yields a quotient of 3 and a remainder of 4, since $3 \times 7 + 4 = 21 + 4 = 25$. On the other hand, -25 divided by 7 yields a quotient of -4 and a remainder of 3, since $(-4) \times 7 + 3 = -28 + 3 = -25$.

We will denote the remainder r when n is divided by d as $\mathrm{R}_d(n)$. Remainders have the following useful properties:

Theorem 1.2 (Properties of remainders) *For any integers k, n, d,*

1. $\mathrm{R}_d(n + kd) = \mathrm{R}_d(n)$,
2. $\mathrm{R}_d(k + n) = \mathrm{R}_d(\mathrm{R}_d(k) + \mathrm{R}_d(n))$,
3. $\mathrm{R}_d(kn) = \mathrm{R}_d(\mathrm{R}_d(k)\,\mathrm{R}_d(n))$.

Proof: To prove the first property, suppose (q, r) is the quotient remainder pair when n is divided by d. Then $n + kd = qd + r + kd = (q + k)d + r$ and hence $(q + k, r)$ is the quotient remainder pair for $n + kd$, which shows that the remainder is unchanged.

For the second property, suppose (q_1, r_1) and (q_2, r_2) are the quotient remainder pairs when k and n, respectively, are divided by d. Then $k + n = q_1 d + r_1 + q_2 d + r_2 = (q_1 + q_2)d + r_1 + r_2$ and hence, setting $j = q_1 + q_2$, we can use the first property to show that

$$\mathrm{R}_d(k + n) = \mathrm{R}_d(r_1 + r_2 + jd) = \mathrm{R}_d(r_1 + r_2) = \mathrm{R}_d(\mathrm{R}_d(k) + \mathrm{R}_d(n)). \tag{1.4}$$

Finally, for the third property, assume again the same notation of the quotient remainder pairs when k and n are divided by d. Then

$$kn = (q_1 d + r_1)(q_2 d + r_2) = (q_1 q_2 d + q_1 r_2 + q_2 r_1)d + r_1 r_2 \tag{1.5}$$

and we have again, setting $j = q_1 q_2 + q_1 r_2 + q_2 r_1$,

$$\mathrm{R}_d(kn) = \mathrm{R}_d(r_1 r_2 + jd) = \mathrm{R}_d(r_1 r_2) = \mathrm{R}_d(\mathrm{R}_d(k)\,\mathrm{R}_d(n)). \tag{1.6}$$

□

Examples: there are endless arithmetic games that can be played using these properties. For example, what is the remainder when 10^{26} is divided by 11? We use the third property recursively to yield

$$\mathrm{R}_{11}(10^{26}) = \mathrm{R}_{11}(\mathrm{R}_{11}(10^2)^{13}) = \mathrm{R}_{11}(1^{13}) = 1 \tag{1.7}$$

1.1.2 Greatest common divisors

For any integers n, d, we say that d divides n if $\mathrm{R}_d(n) = 0$. For integers n_1, n_2 not both zero, their *greatest common divisor* is the largest integer d that divides both n_1 and n_2 and is denoted $\gcd(n_1, n_2)$.

Examples and a few very simple properties:

- $\gcd(32, 48) = 16$ because no larger integer than 16 divides both 32 and 48.
- $\gcd(-32, 48) = 16$ too.
- $\gcd(0, n) = |n|$ for any $n \neq 0$.
- For any non-zero integer n, $\gcd(\pm n, \pm n) = |n|$. This is clearly true even if n is prime.
- For any n_1, n_2, $\gcd(\pm n_1, \pm n_2) = \gcd(n_1, n_2)$, i.e., signs are irrelevant for greatest common divisors.
- If n_1 and n_2 are not both zero, $\gcd(n_1, n_2) > 0$.
- For any n, $\gcd(n, 1) = 1$
- If p is a prime number and n is any integer not divisible by p (we will learn a lot more about prime numbers in the next section), then $\gcd(p, n) = 1$ since a prime number is only divisible by 1 and by itself.
- Any two numbers n_1, n_2 for which $\gcd(n_1, n_2) = 1$ are called *co-prime*, and two prime numbers are always co-prime but not vice-versa. For example, 8 and 9 are co-prime because $\gcd(8, 9) = 1$ but neither of them are prime.

Where we offered no justification, the proofs of these properties are very simple and left as an exercise.

Note that some references define greatest common divisors slightly differently so that the definition extends gracefully to other mathematical objects than integers. Some also use the convention that $\gcd(0,0) = 0$. We will ourselves be interested in greatest common divisors for other mathematical objects (polynomials) in the second half of these notes, but our requirements will be simple enough to avoid complicating the definition too much.

The following property will be essential in our calculations involving greatest common divisors:

Theorem 1.3 (Fundamental property of gcds**)** *For any integers n_1, n_2, k,*

$$\gcd(n_1 + kn_2, n_2) = \gcd(n_1, n_2). \tag{1.8}$$

Proof: The proof is left as an exercise and follows from the fact that any number that divides a and b also divides $a + kb$, and vice versa, any number that divides $a + kb$ and b also divides a. □

The theorem above gives us a basic tool for computing gcds fairly efficiently in a recursive manner. If we pick k to be the negative of the quotient of n_1 divided by n_2, then $n_1 + kn_2 = \mathrm{R}_{n_2}(n_1)$ and hence, the theorem implies that

$$\gcd(n_1, n_2) = \gcd(\mathrm{R}_{n_2}(n_1), n_2) \tag{1.9}$$

and this suggests the following algorithm for computing the $\gcd(n_1, n_2)$. This algorithm is described in Euclid's "Elements" and is hence known as Euclid's algorithm:

1. label the two numbers n_1 and n_2 such that $n_1 \geq n_2$
2. compute $r = \mathrm{R}_{n_2}(n_1)$ and assign $n_1 := n_2$, $n_2 := r$
3. if $n_2 = 0$, terminate and output n_1
4. return to step 2

Example: we follow the steps of Euclid's algorithm in a numerical example, for $n_1 = 748$ and $n_2 = 528$. The table below shows the state of the variables n_1, n_2, r at every iteration:

n_1	n_2	r
748	528	220
528	220	88
220	88	44
88	44	0
44	0	

The algorithm exits with the result $\gcd(748, 528) = n_1 = 44$.

For me, there are two very surprising facts about Euclid's algorithm: the first is that a textbook that predates Turing by over two millenia contains what is essentially the outline of a software algorithm. The second is that, while everyone appeared to take for granted for centuries that Euclid's algorithm was surely the most efficient way to compute the gcd, Yossi Stein published in 1967 an alternative algorithm that is more efficient than Euclid's. Stein's algorithm is based on the following properties:

Theorem 1.4 (Even/odd properties of gcds**)** *For any integers n_1 and n_2,*

1. *if n_1, n_2 are both even,* $\gcd(n_1, n_2) = 2\gcd(n_1/2, n_2/2)$
2. *if n_1 is even and n_2 is odd,* $\gcd(n_1, n_2) = \gcd(n_1/2, n_2)$
3. *if n_1, n_2 are both odd,* $\gcd(n_1, n_2) = \gcd\left(\frac{n_1-n_2}{2}, n_2\right)$

Proof: The first two properties are trivial. The last property follows from the fundamental property in Theorem 1.3 implying that $\gcd(n_1, n_2) = \gcd(n_1 - n_2, n_2)$, but since $n_1 - n_2$ is even when n_1 and n_2 are odd, we can immediately apply the second property. □

Applying these three properties recursively until $a-b$ hits zero is known as *Stein's algorithm* or the "binary GCD algorithm".

Example: we follow the steps of Stein's algoritm, applying rules 1, 2, 3 from Theorem 1.4 in turn to compute $\gcd(748, 528)$, where the variable c counts the number of time rule 1 was applied or the number of times the final result will have to be multiplied by 2

n_1	n_2	c	Rule
748	528	0	1
374	264	1	1
187	132	2	2
187	66	2	2
187	33	2	3
77	33	2	3
22	33	2	2
11	33	2	3
11	11	2	3
0	11	2	

and the algorithm returns $\gcd(748, 528) = 2^c \max(n_1, n_2) = 2^2 \times 11 = 44$.

The analysis of the computational speed of Stein's vs. Euclid's algorithm is beyond the scope of this lecture and shows that Stein's algorithm requires about 30% less subtractions than Euclid requires divisions in the worst case. Just in case you were wondering if anyone really cares about computing greatest common divisors efficiently, it may be worth mentioning that in cryptographic and some coding applications, we will be applying these methods to numbers with several hundred, sometimes thousands of digits, so the answer is: yes, algorithmic complexity matters *a lot!*

In many applications, we will be interested in expressing integers as integer combinations of other integers. For example, for the integers n_1 and n_2, we would consider the set of all integers that can be computed as $x = an_1 + bn_2$ for any integers a and b. The following theorem states that the greatest common divisor of two integers is always in that set:

Theorem 1.5 (Greatest Common Divisor Theorem) *For any integers n_1 and n_2 not both zero, there exist (not unique) integers a and b such that*

$$\gcd(n_1, n_2) = an_1 + bn_2 \tag{1.10}$$

There is an elegant number theoretic proof of this theorem that we give as an exercise in Problem 1.3.6. For the purpose of this lecture, the theorem is proved by construction: both Euclid's and Stein's algorithms can be modified to provide integers a and b along the lines of Theorem 1.5 as well as computing the gcd. Note that in some applications, we are more interested in the integers a and b than we are in the gcd. We will see in the Algebra section that, when computing the inverse of an element in a field, we operate the algorithms knowing full well that the gcd we search for is 1, but the corresponding a and b allow us to compute the inverse. Figures 1.1 and 1.2 give flowcharts[2] of the extended Euclid and extended Stein algorithms that add the capability to compute a pair (a, b) such that $\gcd(n_1, n_2) = an_1 + bn_2$ to the algorithm in addition to computing the gcd.

The idea behind the extended Euclid's algorithm is fairly simple. The algorithm maintains two pairs (a_1, b_1) and (a_2, b_2) such that, at every stage of the algorithm,

$$\begin{cases} a_1^k n_1^0 + b_1^k n_2^0 = n_1^k \\ a_2^k n_1^0 + b_2^k n_2^0 = n_2^k, \end{cases} \tag{1.11}$$

where n_1^0 and n_2^0 are the "original" two integers whose gcd is being computed, while n_1^k and n_2^k are the variables of the algorithm at step k. The two pairs are initialised as $(a_1^0, b_1^0) = (1, 0)$ and $(a_2^0, b_2^0) = (0, 1)$ which clearly satisfies (1.11). Since Euclid's algorithm computes $n_2^{k+1} = \mathrm{R}_{n_2^k}(n_1^k) = n_1^k - qn_2^k$ for some q, the following recursive relations

$$\begin{cases} a_1^{k+1} = a_2^k \text{ and } a_2^{k+1} = a_1^k - qa_2^k \\ b_1^{k+1} = b_2^k \text{ and } b_2^{k+1} = b_1^k - qb_2^k \end{cases} \tag{1.12}$$

[2] These figures are reproduced from Jim Massey's lecture notes with many posthumous thanks, although in this particular case the thanks have gone both ways because, altbough Jim was certainly the brains behind the flowcharts, I was one of the teaching assistants back in the mid 1990s who painstakingly transcribed these pictures into LaTeX...

ensure that (1.11) remains satisfied throughout the algorithm. The algorithm ends at the last stage $k^\star$ with

$$\begin{cases} a_1^{k^\star} n_1^0 + b_1^{k^\star} n_2^0 = n_1^{k^\star} = \gcd(n_1^0, n_2^0) \\ a_2^{k^\star} n_1^0 + b_2^{k^\star} n_2^0 = n_2^{k^\star} = 0 \end{cases} \tag{1.13}$$

so we end by setting $(a, b) = (a_1, b_1)$.

Example: We repeat our calculation of $\gcd(528, 748)$:

n_1	n_2	q	r	a_1	b_1	a_2	b_2
748	528	1	220	1	0	0	1
528	220	2	88	0	1	1	-1
220	88	2	44	1	-1	-2	3
88	44	2	0	-2	3	5	-7
44	0			5	-7	-12	17

and we verify that

$$\begin{cases} 5 \times 748 - 7 \times 528 = 44 = \gcd(748, 528) \\ -12 \times 748 + 17 \times 528 = 0. \end{cases} \tag{1.14}$$

We won't describe the extended Stein algorithm in detail but it follows a similar principle of recursive updates of pairs (a_1, b_1) and (a_2, b_2). The extended Stein algorithm is the method of choice for computing the gcd and associated coefficients (a, b) for cryptographic applications. For coding applications, as we will see, we are generally more interested in computing the $\gcd(p_1(X), p_2(X))$ of two polynomials $p_1(X)$ and $p_2(X)$, which is defined as a polynomial $p(X)$ of the highest possible degree that divides $p_1(X)$ and $p_2(X)$. The extended Euclid algorithm is the method of choice here because it applies directly to the polynomial case without any modifications, yielding $p(X) = \gcd(p_1(X), p_2(X))$ and two polynomials $a(X)$ and $b(X)$ such that

$$p(X) = a(X)p_1(X) + b(X)p_2(X). \tag{1.15}$$

1.1.3 Primes and co-primes

We have already encountered prime numbers when discussing gcds. Everyone knows[3] that a number $p > 1$ is prime if it is only divisible by 1 and by itself. We saw that if p is prime

[3] ...except perhaps Bill Gates who, in his 1995 book "The Road Ahead", famously claimed that it would be an obvious mathematical breakthrough if one were to develop an easy way to factor large primes. Obvious indeed...

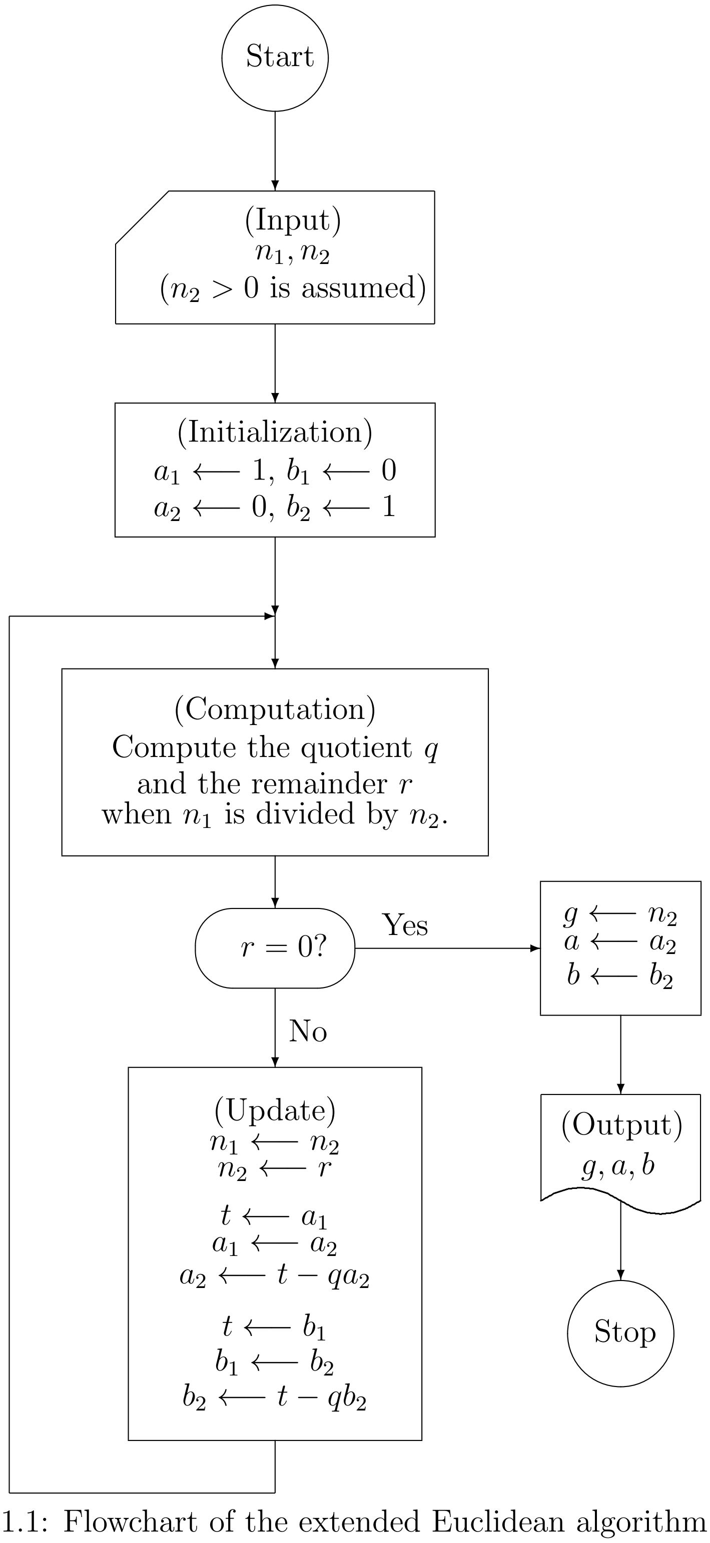

Figure 1.1: Flowchart of the extended Euclidean algorithm for computing $g = \gcd(n_1, n_2)$ together with a pair of integers a and b such that $g = an_1 + bn_2$ for integers n_1 and n_2 with $n_2 > 0$.

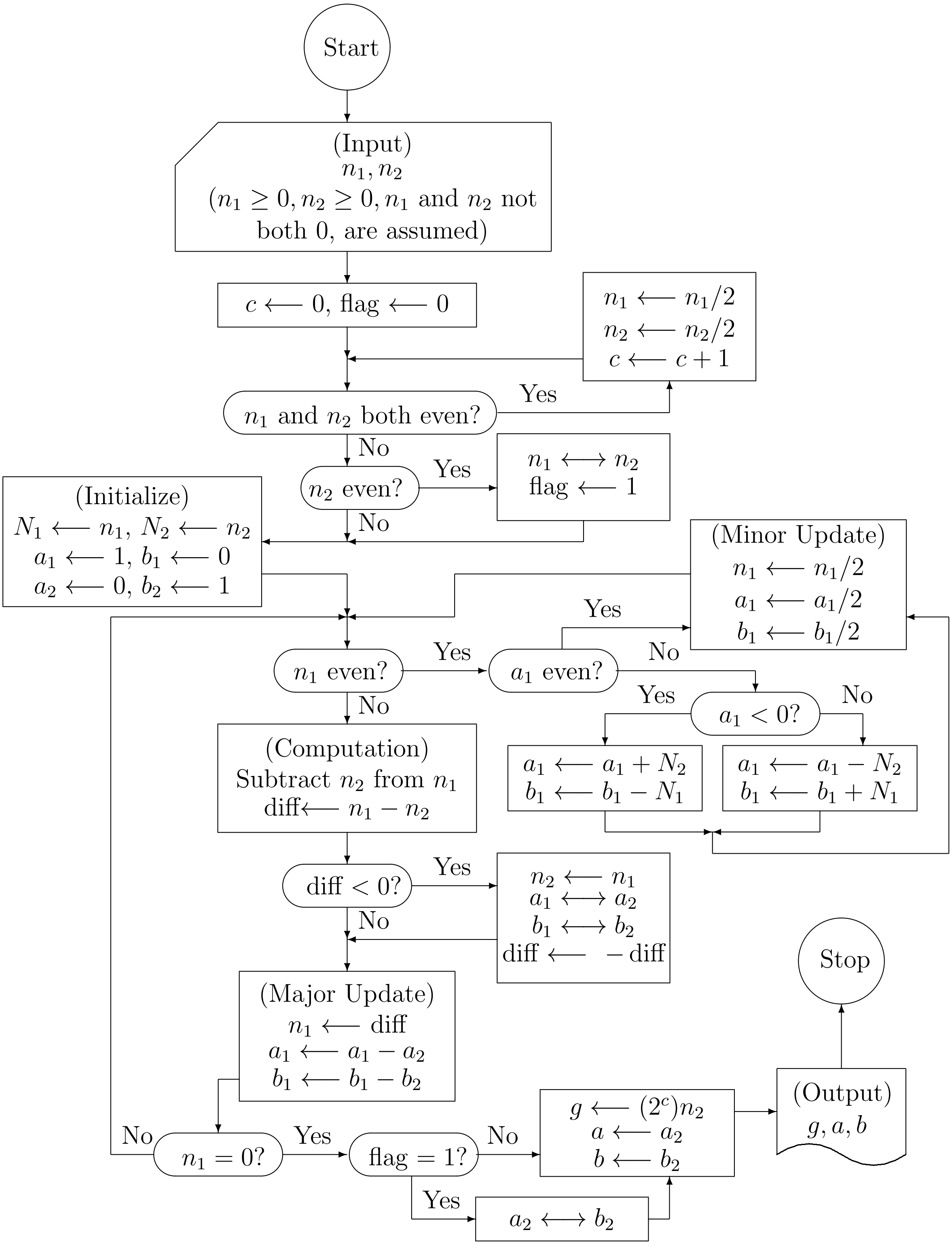
Start
(Input)
n_1, n_2
($n_1 \geq 0, n_2 \geq 0, n_1$ and n_2 not both 0, are assumed)
$c \longleftarrow 0$, flag $\longleftarrow 0$
$n_1 \longleftarrow n_1/2$
$n_2 \longleftarrow n_2/2$
$c \longleftarrow c+1$
Yes
n_1 and n_2 both even?
No
n_2 even?
Yes
$n_1 \longleftrightarrow n_2$
flag $\longleftarrow 1$
No
(Initialize)
$N_1 \longleftarrow n_1$, $N_2 \longleftarrow n_2$
$a_1 \longleftarrow 1$, $b_1 \longleftarrow 0$
$a_2 \longleftarrow 0$, $b_2 \longleftarrow 1$
(Minor Update)
$n_1 \longleftarrow n_1/2$
$a_1 \longleftarrow a_1/2$
$b_1 \longleftarrow b_1/2$
Yes
n_1 even?
Yes
a_1 even?
No
No
Yes
$a_1 < 0$?
No
$a_1 \longleftarrow a_1 + N_2$
$b_1 \longleftarrow b_1 - N_1$
$a_1 \longleftarrow a_1 - N_2$
$b_1 \longleftarrow b_1 + N_1$
(Computation)
Subtract n_2 from n_1
diff$\longleftarrow n_1 - n_2$
diff < 0?
Yes
$n_2 \longleftarrow n_1$
$a_1 \longleftrightarrow a_2$
$b_1 \longleftrightarrow b_2$
diff $\longleftarrow$ $-$diff
No
(Major Update)
$n_1 \longleftarrow$ diff
$a_1 \longleftarrow a_1 - a_2$
$b_1 \longleftarrow b_1 - b_2$
Stop
(Output)
g, a, b
$g \longleftarrow (2^c)n_2$
$a \longleftarrow a_2$
$b \longleftarrow b_2$
No
$n_1 = 0$?
Yes
flag $= 1$?
No
Yes
$a_2 \longleftrightarrow b_2$

Figure 1.2: Flowchart of the extended Stein algorithm for computing $g = \gcd(n_1, n_2)$ together with integers a and b such that $g = an_1 + bn_2$.

and n is any integer such that $R_p(n) \neq 0$, then $\gcd(p, n) = 1$, and that any two numbers n_1 and n_2 are called co-prime if $\gcd(n_1, n_2) = 1$.

The following theorem motivates much of our interest in prime numbers:

Theorem 1.6 (Fundamental theorem of arithmetic) *Every integer $n > 1$ is either prime or can be uniquely expressed as a product of prime numbers.*

Note that we ignore the order of factors when we claim "uniquely expressed" because multiplication is commutative and any re-ordering of factors will obviously give the same product. This theorem, like most of our preceding theorems, was presented in Book 7 of Euclid's "Elements".

Proof: consider any positive non-prime integer n. Since it isn't prime, it can be expressed as a product of two integers $n = ab$, where $0 \leq a < n$ and $0 \leq b < n$. Since this operation can be repeated recursively at will for any non-prime factors of n and there isn't an endless supply of positive numbers smaller than n, it must hold that n will eventually be expressed as a product of prime numbers.

Now let us assume that n has two distinct prime factorisations. To be distinct, there must either be a factor p of n present in one factorisation that isn't present in the other, or the factor p appears in both factorisations an unequal number of times. If p appears only in one factorisation, since p divides n, considering the other factorisation, it must at least divide one of its factors or be expressible as a product of some of its factors. The latter is not possible since p is prime. The former is only possible if p appears in the other factorisation. If p appears k_1 times in one factorisation and k_2 times in the other, the same argument can be made of $p^{\max(k_1,k_2)}$, thereby proving that the factorisation of n into primes is unique.

□

We now give two theorems that give us an idea about the behaviour of prime numbers:

Theorem 1.7 *There are infinitely many primes.*

We won't give a proof of this theorem but the reader is encouraged to consult the book "Proofs from the book" by Aigner and Ziegler, where 6 very pretty and simple proofs of this theorem are given. This book is dedicated to the mathematician Paul Erdős who often spoke of "The Book" in which God keeps the most elegant proof of each mathematical theorem. The following theorem, which we also offer without proof, this time because no simple or elegant proof is yet known, gives an approximation of the "density" of primes:

Theorem 1.8 (Prime number theorem) *For any integer n, let $\pi(n)$ be the number of prime numbers up to and including n. We have*

$$\lim_{n \to \infty} \frac{\pi(n)}{n/\ln(n)} = 1 \qquad (1.16)$$

In other words, the number of primes grows roughly as $n/\ln(n)$.

We finish our short introduction into number theory with one of the most powerful theorems in the field that has countless applications. Remember that two numbers n_1 and n_2 are said to be *co-prime* if $\gcd(n_1, n_2) = 1$.

Theorem 1.9 (Chinese remainder theorem) *Let $m_1, m_2, \ldots, m_k$ be pairwise co-prime integers (also called* moduli *in this context) and let $m = m_1 m_2 \cdots m_k$. Then, for any choice of integers $r_1, r_2, \ldots, r_k$ (also called* residues *in this context) such that $0 \leq r_j < m_j$ for $j = 1, 2, \ldots, k$, there is a unique non-negative integer $n < m$ for which*

$$\mathrm{R}_{m_j}(n) = r_j, \textit{ for } j = 1, 2, \ldots, k. \tag{1.17}$$

Proof: we first prove uniqueness. Assume that there are two integers n and n' that satisfy the conditions. Then for any j,

$$\mathrm{R}_{m_j}(n - n') = \mathrm{R}_{m_j}\left(\mathrm{R}_{m_j}(n) - \mathrm{R}_{m_j}(n')\right) = \mathrm{R}_{m_j}(r_j - r_j) = 0. \tag{1.18}$$

Since $n - n'$ is divisible by all the moduli and the moduli are relatively prime, $n - n'$ must be divisible by the product m of the moduli, but since $n < m$ and $n' < m$ this implies that $n - n' = 0$.

Existence follows from the fact that there are m choices for n such that $0 \leq n < m$ and there are $m = \prod_{j=1}^{k} m_j$ choices for the residue vector $(r_1, r_2, \ldots, r_k)$. Since every residue vector can correspond to at most one number n and every integer $n \in \{0, 1, \ldots, m-1\}$ obviously has residues, it must hold that there is an integer $n \in \{0, 1, \ldots, m-1\}$ for every choice of residues. □

The final argument also shows that the chinese remainder theorem can be interpreted inversely to say that any number $n \in \{0, 1, \ldots, m-1\}$ is uniquely represented by a vector of residues $(r_1, r_2, \ldots, r_k)$.

Example: For any n between 0 and $m - 1$, it is easy to find the corresponding residues. For example, let the set of moduli be $(3, 7, 10, 11)$. It is easy to verify that they are relatively prime, since 3, 7 and 11 are prime and $10 = 2 \times 5$ doesn't contain any of the other primes among its prime factors. Hence we can express any number between 0 and $m - 1 = 3 \times 7 \times 10 \times 11 - 1 = 2309$ as a vector of four residues. For example, the vector of residues for $n = 1425$ is $(0, 4, 5, 6)$.

What about going the other way around? If you are given a set of residues, say $(1, 6, 5, 3)$, can you find a number n between 0 and 2309 that has these residues? This operation requires some further thought.

Let $u_j = m/m_j$ for all j, i.e., the product of all moduli but the corresponding one. Since the moduli are co-prime, it follows that $\gcd(m_j, u_j) = 1$ and hence u_j and m_j are also co-prime. Therefore, following the gcd theorem 1.5, there exist integers a_j and b_j such that

$$a_j m_j + b_j u_j = \gcd(m_j, u_j) = 1 \tag{1.19}$$

and these can be found with the extended Euclid or Stein algorithms. We will see later in the course that the integer b_j we compute in this way is the multiplicative inverse of $\mathrm{R}_{m_j}(u_j)$ in arithmetic modulo m_j. Note that

$$\mathrm{R}_{m_j}(a_j m_j + b_j u_j) = \mathrm{R}_{m_j}(b_j u_j) = \mathrm{R}_{m_j}(1) = 1, \tag{1.20}$$

and, for $t \neq j$,

$$\mathrm{R}_{m_t}(b_j u_j) = 0 \tag{1.21}$$

since u_j contains m_t among its factors. Note as well that if we multiply $b_t u_t$ by any number $\alpha \in \{0, 1, \ldots, m_j - 1\}$, we have

$$\mathrm{R}_{m_t}(\alpha b_t u_t) = \mathrm{R}_{m_t}\left(\mathrm{R}_{m_t}(\alpha)\,\mathrm{R}_{m_t}(b_t u_t)\right) = \mathrm{R}_{m_t}(\alpha \times 1) = \alpha. \tag{1.22}$$

Now consider the integer

$$N = \sum_{j=1}^{k} r_j b_j u_j. \tag{1.23}$$

Using the relations shown above, it is easy to verify that $\mathrm{R}_{m_t}(N) = r_t$, because the remainder of every term for $j \neq t$ will be zero due to (1.21) and the remainder of the t-th term will be r_t due to (1.22). We almost have a solution, except that N is likely to be larger than m and hence doesn't fit the bill. However, we can finally compute

$$n = \mathrm{R}_m(N) \tag{1.24}$$

and note that, since $n = N + cm$ for some c and m is divisible by all m_j, we have

$$\mathrm{R}_{m_t}(n) = \mathrm{R}_{m_t}(N + cm) = \mathrm{R}_{m_t}\left(\mathrm{R}_{m_t}(N) + \mathrm{R}_{m_t}(cm)\right) = \mathrm{R}_{m_t}(N) = r_t. \tag{1.25}$$

Putting it all together, we obtain the following formula

$$n = \mathrm{R}_m\left(\sum_{j=1}^{k} r_j b_j u_j\right). \tag{1.26}$$

Example: going back to the previous example, with the moduli $(3, 7, 10, 11)$, we had $m = 2310$ and we can compute the $u_j = m/m_j$ to yield the numbers $(770, 330, 231, 210)$, and the corresponding b_j are $(2, 1, 1, 1)$, so the values of $b_j u_j$ are $(1540, 330, 231, 210)$. Hence, we can compute any n from its residues as

$$n = \mathrm{R}_{2310}\left(1540 r_1 + 330 r_2 + 231 r_3 + 210 r_4\right). \tag{1.27}$$

For the residues $(1, 6, 5, 3)$ given before, we obtain $n = 685$. It is easy to verify indeed that $\mathrm{R}_3(685) = 1$, $\mathrm{R}_7(685) = 6$, $\mathrm{R}_{10}(685) = 5$ and $\mathrm{R}_{11}(685) = 3$.

1.2 Algebra

Algebra considers sets, and operations defined on those sets. It investigates what properties emerge from these arrangements of sets and operations. Algebra is normally presented using an *axiomatic approach* that is likely to be unfamiliar to most engineering students.

The idea is that, rather than looking at specific examples of sets and operations, generic operations on generic sets are considered, and a set of minimal properties ("axioms") are postulated. Further properties that follow from the axioms are then derived. Engineers are mostly interested in numbers and hence, much of the algebra you have learned at school and over the course of your engineering degree was specialised to the sets of real numbers $\mathbb{R}$, the set of complex numbers $\mathbb{C}$, etc. The operations in this context are addition and multiplication and other arithmetic operations derived from them (e.g. exponentiation, vector and matrix operations, etc.) In coding and cryptology, we will extend our interest to operations on finite sets of numbers, e.g., operations modulo 2, 3, etc., and to polynomials over these sets. Operations on such sets have a lot of similarities with familiar operations on $\mathbb{R}, \mathbb{C}$ and some interesting differences. Mathematicians also consider other sets such as sets of sets and operations that aren't necessarily reducible to a type of addition and multiplication, but in most cases such examples will lie outside our sphere of interest.

1.2.1 Algebraic systems

Table 1.1: Algebraic systems with a single operation

Conditions	**Statement**	$\langle \mathbf{S}, \star \rangle$
(i) Closure	for any $a, b \in S$, $c = a \star b \in S$	
(ii) Associative law	for any $a, b, c \in S$, $(a \star b) \star c = a \star (b \star c)$	Semi-group
(iii) Neutral element	there exists $e \in S$ such that for any $a \in S$, $e \star a = a \star e = a$. e is the neutral element of S	Monoid
(iv) Inverse	for any $a \in S$, there exists $b \in S$ such that $a \star b = b \star a = e$	Group
(v) Commutativity	for any $a, b \in S$, $a \star b = b \star a$	Abelian group

We will begin by presenting the elementary steps of the axiomatic approach. A single operation algebraic system consists of a set S and an operation $\star$ and is denoted $\langle S, \star \rangle$. The operation is applied to two elements of the set to yield a result. There are various classes of algebraic systems depending on the axioms/properties that the operation fulfills. Table 1.1 summarises the axioms of single operation algebraic systems that we will study. The easiest way to understand this table is to give examples of algebraic systems at each level of the table:

Example: Consider the following algebraic systems:

- $\langle S, \star \rangle = \langle \mathbb{R}^n, \cdot \rangle$ the set of n-dimensional vectors over the reals with the "dot" or "scalar" product doesn't satisfy the axiom of closure: it operates on vectors and the outcome is not a vector but a number.

- $\langle S, \star \rangle = \langle \mathbb{R}^3, \times \rangle$ the set of 3-dimensional vectors over the reals with the vector product satisfies closure since the outcome is also a 3-dimensional vector, but doesn't satisfy associativity as can be verified by considering the unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ and observing that $\mathbf{i} \times (\mathbf{i} \times \mathbf{j}) = \mathbf{i} \times \mathbf{k} = -\mathbf{j}$ wheras $(\mathbf{i} \times \mathbf{i}) \times \mathbf{j} = \mathbf{0} \times \mathbf{j} = \mathbf{0}$.

- $\langle S, \star \rangle = \langle \mathbb{N}^\star, + \rangle$ the set of positive (non-zero) integers with addition is a *semigroup.* It satisfies closure (sum of positive numbers is positive) and associativity, but there is no neutral element whose addition leaves an element unchanged, since we exlcuded zero from the set.

We interrupt this exposition to introduce a crucial notation that will be the main focus of our interest throughout the rest of this course. The set $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ is the set of non-negative natural numbers ("integers") strictly smaller than m. In coding and cryptography, we are mostly interested in operations modulo m on $\mathbb{Z}_m$. We will denote addition on $\mathbb{Z}_m$ as $\oplus$ and multiplication as $\odot$ to remind ourselves that these are modulo operations and not simply integer addition and integer multiplication (which don't satisfy closure.) Hence, using the remainder notation introduced in the number theory section, we define, for $a, b \in \mathbb{Z}_m$,

$$a \oplus b \overset{\text{def}}{=} \mathrm{R}_m(a+b) \tag{1.28}$$

and

$$a \odot b \overset{\text{def}}{=} \mathrm{R}_m(a \cdot b). \tag{1.29}$$

$\mathbb{Z}_m^\star = \{1, 2, \ldots, m-1\}$ denotes the set of positive natural numbers smaller than m, i.e., $\mathbb{Z}_m^\star = \mathbb{Z}_m \setminus \{0\}$ or, in plain English, "$\mathbb{Z}_m$ excluding zero".

- $\langle S, \star \rangle = \langle \mathbb{Z}_m, \odot \rangle$ multiplication modulo m is a monoid. It satisfies closure and associativity and the neutral element is 1 since $a \odot 1 = 1 \odot a = a$, but zero has no inverse since there is no element $a \in \mathbb{Z}_m$ such that $a \odot 0 = 1$. In $\langle \mathbb{Z}_m, \odot \rangle$ where m is a *composite* (non-prime), we will observe in the next section that any $a \in \mathbb{Z}_m$ that is not co-prime with m does not have an inverse. The monoid $\langle \mathbb{Z}_m, \odot \rangle$ will be of particular interest to us in the study of public key cryptography.

- $\langle S, \star \rangle = \langle \mathbb{Z}_m, \oplus \rangle$ addition modulo m is an Abelian group. It satisfies closure, associativity, zero is the neutral element, and every element $a \in \mathbb{Z}_m$ has the inverse $b = m - a$ since $a \oplus b = b \oplus a = \mathrm{R}_m(a+b) = \mathrm{R}_m(a+m-a) = \mathrm{R}_m(m) = 0$.

- $\langle S, \star \rangle = \langle \mathbb{Z}_4^\star, \odot \rangle$ multiplication modulo 4 excluding zero doesn't satisfy closure since $2 \odot 2 = 0$.

- $\langle S, \star \rangle = \langle \mathbb{Z}_p^\star, \odot \rangle$ multiplication modulo a prime number p is an Abelian group because every element $a \in \mathbb{Z}_p$ is co-prime with p and hence has an inverse, and multiplication of integers is commutative.

The Abelian groups $\langle \mathbb{Z}_p, \oplus \rangle$ and $\langle \mathbb{Z}_p^\star, \odot \rangle$ will be of particular interest to us in the study of algebraic coding theory.

Table 1.2: Algebraic systems with two operations

$\langle \mathbf{S}, + \rangle$	$\langle \mathbf{S}, \cdot \rangle$	**Conditions**	$\langle \mathbf{S}, +, \cdot \rangle$
Ab. Group	Monoid with $e_\cdot \neq e_+$	Distributive law, for any $a, b, c \in S$, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(a+b) \cdot c = (a \cdot c) + (b \cdot c)$	Ring
Ab. Group	Monoid with $e_\cdot \neq e_+$	$\langle S \setminus \{e_+\}, \cdot \rangle$ is an Ab. Group	**Field**

Algebraic systems can be combined with two operations defined on the same set. Table 1.2 show the axioms that must be fulfilled for algebraic systems with two operations.

Example: The following algebraic systems with two operations will be of interest to us:

- $\langle \mathbb{Z}_m, \oplus, \odot \rangle$ addition and multiplication modulo a composite number m is a ring. $\langle \mathbb{Z}_m, \oplus \rangle$ is an Abelian group, $\langle \mathbb{Z}_m, \odot \rangle$ is a monoid, the neutral element 0 of addition is not the same as the neutral element 1 of multiplication, and the distributive laws follow from the distributive laws for normal addition and multiplication which transfer naturally to operations modulo m due to the properties 1 and 2 of remainders from Theorem 1.2.

- $\langle \mathbb{Z}_p, \oplus, \odot \rangle$ addition and multiplication modulo a prime number p is a field because $\langle \mathbb{Z}_p \setminus \{0\}, \odot \rangle = \langle \mathbb{Z}_p^\star, \odot \rangle$ is an Abelian group.

A *finite* ring or field is a ring or field $\langle S, +, \cdot \rangle$ where the set S has a finite number of elements. There is one more type of finite field called an *extension* field that we will investigate later in this course that has p^e elements, i.e., powers of prime numbers. However, be warned that $\langle \mathbb{Z}_{p^e}, \oplus, \odot \rangle$, i.e., addition and multiplication modulo p^e, is *not* a field but a ring. We will need to define a very different operation on sets of p^e elements to obtain a field.

We will now consider the properties of monoids and groups.

1.2.2 Properties of monoids

The essence of the axiomatic approach is that further properties can now be deduced from the postulated axioms. Mathematicians attach much importance to the fact that the axioms are the minimal rules that need to be postulated and no axioms can be deduced from each other. We will not be so preoccupied with this aspect of the axiomatic approach but we show here two simple properties of monoids that follow from the axioms to give you a flavour of how the axiomatic construction operates.

Theorem 1.10 (Uniqueness of the neutral element) *The neutral element e in a monoid is unique.*

Proof: Assume that a monoid $\langle S, \star\rangle$ has two neutral elements e_1 and e_2. Then we can write

$$e_1 = e_1 \star e_2 = e_2 \tag{1.30}$$

where we have used the property of the neutral element e_2 in the first equality and the property of the neutral element e_1 in the second equality. Hence the two elements are identical. □

Theorem 1.11 (Uniqueness of the inverse) *For a monoid $\langle S, \star\rangle$, if an element $a \in S$ has an inverse $a^{-1} = b$, then this inverse is unique.*

Proof: Again we prove by contradiction by assuming that a has two inverses b and c. Then we can write

$$\begin{aligned} b &= e \star b && (1.31)\\ &= (c \star a) \star b && (1.32)\\ &= c \star (a \star b) && (1.33)\\ &= c \star e && (1.34)\\ &= c && (1.35)\end{aligned}$$

where we have used the property of the neutral elements in the first and the fifth equality, the property of the inverse in the second and the fourth equality, and associativity in the third equality. □

We will now depart from the abstract axiomatic description of monoids and focus on the monoid that interests us most, $\langle \mathbb{Z}_m, \odot\rangle$, multiplication modulo m. For this monoid, the following property holds:

Theorem 1.12 (Invertible elements of $\langle \mathbb{Z}_m, \cdot\rangle$) *An element u of $\mathbb{Z}_m$ is invertible if and only if* $\gcd(u, m) = 1$.

Proof: First, assume that u has an inverse $u^{-1} = b$. In this case,

$$u \odot b = 1 = \mathrm{R}_m(ub) = ub - qm \tag{1.36}$$

for some q. Since both ub and qm are divisible by $\gcd(u, m)$, $ub - qm = 1$ must be divisible by $\gcd(u, m)$ and this implies that $\gcd(u, m) = 1$ since 1 is the only non-negative number that divides 1.

Now let us assume that $\gcd(u, m) = 1$. Then, by the Greatest Common Divisor Theorem 1.5, there exist numbers a and b such that

$$\gcd(u, m) = 1 = am + bu \tag{1.37}$$

and we can find such numbers using Euclid's or Stein's extended algorithms. But

$$\mathrm{R}_m(bu) = \mathrm{R}_m(am + bu) = 1 = \mathrm{R}_m\left(\mathrm{R}_m(b)u\right) = \mathrm{R}_m(b) \odot u \tag{1.38}$$

and hence $\mathrm{R}_m(b)$ is the inverse of u in $\langle \mathbb{Z}_m, \odot \rangle$. □

The proof above did not just show the existence of the inverse but also gave us a practical way to compute the inverse of an element: use Euclid's or Stein's extended gcd algorithm to compute $\gcd(u, m) = au + bm$. If the result is 1, then compute the inverse as $u^{-1} = \mathrm{R}_m(a)$.

For the monoid $\langle \mathbb{Z}_m, \odot \rangle$, an interesting question is to know how many elements are invertible. This is known as Euler's function

$$\varphi(m) \overset{\text{def}}{=} \#\{k : 0 \leq k < m, \gcd(k, m) = 1\} \tag{1.39}$$

where the notation $\#S$ denotes the cardinality, or number of elements in a set. In other words, Euler's function counts the number of elements that are co-prime with m.

Obviously, if $m = p$ is prime, then we have

$$\varphi(p) = p - 1 \tag{1.40}$$

since every element except zero is co-prime with any prime number. If $m = p^e$, it is easy to see that

$$\varphi(m) = (p - 1)p^{e-1} \tag{1.41}$$

because, since any number between 0 and $p^e - 1$ can be written as $qp + r$, where $0 \leq q < p^{e-1}$ and $0 \leq r < p$, and only those numbers for which $r = 0$ are not co-prime with p^e, we have p^{e-1} choices for the value of q and $p - 1$ choices for the value of r.

Finally, the general expression for $\varphi(m)$ for any composite $m = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ is

$$\varphi(m) = \prod_{j=1}^{k} (p_j - 1)p_j^{e_j - 1}. \tag{1.42}$$

To see this, we need the following property of the Chinese Remainder Theorem:

Theorem 1.13 (Multiplication and inversion property of the CRT) *For a set of co-prime moduli $(m_1, \ldots, m_k)$ and $m = m_1 \cdot m_2 \cdots m_k$, let a and b be two elements of $\mathbb{Z}_m$ with residuals $(a_1, a_2, \ldots, a_k)$ and $(b_1, b_2, \ldots, b_k)$, respectively. Then the residuals of the product $a \odot b = R_m(ab)$ are the products of the residuals, i.e.,*

$$\begin{aligned}(r_1(a \odot b), r_2(a \odot b), \ldots, r_k(a \odot b)) &= (a_1 \odot_{m_1} b_1, a_2 \odot_{m_2} b_2, \ldots, a_k \odot_{m_k} b_k) && (1.43)\\ &= (\mathrm{R}_{m_1}(a_1 b_1), \mathrm{R}_{m_2}(a_2 b_2), \ldots, \mathrm{R}_{m_k}(a_k b_k)) && (1.44)\end{aligned}$$

where we have used the notation $\odot_{m_k}$ for the product operation in $\langle \mathbb{Z}_{m_k}, \odot \rangle$. Furthermore, if $a \in \mathbb{Z}_m$ is invertible, the residuals of the inverse a^{-1} of a are the inverses of the residuals.

Proof: The statement about the product can be verified by noting that

$$\mathrm{R}_{m_j}(a \odot b) = \mathrm{R}_{m_j}(\mathrm{R}_m(ab)) = \mathrm{R}_{m_j}(ab - qm) \tag{1.45}$$

for some q. But m is divisible by m_j, and hence

$$\mathrm{R}_{m_j}(a \odot b) = \mathrm{R}_{m_j}(ab) = \mathrm{R}_{m_j}\big(\mathrm{R}_{m_j}(a)\,\mathrm{R}_{m_j}(b)\big) = \mathrm{R}_{m_j}(a_j b_j) = a_j \odot_{m_j} b_j. \tag{1.46}$$

The statement about the inverse follows from the observation that 1 in $\mathbb{Z}_m$ has residuals $(1, 1, \ldots, 1)$ and, by the Chinese Remainder Theorem, 1 is the only number to have these residuals. Hence, for any $a \in \mathbb{Z}_m$, if there exists an element $b \in \mathbb{Z}_m$ such that $a \odot b = 1$, then by the property of the product just proved, the residuals of b must be the inverses in $\mathbb{Z}_{m_j}$ of the residuals of a for $j = 1, 2, \ldots, k$. □

As the theorem shows, choosing invertible elements in $\mathbb{Z}_m$ is equivalent to choosing invertible residuals in the Chinese Remainder Theorem. Hence, the expression of Euler's function for a composite m in (1.42) follows from the expression of Euler's function for prime powers (1.41) by picking residuals for the moduli $(p_1^{e_1}, p_2^{e_2}, \ldots, p_k^{e_k})$.

Furthermore, the theorem did not just give us a way to count invertible elements but also provided a practical way to verify if an element of $\mathbb{Z}_m$ is invertible and to compute its inverse if a prime decomposition of m is known. This approach is the essential tool for a public key cryptographic method known as Rivest-Shamir-Adelmann (RSA) that we will learn about later in this course.

1.2.3 Properties of groups

We now move on from monoids to groups. Remember that the difference between monoids and groups is that every element in a group is invertible. As stated earlier, the algebraic system $\langle \mathbb{Z}_m, \oplus \rangle$ of addition modulo any integer m is a group, because for any integer a such that $0 \leq a < m$, $b = m - a$ is the additive inverse of a, denoted $-a$, because $a \oplus b = \mathrm{R}_m(a + b) = R_m(m) = 0$ the neutral element under addition. On the other hand, the algebraic system $\langle \mathbb{Z}_m, \odot \rangle$ is never a group for any m, because 0 never has an inverse under multiplication. If m is a prime $m = p$ and we exclude zero to form $\mathbb{Z}_p^\star = \mathbb{Z}_p \setminus \{0\}$, then $\langle \mathbb{Z}_p^\star, \odot \rangle$ is a group because every element $a \in \mathbb{Z}_p^\star$ is co-prime with p and hence, since $\gcd(a, p) = 1$ it must have an inverse, as we saw when we studied monoids in the previous section.

Furthermore, for any monoid $\langle M, \star \rangle$, the algebraic system $\langle M^\star, \star \rangle$ where $M^\star$ is the set of invertible elements in M is a group: it clearly inherits associativity from $\langle M, \star \rangle$; every element is invertible by definition; so we only need to verify closure to establish this fact. Let $a, b \in M^\star$, then

$$b^{-1} \star a^{-1} \star a \star b = b^{-1} \star (a^{-1} \star a) \star b = b^{-1} \star e \star b = (b^{-1} \star e) \star b = b^{-1} \star b = e \tag{1.47}$$

hence $a \star b$ has an inverse $b^{-1} \star a^{-1}$ and hence $a \star b \in M^\star$, verifying closure.

Groups are essential in algebra because using group operations ensures that the "laws of algebra" we learned in primary school work. Most of what we learned about re-arranging equations and using variables relies on group properties. In particular, in a group $\langle G, \star \rangle$, the equation

$$a \star x = b \tag{1.48}$$

for $a, b \in G$ always has *a unique solution* $x = a^{-1} \star b \in G$. This not the case for a monoid, e.g., $0 \odot x = 0$ has many solutions in $\langle \mathbb{Z}_m, \odot \rangle$, $0 \odot x = 1$ has no solutions, and $2 \odot x = 0$ has two solutions in $\langle \mathbb{Z}_4, \odot \rangle$.

We now consider exponentiation in groups. For an element $a \in G$ in a group, we define

$$a^n \overset{\text{def}}{=} \overbrace{(a \star a \star a \star \cdots \star a)}^{n \text{ times}}. \tag{1.49}$$

It is easy to verify that the usual rules apply for exponentiation, i.e., $a^{n_1} \star a^{n_2} = a^{n_1+n_2}$, $a^{n_1} \star a^{-n_2} = a^{n_1-n_2}$ (where a^{-n_2} is the inverse of a^{n_2}), and $(a^{n_1})^{n_2} = a^{n_1 n_2}$.

The order of an element $a \in G$ in a group, denoted $\operatorname{ord}(a)$, is the smallest positive integer n such that $a^n = e$, i.e., a operated n times gives the neutral element. In a finite group, every element $a \in G$ always has a well defined finite order: since there is a finite supply of elements in the group, if we compute a^n for $n = 1, 2, 3, \ldots$ we are bound to have repetitions of the same element somewhere along the way. Let $n_1 < n_2$ such that $a^{n_1} = a^{n_2}$, then

$$e = a^{n_2} \star a^{-n_1} = a^{n_2-n_1} \tag{1.50}$$

and hence there exists a finite exponent $n = n_2 - n_1$ such that $a^n = e$ and, since the set of such exponents is not empty, then it must have a smallest element.

Consider any element a in a group $\langle G, \star \rangle$ whose order is $n = \operatorname{ord}(a)$, and now consider the algebraic system $\langle \{a^1, a^2, \ldots, a^{n-1}, a^n\}, \star \rangle$. This system is a group because it contains the neutral element $e = a^n$ which we denote a^0 by convention, and it is closed because, for every $n_1, n_2 \in \{0, 1, \ldots, n-1\}$,

$$a^{n_1} \star a^{n_2} = a^{qn+\mathrm{R}_n(n_1+n_2)} = e^q \star a^{R_n(n_1+n_2)} = a^{R_n(n_1+n_2)}. \tag{1.51}$$

This group is called the *cyclic group* generated by a and a is called a generator of the group. The generator is not unique and it is easy to see that any a^m for which $\gcd(m, n) = 1$

generates the same cyclic group, and hence there are $\varphi(n)$ generators in a cyclic group. More generally, for any element $b = a^m$ in the cyclic group generated by a,

$$\operatorname{ord}(b) = \frac{n}{\gcd(m,n)}. \tag{1.52}$$

The number $\#G$ of elements in a cyclic group generated by a is $\operatorname{ord}(a)$ and, by extension, we will call the number of elements in any group its *order* although it is not true in general that every group has an element that generates it, because there exist finite groups that are not cyclic (but we won't encounter any in the context of this course.) Cauchy's theorem (which we will not prove here) states that there exist elements of any prime order that divides the order of a group, so if the order of a group is itself a prime number then it is necessarily cyclic.

For any group $\langle G, \star\rangle$, and let $H \subseteq G$ be a subset of G. If $\langle H, \star\rangle$ is itself a group, then we call it a sub-group of $\langle G, \star\rangle$. Essentially $\langle H, \star\rangle$ inherits associativity from its parent group, so all that needs to be verified is that $\langle H, \star\rangle$ satisfies closure under $\star$ and under inversion. As a consequence, H must contain the neutral element $e \in G$. It is easy to see that for any element $a \in G$, the cyclic group generated by a is a sub-group of G. The following is probably the most powerful and surprising theorem of algebra:

Theorem 1.14 (Lagrange's theorem) *The order of any subgroup of a finite group divides the order of the group.*

Proof: Let H be a subgroup of a group $\langle G, \star\rangle$. Now consider an element a not in H and consider the set $a \star H$ of elements of G obtained as $a \star b$ for all $b \in H$. The set $a \star H$ has two crucial properties

- it contains a, since $e \in H$ and hence it contains $a \star e = a$.
- it has the same number of elements as H, since $a \star b$ must give distinct values for all distinct b, otherwise $a \star b = c$ would have multiple solutions.

These two properties allow us to partition G into non-intersecting subsets $H, a_1 \star H, a_2 \star H, \ldots$ where we can continue to generate such subsets as long as we can find elements of G that are not already within one of the subsets generated. Since all of these subsets have the same number of elements as H, it follows that the order of H must divide the order of G, completing the proof. Note that a subset $a \star H$ as constructed is called a *left coset* of the group H, and similarly a *right coset* can be constructed as $H \star a$. □

As a consequence of Lagrange's theorem and since any element of a group generates its own cyclic group by exponentiation, the order of an element must divide the order of a group. For example, in $\langle \mathbb{Z}_7^\star, \odot\rangle$ multiplication modulo 7, there are 6 non-zero elements, and hence group elements can have orders 1, 2, 3 or 6. The neutral element 1 has order 1 by convention. It is easy to verify that 6 has order 2 because $6^2 = 1 \bmod 7$; 2 and 4 have order 3 because $2^3 = 4^3 = 1 \bmod 7$; and finally 3 and 5 have order 6 and hence generate the group.

There are two consequences of Lagrange's theorem that have earned their own names as theorems because they pre-dated Lagrange:

Theorem 1.15 (Euler's theorem) *For any invertible element $u \in \langle \mathbb{Z}_m, \odot \rangle$, $u^{\varphi(m)} = 1$.*

and

Theorem 1.16 (Fermat's theorem) *For any prime number p and any positive $a < p$, $a^{p-1} = 1$ modulo p.*

1.2.4 Finite fields and vector spaces

We now focus on algebraic systems with two operation, returning to Table 1.2 to find two such systems, *rings* and *fields.* We have already noted that $\langle \mathbb{Z}_m, \odot \rangle$ is a monoid and not a group when m is not prime, and hence $\langle \mathbb{Z}_m, \oplus, \odot \rangle$ is a ring in this case. It is an Abelian group with respect to addition with neutral element 0, a monoid with respect to multiplication with neutral element $1 \neq 0$, and the distributive laws apply to combinations of addition and multiplications. Similarly, $\langle \mathbb{Z}_p, \oplus, \odot \rangle$ is a field when p is prime, since every element except zero is invertible under multiplication and hence the multiplicative monoid becomes an Abelian group once the element 0 is removed. The finite field $\langle \mathbb{Z}_p, \oplus, \odot \rangle$ is called the *Galois field* of order p, denoted GF(p). We will learn in this section that finite fields (fields with a finite number of elements) only exist for prime numbers of elements p or for powers of a prime $q = p^k$, and all of them are known as Galois Fields GF(q). The associated multiplicative group is always cyclic[4] (e.g., has a generator) whereas the additive group is only cyclic for prime fields GF(p) but not for so-called *extension* fields GF(q) where $q = p^k, k > 1$.

Fields are the most powerful algebraic systems on which all the familiar rules of linear algebra apply. We are already familiar with addition and multiplication over the set $\mathbb{R}$ of real numbers and the set $\mathbb{C}$ of complex numbers and both of these algebraic systems are fields. The following concepts from linear algebra exist in all fields, including GF(p):

- matrix multiplication, addition, and multiplication by vectors
- matrix inversion
- determinant and rank
- LU factorisation and QR decomposition
- eigenvalues and eigenvectors, provided they exist, i.e., provided the characteristic polynomial has roots (we will say a little more about the existence of roots later)

[4] we won't prove that but the proof isn't hard and you can prove it as an exercise or look for a proof online if you're interested

The list is not exhaustive. The concepts of vector spaces and dimension also translate gracefully from reals and complex numbers to finite fields, with the difference that a vector space over a finite field has a finite number of elements. For example, the vector space $\mathrm{GF}(3)^3$ of 3-dimensional ternary vectors contains $3^3 = 27$ row vectors:

$$\begin{array}{lll} [0,0,0]\,, & [1,0,0]\,, & [2,0,0]\,, \\ [0,0,1]\,, & [1,0,1]\,, & [2,0,1]\,, \\ [0,0,2]\,, & [1,0,2]\,, & [2,0,2]\,, \\ [0,1,0]\,, & [1,1,0]\,, & [2,1,0]\,, \\ [0,1,1]\,, & [1,1,1]\,, & [2,1,1]\,, \\ [0,1,2]\,, & [1,1,2]\,, & [2,1,2]\,, \\ [0,2,0]\,, & [1,2,0]\,, & [2,2,0]\,, \\ [0,2,1]\,, & [1,2,1]\,, & [2,2,1]\,, \\ [0,2,2]\,, & [1,2,2]\,, & [2,2,2]\,. \end{array} \tag{1.53}$$

Example: consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \tag{1.54}$$

over GF(3). It has determinant

$$\det \mathbf{A} = 1 \odot (2 \odot 2 \ominus 1 \odot 1) \ominus 2 \odot (2 \odot 2 \ominus 0 \odot 1) \oplus 0 \odot (2 \odot 1 \ominus 2 \odot 0) = 1 \tag{1.55}$$

so the matrix is invertible. Hence any equation of the form $\mathbf{Ax} = \mathbf{b}$ has a unique solution $\mathbf{x}$, for example if $\mathbf{b} = [1,0,0]^T$, we can use Gaussian elimination

$$\left[\begin{array}{ccc|c} 1 & 2 & 0 & 1 \\ 2 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 \end{array}\right] \tag{1.56}$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 \end{array}\right] \tag{1.57}$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{array}\right] \tag{1.58}$$

and hence we resolve $\mathbf{x} = [0, 2, 2]^T$. We can also find the matrix inverse the usual way

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 1 & 0 & 0 \\ 2 & 2 & 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \end{array}\right] \tag{1.59}$$

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \end{array}\right] \tag{1.60}$$

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 2 & 1 \end{array}\right] \tag{1.61}$$

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 1 \end{array}\right] \tag{1.62}$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 2 & 2 \\ 0 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 1 \end{array}\right] \tag{1.63}$$

hence

$$\mathbf{A}^{-1} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix} \tag{1.64}$$

Note that there are no concerns with numerical stability when operating in finite fields. This may sound like a mere technicality but is in fact the fundamental reason why we can design and operate excellent error correction codes in finite fields, but most attempts to devise codes in real and complex numbers have not been successful so far.

The characteristic polynomial of $\mathbf{A}$ is

$$\det(\mathbf{A} - \lambda\mathbf{I}) = (1-\lambda)\left((2-\lambda)(2-\lambda) - 1\right) - (2-\lambda) \tag{1.65}$$

where we have stopped using the cumbersome $\oplus, \odot$ notation since it should by now be clear that we are operating in GF(3). This polynomial has no roots in GF(3) as can immediately be seen by noting that neither $\lambda = 0$, $\lambda = 1$ or $\lambda = 2$ make its value 0. Hence, this matrix has no eigenvalues or eigenvectors in GF(3), or in other words there exists no λ and $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$.

The existence of eigenvectors and eigenvalues is closely related to the existence of roots of 1 in a field. In the complex field $\mathbb{C}$, for any n, there exists an element of order n, i.e., an element α for which $\alpha^n = 1$ and $\alpha^k \neq 1$ for $k < n$. This is simply $\alpha = e^{2\pi i/n}$. This is the reason why polynomial roots and hence eigenvalues and eigenvectors always exist in the complex field. In finite fields, roots of 1 exist only for certain values as we have learned from Lagrange's theorem 1.14. We will see in the next chapter (when studying Reed Solomon codes) that the existence of a Discrete Fourier Transform is also closely linked to the existence of roots of 1.

We conclude this chapter with one of the most exciting and possibly unexpected twists of finite field algebra. So far we only considered one type of finite field GF(p) the field of addition and multiplication modulo a prime number. Is this the only finite field there is? There are many other examples of fields with a prime number of elements known to mathematicians but all of them can be shown to be essentially equivalent to GF(p), i.e., for any field $\langle S, +, \cdot \rangle$ with p elements, there exists a function f mapping from S to $\mathbb{Z}_p$ such that, for any $a, b \in S$, $f(a+b) = f(a)+f(b)$ and $f(a \cdot b) = f(a) \cdot f(b)$, where the operations on the left of the equality are in S and the operations on the right are in GF(p). Mathematicians would say that all finite groups with a prime number p of elements are *isomorphic.* But are there any finite fields with a non-prime number of elements? We already concluded that $\mathbb{Z}_m$ is never a field when m is not prime. Nevertheless and somewhat surprisingly, there are prime fields with $q = p^k$ elements where p is prime. These fields are called Galois fields of order q or GF(q), so GF(q) is defined for q prime or any power of a prime number. The trick to generate Galois fields of non-prime orders p^k consists in extending the base field GF(p) in a very similar way that the field of real numbers $\mathbb{R}$ is extended to the field of complex numbers $\mathbb{C}$. We will spend the rest of this chapter learning about *extension fields.* A side-benefit of this learning process is that, if you feel (as I used to feel before learning algebra) that you learned how to tolerate complex numbers as a necessary evil, but never quite got why this bizarre extension of real numbers into the "imaginary" is at all necessary, you may now finally understand why extending real numbers in this way is a natural step, and why you gain such a rich algebraic structure by doing so.

We already know that addition and multiplication modulo p^k does not give us a field (remember for example that $2 \odot_4 2 = 0$). Hence, we need to define a very different operation over a set of p^k elements to obtain a field. There are two equivalent ways to do this and we will study both of them. The first approach is to consider *polynomials* over GF(p). In this approach, an element of GF($q = p^k$) is a polynomial of degree at most $(k-1)$ with coefficients in GF(p), e.g., and element $a(X) \in \text{GF}(p^k)$

$$a(X) = a_0 + a_1 X + a_2 X^2 + \ldots + a_{k-1} X^{k-1} \tag{1.66}$$

where $a_0, a_1, \ldots, a_{k-1} \in \text{GF}(p)$. Note that the indeterminate X has no meaning in this context and is simply an auxiliary variable that we need in order to define our operations, as we will see shortly. We could use any symbol for our indeterminate, and indeed the extension from real numbers to complex numbers uses the symbol i (or j if you are an engineer. . .) and considers all elements of $\mathbb{C}$ to be polynomials of degree 1 or less with real coefficients.

Example: $1 + X + X^3 + X^5$ is an element of GF($2^6 = 64$) and of GF(2^k) for any $k \geq 6$. $2 + X^2 + 6X^3$ is an element of GF($7^4 = 2401$) and of GF(7^k) for any $k \geq 4$.

If we write GF(1771561), it is clear without context that we speak about polynomials of degree 5 and less with coefficients in GF(11) because the number of 1771561 has a unique prime decomposition as 11^6. On the other hand, if anyone writes "GF(6)" you immediately know that it's a mistake because 6 is not a prime power and there exists no finite field with 6 elements.

Addition in an extension field is simply the addition of polynomials ,i.e.,

$$a(X) + b(X) = (a_0 + b_0) + (a_1 + b_1)X + (a_2 + b_2)X^2 + \ldots + (a_{k-1} + b_{k-1})X^{k-1}, \quad (1.67)$$

where the additions within the parentheses are over GF(p). Adding polynomials results in a polynomial of degree at most the larger of the two, and hence the field is closed under polynomial addition.

Multiplication in an extension field is defined as polynomial multiplication *modulo an irreducible polynomial* $\pi(X)$ of degree k. An irreducible polynomial is a polynomial that cannot be expressed as a product of two polynomials of lesser degree. It is the polynomial equivalent of a prime number. If X has order $p^k - 1$ under multiplication modulo $\pi(X)$, then $\pi(X)$ is called a *primitive* polynomial. All primitive polynomials are irreducible but not vice-versa. Any irreducible polynomial can be used to define an extension field, but using a primitive polynomial ensures that the generator of the multiplicative cyclic group is X, which has some practical advantages. For the rest of this chapter, we will always assume that a primitive polynomial $\pi(X)$ is used to define an extension field.

Example: we can use all the familiar prime number search techniques to find irreducible polynomials. For example, we can pick a candidate polynomial and try to divide it by all polynomials of smaller degree. Or we can approach this "bottom up" using the "sieve of Eratosthenes" technique and eliminate all polynomials that are obtainable as a product of polynomials. In our search, we can exclude any polynomials that don't have a leading 1 because they are obviously divisible by the polynomial X.

To extend GF(2), first consider that $\pi_1(X) = X$ and $\pi_2(X) = 1 + X$ are both irreducible polynomials of degree 1. The list of candidates to be irreducible polynomials of degree 2 is $1+X^2$ and $1+X+X^2$. We can compute $(1+X)^2 = 1+0X+X^2 = 1+X^2$ and hence exclude our first candidate to find that $\pi_2(X) = 1 + X + X^2$ is the only irreducible polynomial of degree 2. The list of candidates to be irreducible polynomials of degree 3 is $1 + X^3, 1 + X^2 + X^3, 1 + X + X^3, 1 + X + X^2 + X^3$. We compute

$$(1 + X)(1 + X + X^2) = 1 + X + X^2 + X + X^2 + X^3 = 1 + X^3 \text{ and} \quad (1.68)$$

$$(1 + X)(1 + X^2) = 1 + X + X^2 + X^3 \quad (1.69)$$

so the primitive polynomials of degree 3 are $1 + X^2 + X^3$ and $1 + X + X^3$. For degree

4, we compute all products of degrees 1,2,3 polynomials

$$(1+X)(1+X^3) = 1+X+X^3+X^4 \tag{1.70}$$
$$(1+X)(1+X^2+X^3) = 1+X+X^2+X^4 \tag{1.71}$$
$$(1+X)(1+X+X^3) = 1+X^2+X^3+X^4 \tag{1.72}$$
$$(1+X)(1+X+X^2+X^3) = 1+X^4 \tag{1.73}$$
$$(1+X^2)^2 = 1+X^4 \tag{1.74}$$
$$(1+X+X^2)^2 = 1+X^2+X^4 \tag{1.75}$$
$$(1+X^2)(1+X+X^2) = 1+X+X^3+X^4 \tag{1.76}$$

and find that the irreducible polynomials of degree 4 are $1+X^3+X^4$, $1+X+X^4$ and $1+X+X^2+X^3+X^4$. It is easy to verify that all the irreducible polynomials of degree up to 4 that we've stated so far are primitive (by checking that X has maximum order.)

There are many published tables of irreducible or primitive polynomials, so there is no need to do the searches above manually if you're ever in need of such polynomials. See for example `http://www.hpl.hp.com/techreports/98/HPL-98-135.pdf`.

While multiplication modulo a number m was simply defined as $a \odot_m b = \mathrm{R}_m(ab)$, i.e., you take integer multiplication and then subtract m a number of times to get back within the range $\{0, 1, 2, \ldots, m-1\}$, this approach does not quite map to polynomial multiplication modulo a primitive polynomial. For example in GF(8), if we multiply two polynomials of degree 2 we obtain a polynomial of degree 4 and no matter how many times we subtract our primitive polynomial $\pi(X)$ of degree 3, we will never get rid of the coefficient of X^4. The trick to define modulo arithmetic with polynomials is to operate degree changes one at a time, i.e.,

$$a(X)b(X) = (a_0 + a_1X + a_2X^2 + \ldots + a_{k-1}X^{k-1})b(X) \tag{1.77}$$
$$= a_0b(X) + a_1Xb(X) + a_2X^2b(X) + \ldots + a_{k-1}X^{k-1}b(X) \tag{1.78}$$

and hence we see that all we need here is to multiply powers of X by $b(X)$. We do this gradually by only multiplying up to the power of the primitive polynomial $\pi(X)$ and then removing the leading coefficient by subtracting a factor of $\pi(X)$ to get back within the range of polynomials of degrees $k-1$ or less, etc.

Example: we wish to compute $(1+X^3)(1+X^2+X^3)$ over GF(16) with primitive polynomial $\pi(X) = 1 + X + X^4$. We compute

$$X(1 + X^2 + X^3) = X + X^3 + X^4 + (1 + X + X^4) = 1 + X^3 \quad (1.79)$$

$$X^2(1 + X^2 + X^3) = X(1 + X^3) = X + X^4 + (1 + X + X^4) = 1 \quad (1.80)$$

$$X^3(1 + X^2 + X^3) = X \cdot 1 = X \quad (1.81)$$

where we used the fact that minus and plus are equivalent in GF(2). Hence

$$(1+X^3)(1+X^2+X^3) = 1\cdot(1+X^2+X^3)+X^3(1+X^2+X^3) = 1+X+X^2+X^3. \quad (1.82)$$

To summarise, operations over GF(p^k) are defined as polynomial addition with coefficients in GF(p) and polynomial multiplication modulo a primitive polynomial $\pi(X)$. Note that although it can be shown that all fields constructed in this manner are isomorphic, it's little consolation to a practicing engineer because there is no easy way to compute the function that maps from one field to another, so for practice it is essential when working in GF(p^k) to specify the primitive polynomial $\pi(X)$ used to define multiplication.

Before we progress to the second equivalent approach for defining operations over GF(p^k), we pause to consider a method that can make it easier to compute multiplications on small-ish fields. The trick consists in picking any generator, i.e., an element α of maximum multiplicative order $p^k - 1$ (which can for example be X if $\pi(X)$ is primitive) and to pre-compute all of its powers $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{p^k-2}$. Armed with such a lookup table, it becomes trivial to compute a product since

$$a(X) \cdot b(X) = \alpha^{k_a}\alpha^{k_b} = \alpha^{R_{p^k-1}(k_a+k_b)} \quad (1.83)$$

so multiplication in GF(p^k) is equivalent to addition in $\mathbb{Z}_{p^k-1}$.

Example: Consider GF(16) with multiplication modulo $\pi(X) = 1 + X + X^4$ and pick for example $\alpha = X$, then

$$
\begin{aligned}
X^0 &= 1 && (1.84)\\
X^1 &= X && (1.85)\\
X^2 &= X^2 && (1.86)\\
X^3 &= X^3 && (1.87)\\
X^4 &= 1 + X && (1.88)\\
X^5 &= X(1 + X) = X + X^2 && (1.89)\\
X^6 &= X(X + X^2) = X^2 + X^3 && (1.90)\\
X^7 &= X(X^2 + X^3) = 1 + X + X^3 && (1.91)\\
X^8 &= X(1 + X + X^3) = 1 + X^2 && (1.92)\\
X^9 &= X(1 + X^2) = X + X^3 && (1.93)\\
X^{10} &= X(X + X^3) = 1 + X + X^2 && (1.94)\\
X^{11} &= X(1 + X + X^2) = X + X^2 + X^3 && (1.95)\\
X^{12} &= X(X + X^2 + X^3) = 1 + X + X^2 + X^3 && (1.96)\\
X^{13} &= X(1 + X + X^2 + X^3) = 1 + X^2 + X^3 && (1.97)\\
X^{14} &= X(1 + X^2 + X^3) = 1 + X^3 && (1.98)\\
X^{15} &= X(1 + X^3) = 1 && (1.99)
\end{aligned}
$$

where the last line was only included to verify that all's well and as it should be in Galois-land. A few remarks:

- obviously, if the order of the multiplicative Abelian group is prime, such as is the case for GF(8) where $p^3 - 1 = 7$ is prime, then any element has maximum order and you can pick any element as your generator α

- X is not always a generator and it was the luck of our pick of $1 + X + X^4$ as our primitive polynomial that X turned out to be a generator. In small fields, most practicioners tend to pick a primitive polynomial $\pi(X)$ so that X is a generator. If we had picked $\pi(X) = 1 + X + X^2 + X^3 + X^4$, then $X^5 = 1$ and hence X has order 5 and cannot be used to draw a multiplication table. On the other hand, $\alpha = 1 + X$ is a generator in this case.

The second equivalent approach to defining operations in extension fields is to consider elements of GF(p^k) as k-ary row vectors over GF(p). We will initially assume that $\alpha = X$ is a generator of the multiplicative group, i.e., X has order $p^k - 1$. In this case, the row vector just lists the coefficient of the equivalent polynomial. For example, the element

$a(X) = 1 + X^2 + X^3$ of GF(16) can be written as

$$\mathbf{a} = [1, 0, 1, 1]. \tag{1.100}$$

Addition is simply component-wise addition of vectors over GF(p) very much the way vector addition is always defined, so no suprise here.

For multiplication on the other hand, we need to introduce the concept of a *companion matrix*. Each element a of the field has an alternative representation as a matrix $\mathbf{A}$, in addition to its representation as a vector $\mathbf{a}$. The product is evaluated as a vector-matrix product, e.g., $a \cdot b = \mathbf{aB}$. Let us consider for example the elements $a(X) = 1 + X + X^3$ and $b(X) = 1 + X^2$ of GF(16) with $\pi(X) = 1 + X + X^4$. The vector notation for those is

$$\mathbf{a} = [1, 1, 0, 1] \text{ and} \tag{1.101}$$
$$\mathbf{b} = [1, 0, 1, 0]. \tag{1.102}$$

The companion matrix for b is

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{1.103}$$

When we evaluate $a \cdot b$ as a vector-matrix product, we see that

$$\mathbf{aB} = [1101] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = a_0\mathbf{b_0} + a_1\mathbf{b_1} + a_2\mathbf{b_2} + a_3\mathbf{b_3}, \tag{1.104}$$

where we wrote a_m for the m-th component of the vector $\mathbf{a}$ and $\mathbf{b_m}$ for the m-th row of the matrix $\mathbf{B}$. Mirroring (1.104) in polynomial notation, we are effectively computing

$$a(X)b(X) = a_0 b(X) + a_1 \left(Xb(X)\right) + a_2 \left(X^2 b(X)\right) + a_3 \left(X^3 b(X)\right). \tag{1.105}$$

Hence the rows of the matrix $\mathbf{B}$ are in fact nothing but the pre-computed products of $\mathbf{b}$ by $1, X, X^2$ and X^3. We can neatly read out the companion matrices in the product lookup table we wrote for GF(16) with $\pi(X) = 1 + X + X^4$ in Equations (1.84) to (1.99), which we reproduce here again with the vector notation added in

X^1	X	$[0,1,0,0]$
X^2	X^2	$[0,0,1,0]$
X^3	X^3	$[0,0,0,1]$
X^4	$1+X$	$[1,1,0,0]$
X^5	$X+X^2$	$[0,1,1,0]$
X^6	X^2+X^3	$[0,0,1,1]$
X^7	$1+X+X^3$	$[1,1,0,1]$
X^8	$1+X^2$	$[1,0,1,0]$
X^9	$X+X^3$	$[0,1,0,1]$
X^{10}	$1+X+X^2$	$[1,1,1,0]$
X^{11}	$X+X^2+X^3$	$[0,1,1,1]$
X^{12}	$1+X+X^2+X^3$	$[1,1,1,1]$
X^{13}	$1+X^2+X^3$	$[1,0,1,1]$
X^{14}	$1+X^3$	$[1,0,0,1]$
X^{15}	1	$[1,0,0,0]$

(1.106)

The companion matrix for the element X^m consists of the 4 rows in the table starting at row m, where the table is interpreted cyclically. For example, the companion matrix for X^4 consists of the 4th, 5th, 6th and 7th rows $[1,1,0,0]$, $[0,1,1,0]$, $[0,0,1,1]$ and $[1,1,0,1]$, respectively, corresponding to $1 \cdot X^4$, $X \cdot X^4$, $X^2 \cdot X^4$ and $X^3 \cdot X^4$. Similarly, the companion matrix for X^{14} consists of the 14th, 15th, 1st and 2nd rows $[1,0,0,1]$, $[1,0,0,0]$, $[0,1,0,0]$ and $[0,0,1,0]$, respectively.

Note that the matrix product remains consistent with the product in $\mathrm{GF}(p^k)$ as a consequence of associativity since

$$a \cdot b \cdot c = (\mathbf{aB})\mathbf{C} = \mathbf{a}(\mathbf{BC})$$

and hence $\mathbf{BC}$ must be the companion matrix of $b \cdot c$. This means that we can use simple modulo p matrix multiplication to construct the multiplication table if we are just given the companion matrix of X, which we call $\mathbf{X}$ by a slight abuse of notation. This matrix always has the shape

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ & & -\pi^\star & & \end{bmatrix}, \tag{1.107}$$

i.e., a matrix with an off-diagonal of 1s above the main diagonal, and the last row $-\pi^\star$ contains minus the coefficients of $\pi(X)$ without the leading coefficient. This shape follows from the fact that multiplication of a polynomial $a(X)$ by X simply shifts the coefficients of X^0 to X^{k-2} up by one position, and any coefficient a_{k-1} of X^{k-1} triggers a division by the primitive polynomial $\pi(X)$ which is obtained by subtracting $a_{k-1}\pi(X)$ to get rid of the coefficient of X^k. Since addition and subtraction is the same in GF(2), there is no need for a minus operation when operating in GF(2^k).

To summarise what we've found, for two elements a and b in GF(q) defined via the primitive polynomial $\pi(X)$, we construct the matrix $\mathbf{X}$ as speficied in (1.107), and can compute the companion matrix $\mathbf{C}$ of the GF(q) product $c = a \cdot b$ as

$$\mathbf{C} = (a_0\mathbf{I} + a_1\mathbf{X} + a_2\mathbf{X^2} + \ldots + a_{k-1}\mathbf{X^{k-1}})(b_0\mathbf{I} + \ldots + b_{k-1}\mathbf{X^{k-1}}) \tag{1.108}$$

and its vector representation or polynomial coefficients as $\mathbf{c} = [1, 0, \ldots, 0]\mathbf{C}$ to read out the first row of $\mathbf{C}$, since the first row of a companion matrix is the vector representation of the corresponding element. These are all matrix operations in the underlying prime field (e.g., GF(2)) and hence easy to evaluate in any programming environment that enables matrix operations and modulo arithmetic such as Python, MATLAB, Octave, etc. without the need for further libraries to implement polynomial multiplication modulo $\pi(X)$. Note that $\mathbf{X}^0 = \mathbf{X}^{p^k-1} = \mathbf{I}$ is the identity matrix.

If we work in a field where X is not a generator, e.g., GF(16) with $\pi(X) = 1+X+X^2+X^3+X^4$ for which $\alpha = 1 + X$ is a generator, it is possible to also operate in companion matrix notation by expressing all field elements as polynomials in the generator polynomial α, e.g.,

$$a(\alpha) = a'_0\alpha^0 + a'_1\alpha + \ldots + a'_{k-1}\alpha^{k-1}. \tag{1.109}$$

One has to be careful not to confuse the coefficients $a'_0, a'_1, \ldots, a'_{k-1}$ that multiply powers of the generator α with the original coefficients $a_0, a_1, \ldots, a_{k-1}$ that define the field elements in the variable X, but since polynomial addition remains an element-wise addition in the new indeterminate α, we need never know about the "original" coefficients and can instead view the field as defined through polynomials in α. In a way, this is nothing more than applying an isomorphism from the original field where X is not a generator to a field where X, now called α, is a generator.

One essential consequence of the companion matrix view of extension fields is that any matrix operation in GF(p^k) is in fact a matrix operation in GF(p) where you replace each element in the GF(p^k) matrix by its companion matrix. For example, the operation

$$[1, 1+X]\begin{bmatrix} X & 1+X^2 \\ X^3 & 1+X+X^3 \end{bmatrix} = [1+X^3, 1+X+X^3] \tag{1.110}$$

over GF(16) with $\pi(X) = 1 + X + X^4$ can be written equivalently as

$$[1,0,0,0,\ 1,1,0,0]\left[\begin{array}{cccc|cccc} 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 0&0&0&1&1&1&1&0 \\ 1&1&0&0&0&1&1&1 \\ \hline 0&0&0&1&1&1&0&1 \\ 1&1&0&0&1&0&1&0 \\ 0&1&1&0&0&1&0&1 \\ 0&0&1&1&1&1&1&0 \end{array}\right] = [1,0,0,1,\ 1,1,0,1] \tag{1.111}$$

over GF(2), where we have added lines in the matrix to illustrate the boundaries of the companion matrices but these lines aren't actually there, i.e., the matrix is understood as an 8×8 matrix in GF(2). The consequence of this parallel between matrix operations in $\mathrm{GF}(p^k)$ and matrix operations in $\mathrm{GF}(p)$ is that linear codes over $\mathrm{GF}(p^k)$ that we will study in the next chapter are essentially codes over $\mathrm{GF}(p)$ and the view of the codes in $\mathrm{GF}(p^k)$ is only helpful because of the structure it gives us to enable efficient decoding, or because the channel affects packets of k symbols at a time.

1.3 Problems for Chapter 1

Problems marked as $\star$ are typical exam questions. Problems marked 😎 are borrowed with thanks from Jim Massey's 1990s lectures at ETH Zurich.

Problem 1.3.1: Remainders ($\star$)

Calculate the remainder $R_{11}(5 \times 10^{27} + 256)$.

Problem 1.3.2: Decimal representation (😎)

When we write a nonnegative integer i in decimal notation (i.e., in radix-10 form) as $i_m i_{m-1} \cdots i_1 i_0$, where $0 \leq i_k < 10$, we mean of course that

$$I = i_m 10^m + i_{m-1} 10^{m-1} + \cdots + i_1 10 + i_0.$$

In the following, you will make use of the "algebraic properties of remainders" to establish some interesting properties of such decimal representations.

(a) Find $R_2(I)$ in terms of the digits $i_0, i_1, \ldots, i_{m-1}, i_m$ in the decimal form of I.

Note: You should observe that you have proved the not-very-surprising fact that I is divisible by 2 if and only if i_0 is divisible by 2.

(b) Find $R_3(I)$ in terms of the digits in the decimal form of I.

Note: You should now observe the more surprising fact that I is divisible by 3 if and only if $i_0 + i_1 + \cdots + i_{m-1} + i_m$ is divisible by 3.

(c) Make use of your results in a) and b) to show that certain integers in the following list cannot be primes: 89, 91, 178, 179, 183, 3779, 398421.

Note: There are four non-primes in this list. You should have identified three of these non-primes – which is the remaining non-prime?

Problem 1.3.3: Casting out nines (😎)

"Casting out nines" is a method for detecting certain errors in the addition of nonnegative decimal integers that was popular in pre-calculator days. According to this method, one adds "modulo 9" to all the digits in all of the integers to be summed, then compares this result to the "modulo 9" sum of the digits in the purported answer. If these two sums disagree, then the purported answer is erroneous. As this problem will show, this method of "casting out nines" is a simple consequence of the "algebraic properties of remainders".

(a) Find $R_9(10^n)$ for all $n \geq 0$.

(b) Find $R_9(I)$ in terms of the digits $i_0, i_1, \ldots, i_{m-1}, i_m$ in the decimal form of the nonnegative integer I, i.e., when

$$I = i_m 10^m + i_{m-1} 10^{m-1} + \cdots + i_1 10 + i_0.$$

(c) Now prove that the "modulo 9" sum of all of the digits in the decimal forms of the nonnegative integers $I_1, I_2, \ldots, I_M$ must equal the "modulo 9" sum of the digits in the decimal form of the sum S of these M integers.

(d) Use the method of "casting out nines" to show that some of the following sums are erroneous:

$$\begin{array}{rrrr}
7894 & 5418 & 3397 & 7695 \\
3972 & 7997 & 2789 & 8143 \\
4158 & 4582 & 9999 & 3981 \\
\underline{+6579} & \underline{+9678} & \underline{+4837} & \underline{+7009} \\
22603 & 27675 & 21112 & 27928
\end{array}$$

Are the "possibly correct" sums all correct?

(e) The purported sum S' can be written as $S + E$ where S is the true sum and E is the error. Find the necessary and sufficient condition for a non-zero error E to go undetected by the method of "casting out nines".

(f) Suppose that S and S' disagree in a single digit. Show that "casting out nines" will detect such an error unless these disagreeing digits are 0 and 9 or 9 and 0.

Hint: What is E?

(g) Suppose that S and S' disagree because of a single "carry error" made while performing the addition (which can cause several digits of S and S' to disagree). Suppose the true and erroneous carries are c and c', respectively. Show that "casting out nines" will detect such an error unless $c - c'$ is divisible by 9.

(h) Because $R_3(10^n) = 1$ for all $n \geq 0$, one could also check decimal addition by "casting out threes". It is certainly easier to cast out threes than to cast out nines. How then do you explain the fact that no one ever used "casting out threes" in pre-calculator days?

Hint: Consider f) and g) above.

Problem 1.3.4: Greatest common divisor property

Show that, for any integers x and y,

$$\gcd(2^x - 1, 2^y - 1) = 2^{\gcd(x,y)} - 1.$$

First show that for any d and n, if d divides n, $2^d - 1$ divides $2^n - 1$. You may want to use the expression for the sum of a geometric series in the data book in your proof. This step shows that $2^{\gcd(x,y)} - 1$ is a common divisor of $2^x - 1$ and of $2^y - 1$. You then need to show that $2^x - 1$ and $2^y - 1$ both divide $2^{\gcd(x,y)} - 1$ using the greatest common divisor theorem.

Problem 1.3.5: Euclid and Stein (😎, $\star$)

(a) Find $\gcd(1365, 1092)$ by the use of Euclid's algorithm. Find this same greatest common divisor by the use of Stein's algorithm.

(b) Find $\gcd(1081, 897)$ and integers a and b such that $\gcd(1081, 897) = a(1081) + b(897)$ by the use of the extended Euclidian algorithm.

(c) Find a different pair of integers a' and b' such that $\gcd(1081, 897) = a'(1081) + b'(897)$. Verify that $R_{m_1}(a) = R_{m_1}(a')$ and that $R_{m_2}(b) = R_{m_2}(b')$, where $m_1 = 897/\gcd(1081, 897)$ and $m_2 = 1081/\gcd(1081, 897)$.

Problem 1.3.6: Proof of the greatest common divisor theorem

This problem will lead you through a proof of the greatest common divisor theorem. The aim is to prove that for any integers n_1, n_2, there exist integers a and b such that $\gcd(n_1, n_2) = an_1 + bn_2$. Consider the set

$$\mathcal{D}(n_1, n_2) = \{d : R_d(n_1) = R_d(n_2) = 0\}$$

of common divisors of n_1 and n_2. The greatest common divisor $d^\star = \gcd(n_1, n_2) = \max \mathcal{D}(n_1, n_2)$ is the largest element of that set. Consider also the set

$$\mathcal{C}(n_1, n_2) = \{c = an_1 + bn_2, \forall a, b\}$$

of integers that can be written as $an_1 + bn_2$ (this set is called an *ideal*), and let

$$c^\star = \min\{c \in \mathcal{C}(n_1, n_2), c > 0\}$$

be the smallest positive number that can be expressed as $an_1 + bn_2$.

(a) Show that $d^\star$ divides $c^\star$, and since they are both positive, hence $d^\star \leq c^\star$.

(b) Express $r = R_{c^\star}(n_1)$ as $n_1 - qc^\star$ and hence write r as an integer combination of n_1 and n_2. Use this expression and the fact that $0 \leq r < c^\star$ to conclude that $r = 0$.

(c) Using the result of the previous question, conclude that $c^\star \in \mathcal{D}(n_1, n_2)$, and hence that $c^\star \leq d^\star$.

(d) Combine the results of the previous questions to complete the proof.

This proof shows a little more than just the statement of the theorem: it shows that the greatest common divisor is the smallest positive element of the ideal $\mathcal{C}(n_1, n_2)$ of numbers that can be expressed as an integer combination of n_1 and n_2.

Problem 1.3.7: Music Theory

An octave in music theory corresponds to a doubling of the frequency, i.e., if middle C on a piano is pitched at $f_C = 261.626$ Hz, the next C on the piano should be pitched at double this frequency, or $2f_C = 523.252$ Hz. Similarly, the intervals known as fifth, fourth, major third and minor third are meant to be pitched at frequency ratios of 3/2, 4/3, 5/4 and 6/5, respectively, i.e., the notes G, F, E and E♭ at a fifth, a fourth, a major third and a minor third from middle C should be at frequencies $3f_C/2$, $4f_C/3$, $5f_C/4$ and $6f_C/5$, respectively.

(a) Use number theory to show that no integer number of perfect fifths can be equal to an integer number of perfect octaves.

(b) Despite the above, we are taught that 12 fifths (the progression C, G, D, A, E, B, F♯, C♯, A♭, E♭, B♭, F, C) returns us to a C and hence completes 7 octaves. This is only approximately true and is the reason why pianos cannot be perfectly tuned. What two integers are close enough to warrant the fact that 12 fifths are approximately 7 octaves?

(c) If every half tone on a piano is tuned to a ratio of $2^{1/12}$ above the preceding half-tone, show how the perfect interval ratios $2/1, 3/2, 4/3, 5/4$ and $6/5$ are approximated for an octave, a fifth, a fourth, a major third and a minor third, respectively? Which one of those is exact? A piano tuned in this manner is said to use equal temperament. It is the prevalent method of tuning nowadays. Other tuning methods include Pythagorean tuning, that maintains perfect fiths across all octaves, and just intonation, that maintains all correct intervals but can only be achieved in instruments with a limited range.

Acknowledgment: this problem is inspired from Ueli Maurer's lecture notes for his lecture "Discrete Mathematics" currently taught to year 1 computer scientists at ETH Zurich.

Problem 1.3.8: Inverses (😎)

Recall that an element u in $\mathbb{Z}_m$ has inverse in $\mathbb{Z}_m$ if and only if $\gcd(m, u) = 1$.

(a) Find the inverses – if they exist – of $u = 19$ and $u = 21$ in the ring $\mathbb{Z}_{35}$ by the use of the extended Euclidian algorithm.

(b) Using the extended Euclidian algorithm, compute the inverses of $u = 90$ and $u = 111$ in the ring $\mathbb{Z}_{11111}$.

Problem 1.3.9: Euler's function(😎, $\star$)

Find Euler's function $\varphi(n)$ for each of the following values of n:

(a) $n = 7$;

(b) $n = 49$;

(c) $n = 35$;

(d) $n = 63$.

Problem 1.3.10: Telescopic Chinese Remainder inversion[5]

In the lecture, we gave a constructive way of infering a number n from its residuals $(r_1, \ldots, r_k)$, as summarised in (1.26). While this approach is easily amenable to computer implementation, it is not easy to compute by hand for small moduli without the help of a calculator. An easier approach that is typically amenable to at least partial solution by mental arithmetic is based on considering the "telescopic number",

$$N = c_1 + c_2 m_1 + c_3 m_1 m_2 + c_4 m_1 m_2 m_3 + \ldots + c_k \prod_{i=1}^{k-1} m_i.$$

The aim of the contruction is to determine a set of coefficients for which $n = R_m(N)$ is the number between 0 and $m - 1$ with residuals $(r_1, \ldots, r_k)$.

(a) Show that c_1 can be determined by simply considering $R_{m_1}(N)$ and does not depend on $m_2, m_3, \ldots$.

(b) Show that, with c_1 already determined, c_2 can be determined by considering $R_{m_2}(N)$ and does not depend on $m_3, m_4, \ldots$.

(c) Describe the simple telescopic[6] "algorithm" that emanates from the two steps above.

[5] This question is inspired by feedback received from Andreas Theocharous who took 4F5 in Lent 2021.

[6] We call it "telescopic" because it unfolds into separable computations of increasing difficulty.

(d) Use this algorithm to determine the number between 0 and 1154 with residuals $(1, 3, 2, 1)$ with respect to the moduli $(3, 5, 7, 11)$.

(e) You may want to implement the original method from (1.26) for comparison.

Problem 1.3.11: Chinese Remainder Theorem (😎, $\star$)

(a) Compute the Chinese Remainder Theorem residuals of $n \oplus \tilde{n}$ and compare it to the residuals of n and residuals of $\tilde{n}$ for the moduli $m = m_1 \cdot m_2 \cdot m_3$ in the following cases:

(i) $m_1 = 2$, $m_2 = 3$, $m_3 = 5$, $n = 7$, $\tilde{n} = 24$;

(ii) $m_1 = 9$, $m_2 = 25$, $m_3 = 256$, $n = 2047$, $\tilde{n} = 4100$.

(b) Prove the validity of the following statement: Suppose that the integers $m_1, m_2, \ldots, m_k$ are pairwise relatively prime moduli, that $m = m_1 \cdot m_2 \cdot \ldots \cdot m_k$, and that n and $\tilde{n}$ are elements of $\mathbb{Z}_m$ with CRT residuals $(r_1, r_2, \ldots, r_k)$ and $(\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_k)$, respectively. Then the residuals of the sum $n \oplus \tilde{n}$ in $\langle \mathbb{Z}_m, \oplus \rangle$ is $(r_1 \oplus \tilde{r}_1, r_2 \oplus \tilde{r}_2, \ldots, r_k \oplus \tilde{r}_k)$, the componentwise sum of the residuals of n and $\tilde{n}$, where the sum $r_i \oplus \tilde{r}_i$ is a sum in $\langle \mathbb{Z}_{m_i}, \oplus \rangle$.

(c) Repeat a) and b) with $\oplus$ replaced with $\odot$.

Problem 1.3.12: Arithmetic over Finite Fields ($\star$)

(a) Construct arithmetic over GF(16).

(i) Find an irreducible polynomial $\pi(X)$ of degree 4 with coefficients in GF(2).
Hint: you only need to verify that your polynomial is not divisible by any polynomials of degree 1 or 2, since division by a polynomial of degree 3 would result in a polynomial of degree 1.

(ii) Find an element α of multiplicative order 15 and list all the powers of α.
Hint: Try $\alpha = X$ or $\alpha = X + 1$.

(iii) Compute $(X^3 + X + 1) \times (X^2 + X)$ using polyomial multiplication modulo $\pi(X)$ and using companion matrices.

(b) Construct arithmetic over GF(4) using the first two steps of the previous question, and generate 4×4 multiplication and addition tables for all elements in GF(4).

Chapter 2

Linear Codes over GF(q)

We now proceed to use the mathematical tools we aquired in the previous chapter in designing error correcting codes. But first, we need to preface any discussion of error correction with a very serious disclaimer. Most of the students taking 4F5 have already had some exposure to binary linear codes in 3F7, when they learned about low-density parity-check (LDPC) codes. LDPC codes are one of a new generation of codes[1] that began with the invention of Turbo Codes in 1993 and whose aim is to provide arbitrary reliability at data rates approaching capacity for symmetric discrete-input memoryless channels. To your well-trained information theorist's ears, this probably sounds like a very reasonable aim for a channel coding system. However, you will be surprised to hear that designing codes with this aim was a total novelty and a bombshell in 1993, despite 45 years of active information theory research that had elapsed since Shannon's 1948 paper. Coding theorists until 1993 mainly concerned themselves with the task of *correcting errors*. The dominant line of thinking at the time was that communication engineers would work on providing reliable "bit pipes" but that the bit pipes sometimes made mistakes and that the task of the coding theorist was to detect and, if possible, correct erroneous bits. It did not seem to occur to anyone that deciding on the value of a bit based on soft channel observations without taking into account the structure of a code was a lossy operation, and the capacity of the resulting Binary Symmetric Channel (BSC) was irreparably lower than the original capacity of the channel. The same is true of non-binary channels. The error correction approach to coding is only optimal for Binary Symmetric Channels and equivalent non-binary symmetric channels, where the channel output alphabet equals the channel input alphabet and nearest neighbour decoding in the Hamming distance sense is optimal. The principles of error correction can also be extended to erasure channels, although one speaks of "recovery from erasures" rather than error correction in this context. For any other channel, the task of channel coding should never be seen as one of error correction because this approach is inherently sub-optimal.

Some modern coding theorists would hence argue that "error correction coding" has no

[1] Slightly confusingly, LDPC codes were originally invented by Bob Gallager in his PhD thesis in 1962, but their capacity-approaching properties weren't fully known until David MacKay re-discovered these codes in the mid 1990s at the Cavendish Lab in Cambridge.

place in modern communications courses. We would counter that this is taking too narrow a view of the communications problem. Shannon's mathematical theory of communications based on probability theory is wonderfully insightful and instructive, but there are tasks and scenarios in real-life systems that don't fit the theory exactly. In some cases, you genuinely just want to correct errors, and data rate maximisation is not your ultimate aim. For example, you may want a system that can provably recover from a fixed number of erasures or errors. Coding methods with probabilistic decoding such as LDPC codes have a good error correction capability on average but they do not offer good guarantees on the minimum number of errors that can be corrected. Sometimes the error probabilities your application requires are so low, e.g., train intercoms, air traffic control, data storage, that you cannot evaluate a system through simulation to verify that it achieves the desired probability. For example, if you are aiming for a bit error probability of 10^{-12}, then to accumulate sufficient statistics to evaluate the bit error rate you would have to simulate the transmission of at least 10^{14} bits which may be infeasible on a modern computer system within a reasonable time. In this context, it would help to add a layer of coding at each end of an LDPC or Turbo coding system that has a measured probability of error of, say, 10^{-6}, with a mathematically proven ability to lower the probability of error by 8 orders of magnitude so that no simulation is required to verify it. The added layer may not be rate-optimal, but as long as it's not a terribly inefficient code such as a repetition code, you may be willing to take the slight rate loss in exchange for a provable error performance.

In this chapter, we will hence look at traditional error correction and erasure recovery, where codes are designed over a q-ary alphabet and observations are over the q-ary alphabet (possibly with the addition of an "erasure" symbol) and our aim is to give the best estimate of the transmitted codeword given the received word. We will start with the fundamentals of linear coding that apply to error correction as well as modern channel coding. You have seen many of these definitions less formally in 3F7 but only in the context of binary codes, whereas we will now consider codes over any Galois field GF(q).

2.1 Linear coding fundamentals

2.1.1 Linear codes and encoder matrices

A q-ary (N, K) error correction code in the context of this chapter is a set $\mathcal{C}$ of q^K q-ary row vectors $\mathbf{x}$ of length N. In coding theory, we often call row vectors "words" and those vectors who belong to a code are called "codewords". Coding theory also distinguishes itself from the majority of reasonable disciplines by preferring to consider row vectors when everybody else tends to prefer column vectors (and below, when we consider matrix operations, the natural operation will hence be a vector times a matrix and not the usual matrix times vector.) Please don't be put off by this, and feel free to rant against the fathers of modern coding theory for adopting this silly convention. There is some sense in it when you consider vectors as words, since most coding theorists were originally from an Indo-European heritage where words in natural language are written horizontally.

The operating principle of error correction coding is to process information and to add redundancy so that errors can be corrected or erasures recovered from. Hence, the encoding operation consists in taking K q-ary symbols $\mathbf{u} = [u_1, u_2, \ldots, u_K]$ at a time and mapping them to a corresponding codeword $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ of length N, where $N > K$. In code design, we distinguish between the "encoding" operation, that maps information to codewords, and the code itself, that consists simply of the set $\mathcal{C}$ of codewords. Many performance properties of the coding system will depend only on the code, and one may choose from a variety of distinct encodings into the same code without affecting performance.

In this chapter, we will focus on *linear* codes, i.e., codes that can be generated by matrix operations. Operating over any Galois field GF(q), we can generate a code using a $K \times N$ matrix $\mathbf{G}$, called the *encoder matrix*, as the set of words

$$\mathbf{x} = \mathbf{uG} \tag{2.1}$$

for all row vectors $u \in \mathrm{GF}(q)^K$ of length K. The code in this setup is simply a vector space, namely the subspace of $\mathrm{GF}(q)^N$ generated by the rows of $\mathbf{G}$. Clearly, we can pick any K linearly independent codewords in the code to serve as a basis for the vector space, and any of these bases will give an equivalent alternative matrix $\mathbf{G}$ that generates the same code. Here, $\mathbf{G}$ serves the double purpose of defining the code *and* the encoding, since we can use $\mathbf{G}$ to map an information word to a codeword using (2.1). Any equivalent basis of the vector space gives the same code, but a *different encoding* of information words to codewords. Note that the encoder matrix is often called the *generator matrix* in textbooks. The term "encoder matrix" focuses on the fact that it specifies the encoding of information to codewords, whereas the term "generator matrix" focuses on the fact that it generates the code.

Example: operating in GF(3), consider the encoder matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}. \tag{2.2}$$

The matrix implements the following mapping

$$\begin{aligned}
[u_1, u_2] &\longrightarrow [x_1, x_2, x_3] \quad (2.3)\\
[0,0] &\longrightarrow [0,0,0] \quad (2.4)\\
[0,1] &\longrightarrow [1,0,2] \quad (2.5)\\
[0,2] &\longrightarrow [2,0,1] \quad (2.6)\\
[1,0] &\longrightarrow [1,2,0] \quad (2.7)\\
[1,1] &\longrightarrow [2,2,2] \quad (2.8)\\
[1,2] &\longrightarrow [0,2,1] \quad (2.9)\\
[2,0] &\longrightarrow [2,1,0] \quad (2.10)\\
[2,1] &\longrightarrow [0,1,2] \quad (2.11)\\
[2,2] &\longrightarrow [1,1,1]. \quad (2.12)
\end{aligned}$$

We can pick two other linearly independent codewords from the set of words on the right to obtain a different encoding for the same code, e.g.,

$$\mathbf{G}' = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{bmatrix} \quad (2.13)$$

and obtain the mapping

$$\begin{aligned}
[u_1, u_2] &\longrightarrow [x_1, x_2, x_3] \quad (2.14)\\
[0,0] &\longrightarrow [0,0,0] \quad (2.15)\\
[0,1] &\longrightarrow [0,1,2] \quad (2.16)\\
[0,2] &\longrightarrow [0,2,1] \quad (2.17)\\
[1,0] &\longrightarrow [1,0,2] \quad (2.18)\\
[1,1] &\longrightarrow [1,1,1] \quad (2.19)\\
[1,2] &\longrightarrow [1,2,0] \quad (2.20)\\
[2,0] &\longrightarrow [2,0,1] \quad (2.21)\\
[2,1] &\longrightarrow [2,1,0] \quad (2.22)\\
[2,2] &\longrightarrow [2,2,2]. \quad (2.23)
\end{aligned}$$

You can verify that the codewords in the second list are indeed the same as the codewords in the previous list, but their order is rearranged corresponding to a different encoding. Note that, since the first two columns of $\mathbf{G}'$ happen to form an identity matrix, the first two symbols of every codeword are simply the information symbols themselves. This is no coincidence and an encoder matrix with this property is called "systematic" as will be discussed below.

The following elementary operations on the rows of an encoder matrix will modify the encoding but leave the code unaffected:

- switch two rows
- multiply a row by a non-zero constant
- replace a row by the sum of itself and another row.

Using elementary row operations, it is always possible to bring an encoder matrix into *systematic form*

$$\mathbf{G_s} = \left[\begin{array}{cc} \mathbf{I}_K & \mathbf{P}_{K\times(N-K)} \end{array} \right] \tag{2.24}$$

where $\mathbf{I}_K$ denotes the $K \times K$ identity matrix. When using a systematic encoder matrix, the first K symbols of a codeword are the information symbols. This has the advantage that a communication system may be designed with an optional decoder that operates for example only when processing data that requires extra reliability, or when the battery has sufficient power to operate the decoding algorithm. When operating without the decoder, the systematic symbols can be read straight out of the channel with the channel error rate, while when the decoder is operational, code redundancy is used to correct errors and lower the error rate. There are other considerations, advantages and disadvantages to systematic encoding and a full discussion of those would take several chapters. One main advantage relates to the symbol error rate of the encoder. In this chapter, we will focus mainly on the block error performance of a code, i.e., what is the probability that the received word will be decoded to the wrong codeword or not decoded at all. For this criterion, only the code matters and the encoding is irrelevant[2]. However, if we are concerned with symbol error rate, then it matters to know, when the received word is decoded to a wrong codeword that is fairly similar to the transmitted codeword, how many information symbols will be affected. Systematic encoding ties the similarity of codewords to the similarity of information words, so that small errors in codewords result in small errors in information words, whereas other encodings may not conserve such similarities at all and result in higher symbol error rates for the same block error rate.

2.1.2 Parity-check matrices and dual codes

When thinking of finite vector spaces, it sometimes helps to think in analogy to 3-dimensional real vector spaces that most of us are familiar with since secondary school, and the next topic is one where this analogy helps. We learned that there are two ways one can define a plane through the origin in $\mathbb{R}^3$: you can either specify two basis vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ and express every point in the plane as a linear combination $\mathbf{v} = c_1\mathbf{b}_1 + c_2\mathbf{b}_2$; or, you can specify a normal vector $\mathbf{n}$ and define the plane as the set of vectors orthogonal to the normal vector, $\mathbf{v} \cdot \mathbf{n} = 0$. The first approach parallels what we did in the previous section.

[2] this is by no means trivial but you can probably work it out if you think about it, or else wait until we discuss Hamming distance later in this chapter and it will become clear to you then.

A code is nothing but a K-dimensional hyperplane in $\mathrm{GF}(q)^N$, and the rows of $\mathbf{G}$ form a basis of the hyperplane[3]. An alternative definition of a code defines the set of $N-K$ "normal" vectors that any codeword must be orthogonal to. This representation is called a parity-check matrix $\mathbf{H}$, and any codeword $\mathbf{x}$ must satisfy

$$\mathbf{x}\mathbf{H}^{\mathbf{T}} = 0. \tag{2.25}$$

The parity-check matrix is an $(N-K) \times N$ matrix. Note the transposition in (2.25) that reflects the fact that the rows of $\mathbf{H}$ are vectors in $\mathrm{GF}(q)^N$ and we are taking dot products with these vectors to verify orthogonality.

Example: the matrix

$$\mathbf{H} = [1, 1, 1] \tag{2.26}$$

is a parity-check matrix of the code in the example in the previous section. You can easily verify that the sum of symbols in every codeword is indeed 0.

Note that if the vectors $\mathbf{h_1}$ and $\mathbf{h_2}$ are orthogonal to all codewords, then any linear combination $c_1\mathbf{h_1} + c_2\mathbf{h_2}$ is also orthogonal to all codewords, since

$$\mathbf{x} \cdot (c_1\mathbf{h_1} + c_2\mathbf{h_2})^T = c_1\mathbf{x} \cdot \mathbf{h_1}^T + c_2\mathbf{x} \cdot \mathbf{h_2}^T = 0. \tag{2.27}$$

Hence, the set of vectors that are orthogonal to a code is itself a subspace of $\mathrm{GF}(q)^N$, and the rows of any parity-check matrix $\mathbf{H}$ form a basis of this space. The space generated by the rows of $\mathbf{H}$ is called the *dual code* and denoted $\mathcal{C}^{\perp}$. It follows that, like generator matrices, parity-check matrices are not unique and any basis of the dual code can be used as a parity-check matrix.

One question of interest is how do we go from an encoder to a parity-check matrix and vice versa? Starting from a $K \times N$ encoder matrix $\mathbf{G}$, the first step is to complete the matrix by picking $N-K$ linearly independent rows that are not codewords to form a matrix $\tilde{\mathbf{G}}$, resulting in a matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{G} \\ \tilde{\mathbf{G}} \end{bmatrix} \tag{2.28}$$

that has N linearly independent rows and hence is invertible. We can write its inverse as

$$\mathbf{M^{-1}} = \begin{bmatrix} \tilde{\mathbf{H}}^T & \mathbf{H}^T \end{bmatrix} \tag{2.29}$$

where $\mathbf{H}$ is a matrix we obtain by taking the last $N-K$ columns of the inverse matrix $\mathbf{M^{-1}}$ and transposing them. We will now show that this is indeed the parity-check matrix, by writing

$$\mathbf{M}\mathbf{M^{-1}} = \mathbf{I}_N = \begin{bmatrix} \mathbf{G}\tilde{\mathbf{H}}^T & \mathbf{G}\mathbf{H}^T \\ \tilde{\mathbf{G}}\tilde{\mathbf{H}}^T & \tilde{\mathbf{G}}\mathbf{H}^T \end{bmatrix} \tag{2.30}$$

[3] note that the prefix "hyper" has been introduced by mathematicians keen to impress common mortals with their ability to work in higher dimensions than mere physicists or engineers, but coding engineers have caught up and introduced higher dimensional finite vector spaces to the real world so there is nothing wrong if you prefer to drop the "hyper" and just call it a plane, be my guest...

from where we see that

$$\mathbf{G}\tilde{\mathbf{H}}^T = \mathbf{I}_K \tag{2.31}$$

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}_{K\times(N-K)} \tag{2.32}$$

$$\tilde{\mathbf{G}}\tilde{\mathbf{H}}^T = \mathbf{0}_{(N-K)\times K} \tag{2.33}$$

$$\tilde{\mathbf{G}}\mathbf{H}^T = \mathbf{I}_{N-K} \tag{2.34}$$

where (2.32) shows that **H** is indeed the parity-check matrix. To summarise the steps for going from a general encoder to a corresponding parity-check matrix are

1. complete **G** to an invertible $N \times N$ matrix
2. invert the matrix
3. cut out the last $N - K$ columns of the inverse and transpose to obtain **H**.

The process can be inverted to go from **H** to **G**.

Since a linear code is a vector space, one can talk of its *dimension*, which is defined as in linear algebra as the number of basis vectors it has, or the number of rows K of one of its encoding matrices. A consequence of the process we just described is that the dimensions of a code and its dual code sum to N, the codeword length.

Finally, a special case of the derivation above occurs when we consider a systematic encoder matrix

$$\mathbf{G_s} = \begin{bmatrix} \mathbf{I}_K & \mathbf{P}_{K\times(N-K)} \end{bmatrix}. \tag{2.35}$$

In this case, it is easy to see that the matrix can be completed to an $N \times N$ invertible matrix simply by appending the remaining rows of the identity matrix, i.e.,

$$\mathbf{M} = \begin{bmatrix} \mathbf{I}_K & \mathbf{P}_{K\times(N-K)} \\ \mathbf{0}_{(N-K)\times K} & \mathbf{I}_{N-K} \end{bmatrix}, \tag{2.36}$$

and noting that the inverse $\mathbf{M^{-1}}$ is

$$\mathbf{M^{-1}} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{P}_{K\times(N-K)} \\ \mathbf{0}_{(N-K)\times K} & \mathbf{I}_{N-K} \end{bmatrix} \tag{2.37}$$

which can be easily verified by multiplying the two matrices and verifying that you get the $N \times N$ identity matrix. Consequently, for a systematic matrix, there is no need to operate a matrix inversion explictly to find a parity-check matrix. A parity-check matrix can be read out directly from (2.35) as

$$\mathbf{H_s} = \begin{bmatrix} -\mathbf{P}^T & \mathbf{I}_{N-K} \end{bmatrix}. \tag{2.38}$$

A few remarks are due regarding (2.38):

- a parity-check matrix in this form is called *systematic* although there is no property that follows from this form as there is for the systematic encoder matrix, and the only advantage of this form is that it is easy to find an encoder matrix for it;

- the ease of going betweeen systematic encoder and parity-check matrix suggests an alternative approach for going between general encoder and parity-check matrices: bring an encoder matrix into systematic form by elementary row operations, then read out the systematic parity-check matrix, and conversely to go from parity-check to encoder matrix;

- Warning: some may remember seeing the expression in 3F7 without the minus sign in $-\mathbf{P}^T$. This is because in 3F7 we only looked at binary linear codes and in GF(2), $-x = x$ so there was no need for a minus sign. Luckily, your information data book contains the correct formula *with* the minus sign.

2.1.3 Hamming distance and weight

In this section, we will look at decoding linear codes. Before we do so, there is an interesting property of linear codes that is worth mentioning: *for any of the N positions in an (N, K) q-ary linear code, there is an equal number of codewords that have any of the q symbols in this position.* You can verify this in the example code we studied in the previous sections. This property is easy to prove if we consider any codeword and, say, its j-th position, then take any non-zero element in the j-th column of its encoder matrix and vary the corresponding information symbol through all of its possible q values. By doing this, we vary the value of the j-th position in the codeword through all its values, because it would otherwise violate the invertibility of addition and multiplication in the field. Hence, codewords can be arranged in groups of q codewords with distinct symbols in their j-th position, which proves the property[4]. This means that, when a linear encoder is applied to uniformly distributed data symbols, the probability distribution of any code symbol will also be uniform, and those of you who took 3F7 and learned about information theory will conclude that linear codes, whether viewed as error correction or erasure recovery or probablistically decoded codes, are only ever good for channels with a uniform capacity-achieving distribution such as symmetric channels[5].

In 3F7, a derivation was given for the fact that the optimal decoding rule for the Binary Symmetric Channel (BSC) was to pick the codeword that minimises the *Hamming distance* to the received word. As a reminder, the Hamming distance is simply the number of positions in which two words differ. We will not discuss probabilistic channels in this chapter, but it suffices to say that when error correction and erasure recovery are targeted in any q-ary input q-ary output symmetric channel, the aim is always to minimise the Hamming

[4] technically, we needed to exclude codes with encoder matrices that have an all-zero column for this property to hold, but such columns are just silly and have no practical meaning, although they do play a role in some proofs when averaging over "all" possible codes including silly ones.

[5] making this intuition precise requires a fairly convoluted argumentation that has been the object of many research papers, but the conclusion of those papers is that this intuition is essentially correct.

distance between the received word and the decoded codeword, and the derivation from the Maximum Likelihood (ML) probablistic rule to distance minmisation that was given in 3F7 applies in the same manner to codes over GF(q) as it does for binary codes.

Example: consider again the code with encoder matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}. \tag{2.39}$$

studied in the previous sections. Say we received the word $[2, 2, 1]$. The received word is at Hamming distance 3 from codewords $[0, 0, 0], [1, 0, 2]$ and $[0, 1, 2]$, at Hamming distance 2 from codewords $[2, 1, 0], [1, 2, 0]$ and $[1, 1, 1]$, and at Hamming distance 1 from codewords $[2, 0, 1], [0, 2, 1]$ and $[2, 2, 2]$. Hence the optimal decoding rule should pick any of the last 3 codewords with Hamming distance 1. The fact that two of them require a switch from 0 to 2 and the last one a switch from 2 to 1 is irrelevant because, since the channel is symmetric (or in other words we are only interested in minimising "errors"), we do not weigh these transitions differently from each other.

It is evident that, where Hamming distance is used as the optimisation criterion, an essential property of the code is the minimum Hamming distance d_{min} between any two codewords. The Hamming distance satisfies the triangle inequality, i.e., for any three words $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$,

$$d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c}) \geq d(\mathbf{a}, \mathbf{c}). \tag{2.40}$$

For any received word $\mathbf{r}$ and a codeword $\mathbf{x}$ such that

$$d(\mathbf{x}, \mathbf{r}) < \frac{d_{\text{min}}}{2}, \tag{2.41}$$

the received word will be uniquely decoded to $\mathbf{x}$ because, for any other codeword $\mathbf{x}'$,

$$d(\mathbf{x}', \mathbf{r}) \geq d(\mathbf{x}', \mathbf{x}) - d(\mathbf{r}, \mathbf{x}) \tag{2.42}$$

$$> d_{\text{min}} - \frac{d_{\text{min}}}{2} = \frac{d_{\text{min}}}{2} \tag{2.43}$$

where (2.42) follows from (2.40) and the final step follows from the fact that $\mathbf{x}$ and $\mathbf{x}'$ are at least d_{min} apart and from (2.41). The derivation shows that every codeword comes with a Hamming "sphere" of received words that can be uniquely decoded to it, or, equivalently, that any number of errors strictly smaller than $d_{\text{min}}/2$ will be decoded correctly.

Now consider an erasure channel and consider again a transmitted codeword $\mathbf{x}$. Say now that d_{min} positions of this codeword get erased and that they happen to be precisely the d_{min} positions that are distinct when compared to another codeword $\mathbf{x}'$. Then it will not be possible to uniquely determine the transmitted codeword, since both $\mathbf{x}$ and $\mathbf{x}'$ match all the non-erased positions and their differences are hidden by the erasures. Now consider instead $d_{\min} - 1$ erasures. By the same argument as above, there is at least one non-erased position that will uniquely point to one codeword and not to the other, since all codewords are different in at least $d_{\min}$ positions. To summarise both arguments above, we state the following theorem:

Theorem 2.1 *A code (linear or not) with minimum distance $d_{\min}$ can correct at least $\lfloor \frac{d_{\min}-1}{2} \rfloor$ errors and recover from at least $d_{\min} - 1$ erasures.*

This statement focuses on the minimal decoding ability and emphasises the fact that all our assumptions in the argumentation above (codewords differ in exactly the positions erased, or triangle inequality satisfied with equality) are worst case assumptions. An alternative more common way of stating the same theorem focuses on the number of errors or erasures that are guaranteed to be decoded:

Theorem 2.2 *A code with minimum distance $d_{\min}$ can correct all patterns of t or fewer errors if and only if $d_{\min} > 2t$, and recover from all patterns of $d_{\min} - 1$ or fewer erasures.*

In the final part of this section, we will look at distance properties of linear codes. Let $w(\mathbf{x})$ be the Hamming weight of a codeword $\mathbf{x}$, defined as the Hamming distance between the codeword and the all-zero codeword

$$w(\mathbf{x}) = d(\mathbf{0}_N, \mathbf{x}). \tag{2.44}$$

Note that the all-zero word $\mathbf{0}_N$ is always a codeword of a linear code because for any encoder matrix $\mathbf{G}$,

$$\mathbf{0}_K\mathbf{G} = \mathbf{0}_N. \tag{2.45}$$

Note that the following relation holds for any words $\mathbf{x}$ and $\mathbf{x}'$

$$d(\mathbf{x}, \mathbf{x}') = w(\mathbf{x} - \mathbf{x}'), \tag{2.46}$$

since the difference on the right will be zero in all positions in which $\mathbf{x}$ and $\mathbf{x}'$ agree and only non-zero in positions where they differ. The difference between two codewords in a linear code is always a codeword, since the code is a vector space that satisfies the axiom of closure. This can also be verified by noting that for two codewords $\mathbf{x}$ and $\mathbf{x}'$,

$$(\mathbf{x} - \mathbf{x}')\mathbf{H}^T = \mathbf{x}\mathbf{H}^T - \mathbf{x}'\mathbf{H}^T = 0 - 0 = 0 \tag{2.47}$$

for any parity-check matrix $\mathbf{H}$ of the code. Combining this fact with (2.46), we obtain the following theorem for linear codes:

Theorem 2.3 (Weight-distance equivalence of linear codes) *For any two codewords in a linear code at distance d from each other, there exists a codeword of weight d. Equivalently, if we list the number of codewords at every distance from any particular codeword $\mathbf{x}$, we get the same list no matter which codeword $\mathbf{x}$ we pick. In particular, we also get the same list if we list the number of non-zero codewords of every weight (for example "5 codewords of weight 7, 3 codewords of weight 8, etc."). And finally, for a linear code, the minimum distance is the minimum weight of all non-zero codewords $d_{\min} = w_{\min}$.*

The last statement in particular is very powerful. It allows us to examine every codeword in a linear code in order to determine $d_{\min}$, rather than examining every pair of codewords. Still, determining the minimum distance of an actual code of a good size is not an easy task.

Those of you who took 3F7 have already had a foretaste of how weight relates to the properties of a parity-check and encoder matrix. For a parity-check matrix, the minimum weight is the minimum number of columns in the matrix that are linearly dependent. This can be seen because, say a codeword $\mathbf{x}$ of weight w satisfies $\mathbf{x}\mathbf{H}^T = \mathbf{0}$, writing $\mathbf{h}_j$ for the j-th column of $\mathbf{H}$, this can also be written as

$$x_{m_1}\mathbf{h}_{m_1}^T + x_{m_2}\mathbf{h}_{m_2}^T + \ldots + x_{m_w}\mathbf{h}_{m_w}^T = \mathbf{0}_{N-K} \tag{2.48}$$

where $x_{m_1}, x_{m_2}, \ldots, x_{m_w}$ are the non-zero entries of $\mathbf{x}$. Hence, the minimum weight is the minimum linear combination of columns of $\mathbf{H}$ that yields zero. If you remember your 2P7 Linear Algebra course, the minimum linear combination of columns that yields zero is one more than the maximum number of linearly independent columns, which is also called the rank of the matrix. One of the most surprising results of linear algebra shows that the column rank and the row rank are the same, i.e., the minimum number of linearly independent columns is always the same as the minimum number of linearly independent rows. But since there are only $N-K$ rows in $\mathbf{H}$, the rank can be at most $N-K$ and hence the minimum distance at most one more than that. We have shown the following

Theorem 2.4 (Singleton Bound) *The minimum distance $d_{\min}$ of an (N,K) linear codes satisfies*

$$d_{\min} \leq N - K + 1. \tag{2.49}$$

Example: Consider the single parity-check code over GF(2) with parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{2.50}$$

Its number of rows is 1 and hence $N-K=1$ or $N=K+1$. Therefore, the bound yields

$$d_{\min} \leq N - K + 1 = 2. \tag{2.51}$$

But clearly for this matrix any single "column" is linearly independent on its own, so the minimum distance is indeed 2. This can also be seen by considering that codewords are all the words whose symbols sum to 0 modulo 2, and if any element of a word is 1 you need at least one more to bring it back to zero. Hence, this simple code satisfies the Singleton bound with equality.

Consider now the repetition code over GF(2) with encoder matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{2.52}$$

This code has only two codewords, the all zero and the all 1 codeword and hence its minimum distance is $d_{\min} = N$. Since $K = 1$ in this case, the Singleton bound states that $d_{\min} \leq N - 1 + 1 = N$ so we see that this code satisfies the bound with equality as well.

Now consider a parity check matrix that has as columns all the non-zero binary words of length 3. There are $2^3 - 1 = 7$ columns and hence for this code $N = 7$, $N - K = 3$ and $K = 4$. This is the (7,4) Hamming code that you studied in 2P6 communications. It has minimum distance $d_{\min} = 3$ so in this case, satisfies the strict Singleton bound since $d_{\min} = 3 < N - K + 1 = 4$.

We've seen 3 examples and two of them, fairly trivial binary codes, satisfy the Singleton bound with equality. You may be mislead to think that it is easy to design matrices that satisfy the Singleton bound with equality, i.e., for which any combination of $N-K$ columns is linearly independent. Far from it. Indeed, it can be shown that the two trivial examples we showed are the *only* examples of binary codes that satisfy the Singleton bound with equality. In other words, it is *impossible* to design an $(N - K) \times N$ binary matrix such that any selection of $N - K$ columns is lineary independent, other than the two simple examples of the single parity-check code and the repetition code. Codes that satisfy the Singleton bound with equality are called *maximum distance separable* (MDS) and we will have to operate on higher order alphabets, i.e., GF(q) for $q > 2$ to find examples of codes that satisfy the Singleton bound with equality. These are called Reed-Solomon codes and will be the subject of the next section.

A last remark about the Singleton bound: our exposition above has centered around properties of the parity-check matrix. It is possible to make the same arguments based on the encoder matrix. The easiest way to see this is to consider erasure decoding. The encoder matrix maps the K information symbols to N code symbols. If m of the N code symbols are erased, we can erase the corresponding m columns in the encoder matrix and we are left with a system of equations with K unknowns and $N - m$ equations. We can solve this system if any subset of K out of the surviving $N-m$ columns yields an invertible $K \times K$ matrix. Obviously, for this to be possible, $m \leq N - K$ must hold, because if you erase more than $N - K$ columns there is no way the remaining columns can give you K independent equations since there are already less than K left. Hence the number of erasures that can be decoded, which we know is $d_{\min} - 1$, has to be smaller than $N - K$, which proves the Singleton bound. This also shows us that designing an MDS code is equivalent to requiring an encoder matrix to have the property that any subset of K of its columns has to form an invertible $K \times K$ matrix.

2.1.4 The MacWilliams Identity

We will push the study of Hamming weight of codes and dual codes a bit further and cover the MacWilliams identity, a tool that allows one to obtain the full weight profile of a linear code from the weight profile of its dual code. Although it is unlikely that you will ever want to do this for a practical code, there are several reasons why we decided to include this identity in our short course:

- it's one of the most beautiful and surprising results of coding theory;
- its inventor Jessie MacWilliams was one of the only female researchers among early coding and information theory researchers;
- Jessie MacWilliams was a graduate of Newnham College, so Cambridge-educated coding and information theorist have a historic duty to know about her result;
- MacWilliams' proof is a combinatorial proof and would probably not appeal to most engineering students. The more appealing proof we present here is due to Chang and Wolf, where the second author Jack Wolf was one of the kindest friendliest researchers in the information theory community, a personal friend of your lecturer, and I really wanted to include something of Jack's in my lecture.

We will prove the identities for binary linear codes only but the derivation is easy to extend to non-binary. The proof is based on probabilities but note that the probabilistic experiment that will allow us to derive the identity is only a thought experiment and does not correspond to anything practical we would do with a linear code.

Let A_k for $k = 0, 1, \ldots, N$ be the weight profile of an (N, K) linear code $\mathcal{C}$, i.e., A_k counts the number of codewords of Hamming weight k in the code. For example, the $(7, 4)$ Hamming code has one codeword of weight 0 (the all-zero codeword), 7 codewords of weight 3, 7 codewords of weight 4, and one codeword of weight 7 (the all-ones codeword), hence

$$(A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7) = (1, 0, 0, 7, 7, 0, 0, 1). \tag{2.53}$$

For reasons that will become clear, this is often written as a polynomial $A(x) = A_0 + A_1x + A_2x^2 + \ldots + A_Nx^N$ where there is no physical meaning to the variable x in this context[6], so for the $(7, 4)$ Hamming code the polynomial is $A(x) = 1 + 7x^3 + 7x^4 + x^7$.

In order to derive the identity, consider generating N Bernoulli random variables $X_1, \ldots, X_N$ with Bernoulli parameter p and let $P(E)$ be the probability that the resulting vector is a codeword of our linear code $\mathcal{C}$. The identities will result from our writing this probability in two different ways, once based on the codewords in $\mathcal{C}$ and once based on the dual code $\mathcal{C}^\perp$.

The probability that any specific codeword of weight w is selected in our experiment is $p^w(1-p)^{N-w}$ since the codeword has w ones occurring each with probability p and $N - w$ zeros occuring each with probability $1 - p$ in our experiment. Since the events that the random sequence equals each of the codewords are mutually exclusive, the probability that any codeword will be chosen is simply the sum of the probabilities, and since there are A_w codeword of weight w, the probability is

$$P(E) = \sum_{k=0}^{N} A_k p^k (1-p)^{N-k}. \tag{2.54}$$

[6] Many describe the weight profile using a bi-variate polynomial $A(x, y) = \sum_{k=0}^{N} A_k x^k y^{N-k}$. This has some operational advantages and the MacWilliams identities look prettier when stated in the bi-variate format, but we decided not to follow that path in our derivation as it's confusing enough that we're using polynomials out of context without making it even more confusing by making them bi-variate.

We can further develop this as

$$P(E) = \sum_{k=0}^{N} A_k p^k (1-p)^{-k} (1-p)^N = (1-p)^N \sum_{k=0}^{N} A_k \left(\frac{p}{1-p}\right)^k = (1-p)^N A\left(\frac{p}{1-p}\right) \tag{2.55}$$

where the last expression uses the polynomial notation $A(x)$ with $x = p/(1-p)$.

Now let us express $P(E)$ by considering the dual code. In order for $\mathbf{X} = [X_1, \ldots, X_N]$ to form a codeword, it must be orthogonal to every one of the 2^{N-K} codewords in the dual code $\mathcal{C}^{\perp}$. Let $\mathbf{S} = \mathbf{X}\mathbf{H}^T$ be the "syndrome" that we get by multiplying the random vector $\mathbf{X}$ by the parity-check matrix $\mathbf{H}$ of $\mathcal{C}$. Remember that the codewords of the dual code are obtained by forming linear combinations of the rows of the parity-check matrix,

$$\mathbf{c} = \lambda_1 \mathbf{h_1} + \ldots + \lambda_{N-K} \mathbf{h_{N-K}} \tag{2.56}$$

for all $(\lambda_1, \ldots, \lambda_{N-K}) \in \{0,1\}^{N-K}$. Hence, if $\mathbf{S} = \mathbf{X}\mathbf{H}^T = \mathbf{0}$, then

$$\mathbf{X}\mathbf{c}^T = \sum_{k=1}^{N-K} \lambda_k \mathbf{X}\mathbf{h_k}^T = 0 \tag{2.57}$$

while if $\mathbf{S} \neq \mathbf{0}$, then

$$\mathbf{X}\mathbf{c}^T = \sum_{k=1}^{N-K} \lambda_k \mathbf{X}\mathbf{h_k}^T = \sum_{k=1}^{N-K} \lambda_k s_k \tag{2.58}$$

which, over all $(\lambda_1, \ldots, \lambda_{N-K}) \in \{0,1\}^{N-K}$, will equal 0 for half of the codewords and 1 for the other half[7]. Hence, we have established that the probability $1 - P(E)$ that the syndrome is *not* zero is the sum of the probabilities that each product $\mathbf{X}\mathbf{c}^T$ is non-zero divided by half the number of codewords in the dual code, $2^{N-K}/2$,

$$1 - P(E) = \frac{1}{2^{N-K-1}} \sum_{\mathbf{c} \in \mathcal{C}^{\perp}} P(\mathbf{X}\mathbf{c}^T \neq 0). \tag{2.59}$$

For a codeword of Hamming weight w, the probability that the dot product with $\mathbf{X}$ is non-zero is simply the probability that an odd number of the non-zero positions in $\mathbf{c}$ are ones in X. The probability of obtaining an even number of Bernoulli random variables has been calculated in 3F7 in the context of LDPC codes in Handout 12, Slide 13, and found to be

$$P(\text{Even number out of } k \text{ Bernoulli r.v.}) = \frac{1}{2} + \frac{1}{2}(1-2p)^k \tag{2.60}$$

where the expression in 3F7 could deal with unequal Bernoulli parameters and hence can be simplified to $(1-2p)^k$ here. It is easy to see that the equivalent expression for an odd number of random variables is

$$P(\text{Odd number out of } k \text{ Bernoulli r.v.}) = \frac{1}{2} - \frac{1}{2}(1-2p)^k \tag{2.61}$$

[7] This is a subtle but crucial argument! If you aren't sure, think of how every $s_k \neq 0$ will be present in half the linear combinations $(\lambda_1, \ldots, \lambda_{N-K})$ and skipped in the other half.

and hence we obtain

$$P(\mathbf{X}\mathbf{c}^T \neq 0) = P(\text{odd no. of ones among } w \text{ non-zero elements of } \mathbf{c}) = \frac{1}{2} - \frac{1}{2}(1-2p)^w. \tag{2.62}$$

This leads finally to

$$1 - P(E) = 2^{-(N-K-1)} \sum_{\mathbf{c}\in\mathcal{C}^\perp} \left(\frac{1}{2} - \frac{1}{2}(1-2p)^{w_\mathbf{c}}\right). \tag{2.63}$$

Let $(B_1, \ldots, B_{N-K})$ be the weight enumerator of the dual code and $B(X)$ be the corresponding polynomial. We can re-write (2.63) as

$$1 - P(E) = \frac{1}{2^{N-K-1}} \sum_{k=0}^{N} B_k \left(\frac{1}{2} - \frac{1}{2}(1-2p)^k\right) = 1 - \frac{1}{2^{N-K}} B(1-2p). \tag{2.64}$$

Finally, putting (2.55) and (2.64) together, we obtain the following result

$$(1-p)^N A\left(\frac{p}{1-p}\right) = \frac{1}{2^{N-K}} B(1-2p). \tag{2.65}$$

and, applying a variable transformation $x = \frac{p}{1-p}$ (remember that p had no phyiscal meaning), finally

Theorem 2.5 (MacWilliams identity)

$$A(x) = \frac{(1+x)^N}{2^{N-K}} B\left(\frac{1-x}{1+x}\right) \tag{2.66}$$

Example: Let us apply the identity to find the weight distribution of a $(7,4)$ Hamming code. The parity-check matrix of the code is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.67}$$

from which it is easy to see that all codewords of the dual code except the all-zero codeword have Hamming weight 4, and hence $B(x) = 1 + 7x^4$. Applying the identity, we get

$$\begin{aligned} A(x) &= \frac{(1+x)^7}{2^3}\left(1 + 7\frac{(1-x)^4}{(1+x)^4}\right) && (2.68) \\ &= \frac{(1+x)^3}{8}\left[(1+x)^4 + 7(1-x)^4\right] && (2.69) \\ &= (1+3x+3x^2+x^3)(1-3x+6x^2-3x^3+x^4) && (2.70) \\ &= 1 + 7x^3 + 7x^4 + x^7 && (2.71) \end{aligned}$$

The reader is referred to the paper by Chang and Wolf[8] for a generalisation of the identity to non-binary codes, and to the Wikipedia article[9] on the MacWilliams identity for a statement as bi-variate polynomial.

A practical use for the MacWilliams identity might be for example to investigate properties of LDPC codes. Take a regular $(d_v, d_c) = (3, 6)$ LDPC code for example. Its parity-check matrix has $N - K$ rows of weight 6, which is a fairly low weight compared to the length N of the codewords. Of course the rows of the parity-check matrix are codewords of the dual code, not of the code itself, so the low weight of those codewords may not directly impact upon the performance of the code. However, one may be justified in wondering whether the $N - K$ codewords of very low weight in the dual code have any impact on the weight distribution of an LDPC code. One could set up the identity as

$$A(x) = \frac{(1+x)^N}{2^{N-K}} \left(1 + B_6 \frac{(1-x)^6}{(1+x)^6} + \sum_{k \geq 1, k \neq 6}^{N-K} B_k \frac{(1-x)^k}{(1+x)^k} \right) \tag{2.72}$$

with $B_6 \geq N - K$ then develop the expression to see if any bounds on the coefficients of $A(x)$ can be inferred.

[8] S.C. Chang, Jack K. Wolf, "A simple derivation of the Macwilliams' identity for linear codes", IEEE Trans. on Information Theory, Vol. IT-26, No. 4, July 1980.

[9] `https://en.wikipedia.org/wiki/Enumerator_polynomial#MacWilliams_identity`

2.2 Reed Solomon Codes

Before we embark on a description of Reed Solomon codes, we will start with a motivating example. Consider the following code:

$$[u_1 \dots u_k] \longrightarrow [u_1 \dots u_k 00 \dots 0], \tag{2.73}$$

i.e., the code simply appends $N-K$ zeros to the information words. Take a minute to think about the properties of this code, and in particular think of the answers to the following questions:

- is the code linear?
- is the encoder systematic?
- is it a good code?
- if not, why?

The answers are positive to the first two questions and the encoder matrix has a property that we called "silly" in a footnote earlier in this chapter, which gives a strong hint as to what the answer to the third question should be. More importantly, thinking about such a pathological code gives us a very good intuition about what properties a good code should have: it should introduce strong dependencies between all symbols, so that information lost through missing or wrong symbols can be recovered from its neighbours and its neighbours' neighbours. In the example we gave, the "parity" portion of the codeword does not depend at all on the systematic portion, and the code does not introduce any dependencies between the systematic symbols[10]. In short, a good coding system needs an instrument that makes all code symbols as dependent as possible on each other in addition to adding redundancy. The code above adds redundancy but does not introduce dependency. We will present Reed Solomon coding in a manner that separates these two operations: we will use the "silly" code above to add redundancy, and append a tool whose only function is to introduce dependency. This is a tool most of you are already intimately familiar with, but we will need to re-introduce it in the new context of finite fields: the Discrete Fourier Transform (DFT).

Before we proceed to study the DFT, we should warn that there are several ways to understand Reed Solomon codes and their decoders. Most textbooks use an approach based on roots of polynomials. We will use a different approach often described as the "spectral" view of Reed Solomon coding. Both approaches are very elegant and insightful, but we find that the spectral approach is more suited for teaching engineers because it draws on many mathematical tools that they are very familiar with and "brings it all together" in an exceptionally well suited illustration of everything they've learned (DFT,

[10] the source symbols may arrive with dependencies from the source, but in general we assume that the actual information source would have been compressed before channel coding and hence any dependencies would have been eliminated by compression, since dependency is a form of redundancy.

z-transform, partial fractions, convolution.) The spectral approach was originally described in a textbook by Richard Blahut[11] and used in Jim Massey's lecture notes that were a strong inspiration for the present lecture notes, as mentioned earlier. The approach I will present deviates slightly from Jim's lecture notes in that it makes the didactic choice of introducing frequency-domain encoding, which is an approach to the subject I developed during the year I had the privilege of teaching 4F5 with the help of a "teaching buddy" (his words, not mine) Professor Sir David MacKay while he was fighting his battle against cancer. David passed away before the end of the academic year in which we taught together and I am forever thankful for the experience of teaching with him.

2.2.1 The Discrete Fourier Transform

You have been taught the Discrete Fourier Transform over the complex field $\mathbb{C}$ and defined it using the following equations, for a complex vector $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$[12]:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-i2\pi kn/N} \qquad x_k = \frac{1}{N}\sum_{n=0}^{N-1} X_n e^{i2\pi kn/N}. \tag{2.74}$$

Remaining in $\mathbb{C}$, the first step towards generalising the DFT is to realise that its definition and that of its inverse implement matrix operations. If we define

$$\alpha \overset{\text{def}}{=} e^{-i2\pi/N}, \tag{2.75}$$

then the DFT and its inverse can be written as

$$\mathbf{X} = \mathbf{x}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{N-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^k & \alpha^{2k} & \cdots & \alpha^{k(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{N-1} & \alpha^{2(N-1)} & \cdots & \alpha^{(N-1)^2} \end{bmatrix} \tag{2.76}$$

and

$$\mathbf{x} = \mathbf{X}\frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha^{-1} & \alpha^{-2} & \cdots & \alpha^{-(N-1)} \\ 1 & \alpha^{-2} & \alpha^{-4} & \cdots & \alpha^{-2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{-k} & \alpha^{-2k} & \cdots & \alpha^{-k(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{-(N-1)} & \alpha^{-2(N-1)} & \cdots & \alpha^{-(N-1)^2} \end{bmatrix}. \tag{2.77}$$

[11] Richard E. Blahut, "Theory and Practice of Error Control Codes", Addison-Wesley, 1984

[12] the indexing starting from 0 is convenient for the DFT and we use it here, but note that in general in these notes our indexing starts with 1.

We've chosen to put the minus in the exponents of the inverse transform out of personal preference. We could just as well have defined α without the minus and kept the minus in the definition of the direct transform. We will call the matrices in (2.76) and (2.77) the *DFT matrix* $\mathcal{F}$ and the *inverse DFT matrix* $\mathcal{F}^{-1}$, respectively. If you are wondering whether the two matrices are indeed inverses in the matrix inversion sense, the answer is obviously yes since the combination of the transform and its inverse is the identity, but you are welcome to try and multiply the two matrices to persuade yourself that this is correct. Note as well that we use the coding convention of using row vectors and vector-matrix multiplication, but since the DFT matrix and its inverse are symmetric, they work both ways and you can write the transform as a matrix-vector multiplication using column vectors if you prefer.

In order to generalise the DFT to finite fields, we would like to use the same matrices as those in (2.76) and (2.77), but with an α that "lives" in finite fields. What are the properties of α that make this transform interesting? Two properties spring to mind:

1. $\alpha^N = 1$ while $\alpha^k \neq 1$ for $k = 1, 2, \ldots, N-1$.

2. $\alpha^k = \alpha^{\mathrm{R}_N(k)}$, i.e., larger powers of α reduce modulo N.

These properties should look very familiar to you from when we studied the order of elements in finite fields. The complex field $\mathbb{C}$ is an exceptionally rich field in that there exist elements of any finite order. For any N, $e^{-i2\pi/N}$ and $e^{i2\pi/N}$ both have order N. In contrast, the real field $\mathbb{R}$ only has one element of order 2, $\alpha = -1$, and no elements of any higher order. This is in a nutshell the reason why all Fourier transforms are defined over the complex field, a fact that many of you may have been uncomfortable with ever since we moved from the very contrived real Fourier series in Part IA maths to the much more natural complex Fourier series. The only transforms of interest over $\mathbb{R}$ are the 2-point Fourier transform and the multi-dimensional 2-point Fourier transform called the Walsh-Hadamard transform (whose study lies outside the scope of this course.)

Finite fields are not as rich as the complex field but richer than the real field. Elements exist for some orders but not for any order, as we learned in the previous chapter. As a reminder, elements can only have orders that divide the order of the multiplicative group (Lagrange, 1.14) and Cauchy's theorem states that there will always be elements of all prime possible orders, and since the multiplicative group of $\mathrm{GF}(q)$ is cyclic, there will always be at least one element of maximum order $q-1$.

The DFT in a finite field $\mathrm{GF}(q)$ is defined for lengths N that divide $q-1$ by picking an element of order N α and applying the transforms in (2.76) and (2.77) over $\mathrm{GF}(q)$, with one minor difference: the division in the inverse transform is not by N but by a number $N^\star$ defined as follows:

$$N^\star = \sum_{k=0}^{N-1} 1, \tag{2.78}$$

i.e., the sum of N ones over the Galois field. In a prime field where q is prime, this is simply N (e.g., $q-1$ if $N = q-1$) and so you need to divide by N just like in the complex

DFT. However, in an extension field, the sum of N ones is not necessarily equal to 1. For example in GF(8) if operating with $N = 7$, the sum of 7 ones in GF(8) is 1, so $N^\star = 1$ in this case and there is no division. In summary, the discrete Fourier transform and its inverse over finite fields are defined as

$$X_n = \sum_{k=0}^{N-1} x_k \alpha^{kn} \qquad x_k = \frac{1}{N^\star} \sum_{n=0}^{N-1} X_n \alpha^{-kn}. \tag{2.79}$$

where $N^\star$ is defined in (2.78) and α is an element of order N, or alternatively via the matrix equations (2.76) and (2.77) with N suitably replaced by $N^\star$ in the inverse transform.

All properties of the DFT that you learned (or should have learned) in IB Paper 6 Signal & Data Analysis hold for the finite field DFT. In particular

Cyclic convolution property: if $\mathbf{z} = \mathbf{x} \star_N \mathbf{y}$ where $\star_N$ is the cyclic convolution

$$z_k = \sum_{n=0}^{N-1} x_n y_{\mathrm{R}_N(k-n)}, \tag{2.80}$$

then

$$Z_k = X_k Y_k \text{ for } k = 0, 1, \ldots, N-1, \tag{2.81}$$

i.e., cyclic convolution in the time domain is equivalent to pointwise multiplication in the frequency domain.

Inverse convolution property: the same is true in reverse, i.e., cyclic convolution in the frequency domain is equivalent to pointwise multiplication in the time domain.

Frequency shift property: pointwise multiplication of the time domain sequence by the vector $[\alpha^0, \alpha^{-k}, \alpha^{-2k}, \ldots, \alpha^{-k(N-1)}]/N^\star$ results in a cyclic frequency shift by k in the frequency domain, i.e., if $z_n = x_n \alpha^{-kn}$ for $n = 0, \ldots, N-1$, then $Z_n = X_{\mathrm{R}_N(n+k)}$.

Time shift property: equivalently, multiplying the spectrum pointwise by the vector $[\alpha^0, \alpha^k, \alpha^{2k}, \ldots, \alpha^{k(N-1)}]$ results in a cyclic time shift by k, i.e., if $Z_n = X_n \alpha^{kn}$ for $n = 0, \ldots, N-1$, then $z_n = x_{\mathrm{R}_N(n+k)}$.

These properties are easy to show and the two shift properties follow directly from the convolution property since the inverse transform of the vector $[\alpha^0, \alpha^k, \ldots, \alpha^{k(N-1)}]$ is simply a 1 at position k and 0 elsewhere.

Example: Consider GF(7): the order of the multiplicative group is 6, so there are DFTs of length 2, 3 and 6. We pick $\alpha = 4$ and see that $\alpha^2 = 2$ and $\alpha^3 = 1$ so we have $N = 3$ and define the 3 point tranforms

$$\mathcal{F} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 2 \\ 1 & 2 & 4 \end{bmatrix} \tag{2.82}$$

and

$$\mathcal{F}^{-1} = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 5 & 5 \\ 5 & 3 & 6 \\ 5 & 6 & 3 \end{bmatrix} \tag{2.83}$$

where the last step follows from the fact that $1/3 = 5$ in GF(7). These matrices can be computed in MATLAB using the commands `F = rem(4.^((0:2)'*(0:2)),7)` and `iF = rem(5*2.^((0:2)'*(0:2)),7)` where I used $\alpha^{-1} = 2$ in the inverse transform.

We can now also verify the convolution property, for example take $\mathbf{x} = [1, 2, 3]$ and $\mathbf{y} = [4, 5, 6]$. It is rather tedious to verify by hand, but using the command `z = rem(cconv(x,y,3),7)` in MATLAB I was able to establish that $\mathbf{z} = \mathbf{x} \star_3 \mathbf{y} = [3, 3, 0]$. The DFT of $\mathbf{z}$ is $\mathbf{Z} = [6, 1, 2]$, and it is easy to verify that this is also obtained by pointwise multiplication of the DFTs of $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{X} = [6, 1, 3]$ and $\mathbf{Y} = [1, 1, 3]$, respectively.

We can also verify the frequency shift property by multiplying $\mathbf{x}$ pointwise by $[1, 4, 2]$ for example to yield $\mathbf{z} = [1, 1, 6]$ and verify that its DFT $\mathbf{Z} = [1, 3, 6]$ and see that it indeed $\mathbf{X} = [6, 1, 3]$ shifted cyclically back by 1.

We can repeat the exercise for an extension field, although since it is much harder to use MATLAB to realise operations in extension fields we would have to verify everything by hand. For example, considering GF(16) with multiplication modulo $\pi(X) = 1 + X + X^4$, we can take $\alpha = X^5 = X + X^2$ to obtain again a DFT of length $N = 3$ (for simplicity of exposition) and get a DFT and inverse DFT matrix

$$\mathcal{F} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & X + X^2 & 1 + X + X^2 \\ 1 & 1 + X + X^2 & X + X^2 \end{bmatrix} \tag{2.84}$$

and

$$\mathcal{F}^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 + X + X^2 & X + X^2 \\ 1 & X + X^2 & 1 + X + X^2 \end{bmatrix} \tag{2.85}$$

where note that there was no division by $N^\star$ in the inverse transform since the sum of 3 ones in GF(16) is simply 1.

2.2.2 Recurrence relations, linear complexity and Blahut's theorem

There is one more property of the DFT that we will need and that you have not learned and in fact it is a little known property, which is surprising given how much insight it gives into what the DFT actually does.

Before we introduce this property, we need to prepare by discussing a few familiar topics. The first is *recurrence relations* which are also known as *difference equations*, e.g.,

$$x_k = c_1 x_{k-1} + c_2 x_{k-2} + \ldots + c_L x_{k-L}. \tag{2.86}$$

You have studied those in 1A Maths and in 3F1 if you took 3F1. We are interested in sequences that are generated by recurrence relations, i.e., you pick the first L elements of the sequence and then let the recurrence relation run forever to generate a semi-infinite sequence $\mathbf{x}$. Note that coding theorists often talk about "Linear Feedback Shift Registers" (LFSR) to visualise the hardware that implements a recurrence relation, but you can take this term to be another synonym of recurrence relation in the context of this course. If we consider any sequence $\mathbf{x}$, we will call its *linear complexity* $\mathcal{L}(\mathbf{x})$ the length L of the shortest recurrence relation that generates the sequence. Some sequences have infinite linear complexity, e.g., if we consider $\mathbf{x}$ to be all the digits of π, there is no recurrence relation that can generate that sequence. By convention, the all-zero sequence has linear complexity 0.

Example: The linear complexity of the sequence

$$1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 \ldots \tag{2.87}$$

is 1, since the recurrence relation $x_k = x_{k-1}$ generates it and there is no shorter recurrence relation possible.

The linear complexity of the sequence

$$1, 2, 3, 3, 1, 2, 6, 6, 3, 5, 1, 6, 2, 3, 4, 2, 5, 0, 2, 3, 0, 5, 0, \ldots \tag{2.88}$$

over GF(7) is 3 because it is generated by the recurrence relation

$$x_k = x_{k-1} + 2x_{k-2} + 3x_{k-3} \tag{2.89}$$

primed with the initial values $[x_1, x_2, x_3] = [1, 2, 3]$. It is not trivial to verify that there is no shorter recurrence relation that generates the sequence, but one can do it by setting up a system of two equations with two unknowns c_1 and c_2 to hypothesise a recurrence relation of length 2 that generates x_3 from x_1 and x_2, and x_4 from x_2 and x_3, and in this case we find that the system has no solution.

The final considerations of the example also give us a way forward in determining the recurrence relation when presented with a sequence. Knowing the first $L+1$ symbols of the sequence $\mathbf{x}$, we can set up a first equation tying the L coefficients $c_1, \ldots, c_L$. With every subsequent symbol of $\mathbf{x}$, we get an extra equation and once we've observed $2L$ symbols, we have L equations with L unknowns and the system of equations can be solved in the usual manner using Gauss elimination, QR factorisation or matrix inversion. Solving a system with this particular structure (every row of the matrix is a shift of the preceding row) called a Toeplitz matrix efficiently has received wide attention of mathematicians, signal processing specialists and coding theorists. In mathematics, there are algorithms known as Levinson-Durbin recursions to solve such systems in $O(n^2)$, and in the context of coding theory the algorithm in wide use is called the Berlekamp-Massey algorithm. Interesting as they are, we will not study the actual algorithms here as they are not essential for your understanding of Reed Solomon codes or of their decoders. We do retain the following theorem from the discussion above:

Theorem 2.6 *For a sequence* $\mathbf{x}$ *of linear complexity* $\mathcal{L}(\mathbf{x}) = L$*, observing the any* $2L$ *consecutive elements of* $\mathbf{x}$ *suffices to reconstruct the recurrence relation that generates the sequence.*

Recurrence relations are related to the z-Transform that some of you studied[13] in 3F1, which is defined for a sequence $\mathbf{x}$ as

$$X(z) = \sum_{k=0}^{\infty} x_k z^{-k} \tag{2.90}$$

and has neat properties such as the fact that a time shift by m has transform

$$\mathscr{Z}(x_{i-m}) = z^{-m}X(z) + \sum_{k=1}^{m} z^{-(m-k)}x_{-k}, \tag{2.91}$$

where the sum after the first term depends on the "initial values" of $\mathbf{x}$ which, in this expression (which is in your data book), is taken to mean the values at negative times. A recurrence relation such as the one in (2.86) translates in the z-domain to

$$X(z) = c_1 z^{-1}X(z) + c_2 z^{-2}X(z) + \ldots + c_L z^{-L}X(z) + P(z) \tag{2.92}$$

where the term $P(z)$ summarises the influence of the initial terms from all the time shifts in the recurrence relation. Since $P(z)$ comes from the sums in (2.91) and results in negative powers of z up to $m-1$ where m is the magnitude of the time shift, and since the largest time shift in the recurrence relation is L, we conclude that the largest negative power of z in $P(z)$ is $L-1$. If we define

$$C(z) \stackrel{\text{def}}{=} 1 - c_1 z^{-1} - c_2 z^{-2} - \ldots - c_L z^{-L} \tag{2.93}$$

we obtain the relation

$$C(z)X(z) = P(z) \tag{2.94}$$

or

$$X(z) = \frac{P(z)}{C(z)} \tag{2.95}$$

[13] If you haven't studied the z-transform, never mind, it is not essential for you to understand how Reed-Solomon codes work, although it will be useful in the proof of the theorem below and you'll have to take that theorem without proof.

where $P(z)$ has degree[14] at most $L-1$ and $C(z)$ has degree L. Hence, in z-transform terms, the linear complexity of a sequence $\mathbf{x}$ is the smallest degree of a polynomial $C(z)$ such that you can write $X(z)$ in the form of (2.95) with any polynomial $P(z)$ of smaller degree than $C(z)$.

Finally, before we proceed to the fundamental theorem linking the DFT to recurrence relations, we need to remind ourselves that the tools we just introduced apply to semi-infinite sequences $\mathbf{x}$, whereas with the DFT we live in a "cyclic world" of vectors of length N. We can define the linear complexity of a vector $\mathbf{x}$ simply as the smallest linear complexity of all sequences that begin with the elements of $\mathbf{x}$, irrespective of how they continue. While this is a logical definition, it is sometimes hard to work with it, so we can alternatively work with the linear complexity of the periodic continuation of $\mathbf{x}$, i.e., the sequence $\bar{\mathbf{x}}$ that consists of the vector $\mathbf{x}$ repeated ad infinitum.

We are now ready to introduce what is probably the most surprising theorem of this course:

Theorem 2.7 (Blahut's theorem) *Let $\mathbf{x}$ be a vector of length N and of Hamming weight $w(\mathbf{x}) < N/2$ over any field ($\mathbb{C}, \mathrm{GF}(q)$, etc.), then the Hamming weight of $\mathbf{x}$ equals the linear complexity of its DFT, i.e.,*

$$w(\mathbf{x}) = \mathcal{L}(\mathbf{X}). \tag{2.96}$$

When I discovered this theorem, it generated a firework of insights and I hope you will have the same experience. So this is what Fourier transforms are all about... Linear complexity is a measure of how elaborate the linear dependency of a sequence is, whereas Hamming weight is a measure of its sparsity. The sparser a sequence, the simpler the linear dependencies in its discrete Fourier transform. Due to the symmetric definitions of the DFT and its inverse, it is not surprising that the theorem holds in reverse as well (and you will prove this in the examples paper) and hence the DFT also captures the level of linear dependency of a sequence by returning a transform that is sparse for sequences that have simple dependencies, and dense for sequences that have more complex dependencies. You may also be a touch of disappointed that the mighty Fourier transform that seems so powerful in enabling spectral analysis of signals in fact captures only linear dependencies, while non-linear dependencies are lost. This is not so surprising given that it is a linear transform. Finally, you may have woken up to the importance of the DFT when dealing with error correction, given that Hamming weight has now popped up in a theorem in conjunction with the DFT. If we are out to correct up to t errors, then we know that the Hamming weight of the error sequence is at most t and hence the linear complexity of its

[14] We extend the notion of "degree" to negative powers here, because of the strange decision by control theorists to define their beloved z-transform in terms of z^{-1}. Indeed, coding theorists dislike this approach and tend to introduce what they call a D-transform which is identical to the z tranform in its definition and in all its properties except that there is no minus in the exponent of D, with the benefit that you can then speak about "degrees" of polynomials in the usual way without cringing in fear at what happens when the degrees are negative. We decided not to introduce yet another transform in order not to confuse you, but rest assured that all degree calculations are fine as you can easily picture by re-defining $D = z^{-1}$ or by multiplying quotients of polynomials by z^m where m is the largest degree.

DFT is at most t. We will use this fact in the next section in the design and decoding of Reed-Solomon codes. We now prove the theorem.

Proof: we prove the theorem for the linear complexity of the periodic repetition $\bar{\mathbf{X}}$. In the examples paper, you will have the opportunity to reflect on whether the result still holds for the stricter definition of the linear complexity of the vector $\mathbf{X}$. We have

$$\begin{aligned}\sum_{k=0}^{\infty}\bar{X}_k z^{-k} &= \sum_{k=0}^{\infty} X_{\mathrm{R}_N(k)} z^{-k} && (2.97)\\ &= \sum_{k=0}^{\infty}\sum_{n=0}^{N-1} x_n \alpha^{n\,\mathrm{R}_N(k)} z^{-k} = \sum_{k=0}^{\infty}\sum_{n=0}^{N-1} x_n \alpha^{kn} z^{-k} && (2.98)\\ &= \sum_{n=0}^{N-1} x_n \sum_{k=0}^{\infty}(\alpha^n z^{-1})^k && (2.99)\\ &= \sum_{n=0}^{N-1}\frac{x_n}{1-\alpha^n z^{-1}} && (2.100)\end{aligned}$$

where we used the expression for the sum of geometric sequences from your maths data book in the last step. The final expression (2.100) is a proper partial fraction expansion of a rational function with $w(\mathbf{x})$ distinct roots, and hence can be re-written as quotient of polyomials $P(z)/C(z)$ where $P(z)$ has degree at most $w(\mathbf{x})-1$ and $C(z)$ has degree $w(\mathbf{x})$.

□

2.2.3 Reed Solomon coding

A t error correcting Reed-Solomon code is the set of words in $\mathrm{GF}(q)^N$ whose discrete Fourier transform (DFT) is zero in its first $2t$ positions, where N must be a length for which the DFT exists in $\mathrm{GF}(q)$. When a codeword $\mathbf{c}$ is transmitted over the channel, an error vector $\mathbf{e}$ with up to t errors is added to it to yield a received vector $\mathbf{r}$, i.e.,

$$\mathbf{r} = \mathbf{c} + \mathbf{e}, \text{ where } w(\mathbf{e}) \leq t. \tag{2.101}$$

Since the DFT is a linear operation, if we take the DFT of $\mathbf{r}$, we obtain the same expression in the frequency domain

$$\mathbf{R} = \mathbf{C} + \mathbf{E}. \tag{2.102}$$

Since $\mathbf{C}$ is zero in it first $2t$ positions, the first $2t$ positions of $\mathbf{R}$ belong to the DFT of the error vector $\mathbf{E}$. By Blahut's theorem, the linear complexity of the error vector is less than t

$$\mathcal{L}(\mathbf{E}) \leq t \tag{2.103}$$

and hence observing $2t$ consecutive symbols of $\mathbf{E}$ allows us to reconstruct all of $\mathbf{E}$ and hence to recover $\mathbf{C}$ from $\mathbf{R}$. That's all there is to know about Reed Solomon codes really, and if

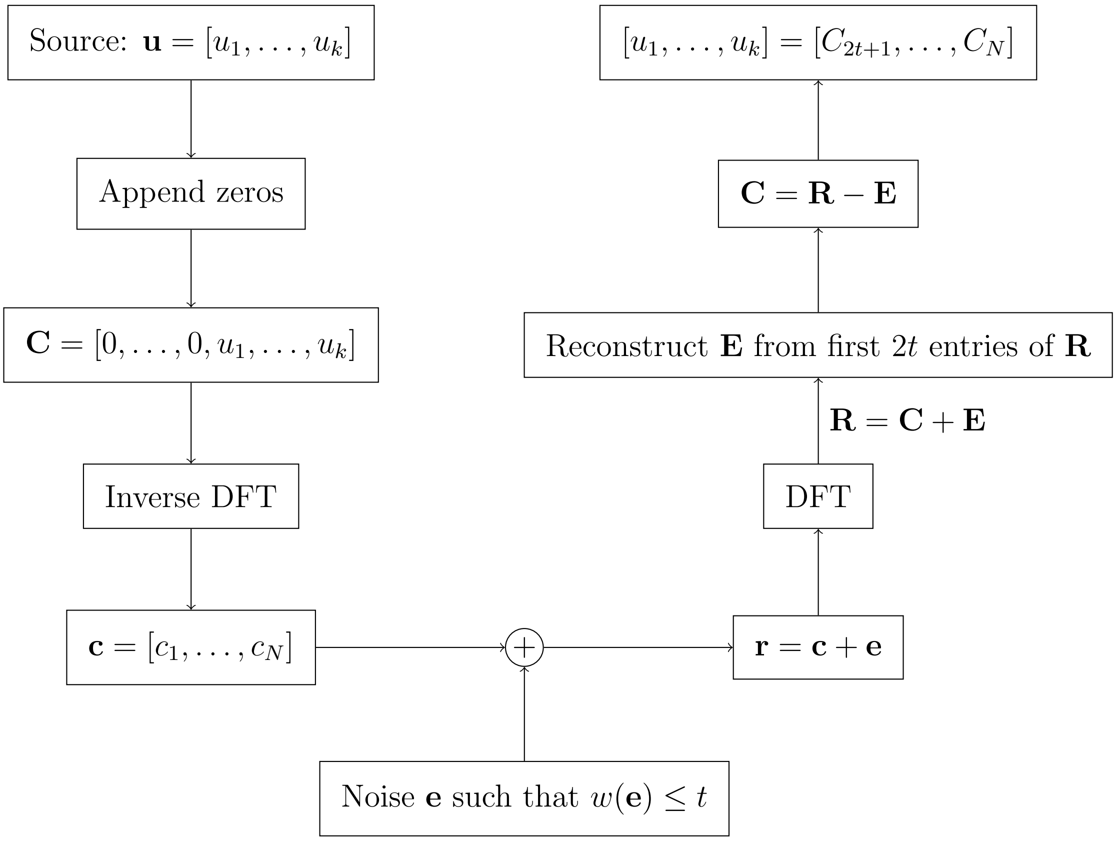
Source: $\mathbf{u} = [u_1, \dots, u_k]$
Append zeros
$\mathbf{C} = [0, \dots, 0, u_1, \dots, u_k]$
Inverse DFT
$\mathbf{c} = [c_1, \dots, c_N]$
+
Noise $\mathbf{e}$ such that $w(\mathbf{e}) \leq t$
$\mathbf{r} = \mathbf{c} + \mathbf{e}$
DFT
$\mathbf{R} = \mathbf{C} + \mathbf{E}$
Reconstruct $\mathbf{E}$ from first $2t$ entries of $\mathbf{R}$
$\mathbf{C} = \mathbf{R} - \mathbf{E}$
$[u_1, \dots, u_k] = [C_{2t+1}, \dots, C_N]$

Figure 2.1: Reed Solomon frequency-domain encoding and corresponding decoding

you understand what I've explained so far you can stop reading and derive everything else in this section by yourself, as it all follows from the argumentation presented so far.

The simplest practical encoding and decoding scheme for a Reed Solomon code is obtained by combining the "silly" encoder we proposed at the beginning of this chapter with the DFT. This is illustrated in Figure 2.1. The codeword in the frequency domain is formed by appending the k information symbols to a prefix of $2t = N - K$ zeros, then the inverse DFT is taken to form the codeword in the time domain. This is transmitted over a channel that will scramble at most t symbols of the codeword. This "scrambling" process can be interpreted as an addition of an error vector $\mathbf{e}$ to the codeword (even if the actual channel is not physically performing additions). The receiver takes the DFT of the received sequence and works out the recurrence relation that will generate the whole error sequence from its first $2t$ values, which are in clear in the DFT of the received sequence since the codeword is zero in the first $2t$ positions. Finally, once the error sequence is reconstructed, it can be subtracted from the received sequence to obtain the codeword in the frequency domain, from which the information symbols can be read out as the last K symbols.

Although the procedure in Figure 2.1 speaks for itself and does not require any further description, since Reed Solomon codes are linear codes, you may still be curious to find out what is the encoder matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$ that results from the description given. Remember that the parity-check matrix $\mathbf{H}$ can be used to verify that a word $\mathbf{x}$ is a codeword if it fulfils the equation $\mathbf{x}\mathbf{H}^T = \mathbf{0}$. For Reed-Solomon codes, all we need to verify is that the first $N - K = 2t$ positions of the DFT of the word are zero. Hence, the parity-check matrix $\mathbf{H}$ consists simply of the first $2t = N - K$ rows of the DFT matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & \alpha^{4t} & \cdots & \alpha^{2t(N-1)} \end{bmatrix}. \tag{2.104}$$

On the other hand, the encoder we used prefixes $2t$ zeros to the information word and then takes the inverse DFT to obtain the codeword. The inverse DFT is a matrix multiplication, but the first $2t$ rows of this matrix are not used in this operation since the first $2t$ elements of the codeword in the frequency domain are zero. Hence, the encoder matrix $\mathbf{G}$ consists simply of the last K rows of the inverse DFT matrix

$$\mathbf{G} = \frac{1}{N^\star} \begin{bmatrix} 1 & \alpha^{-(2t+1)} & \alpha^{-2(2t+1)} & \cdots & \alpha^{-(2t+1)(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{-(N-1)} & \alpha^{-2(N-1)} & \cdots & \alpha^{-(N-1)^2} \end{bmatrix}. \tag{2.105}$$

The scheme we presented can be viewed as being "systematic in the frequency domain". This is nice to know, but there is not much benefit in being systematic in the frequency domain as one needs to take the DFT to benefit from this systematicity, which is a fairly complex operation. If you do want to take advantage of the benefits of a systematic code, there is of course nothing preventing you from encoding Reed-Solomon codes using a systematic encoder, which is obtained by performing elementary operations on $\mathbf{G}$ and bringing it into systematic form, or, alternatively, by performing elementary operations on $\mathbf{H}$ to bring it into systematic form and then reading out $\mathbf{G_s}$ from $\mathbf{H_s}$ using the relations (2.35) and (2.38). The encoding is then performed simply by multiplication of the information word $\mathbf{u}$ by $\mathbf{G_s}$,

$$\mathbf{c} = \mathbf{u}\mathbf{G_s}. \tag{2.106}$$

For the decoding, one needs to take the DFT, perform the same operations as in Figure 2.1 up to obtaining $\mathbf{C}$ by subtracting the reconstructed DFT of the error, but then we need to take one more inverse DFT to recover $\mathbf{c}$ and read out the systematic part. Hence, this coding approach requires a matrix multiplication (and no inverse DFT) in the transmitter, and both a DFT and an inverse DFT in the receiver, while the system we presented requires only an inverse DFT in the transmitter and a DFT in the receiver. Note that in all practical applications that I am aware of (compact disks, QR codes, etc.), systematic encoders are used, so the approach we presented in Figure 2.1 is not widely known or adopted. Typical implementations of Reed-Solomon codes are over GF($2^8 = 256$) with length $N = 255$, to corect up to $t = 6$ errors, i.e., $N - K = 2t = 12$ and $K = 243$.

In the examples paper, you will have ample opportunity to test your skills at both frequency domain encoding and systematic encoding and decoding of Reed Solomon codes. I suggest that you do the first example "crib in hand" and, once you feel confident, do the remaining examples on your own to make sure you get all the elements of the coding system.

Before concluding, it is worth noting that a Reed-Solomon code can correct t errors where $2t = N - K$. Hence, its minimum distance must be at least $2t + 1$. The Singleton bound states that

$$d_{\min} \leq N - K + 1 = 2t + 1, \tag{2.107}$$

implying that the Reed-Solomon code is Maximum Distance Separable (MDS) and fulfils the Singleton bound with equality. We have finally constructed our $K \times N$ encoding matrix for which every subset of K columns is linearly independent, and it turned out to be simply a subset of the rows of the discrete Fourier transform matrix! Note that we chose to place our zeros in the DFT of the codeword in the first $2t$ positions out of convenience: $\mathbf{H}$ comes out as the first $N - K$ rows of the DFT matrix and $\mathbf{G}$ comes out as the last K rows of the inverse DFT matrix. This choice is not compulsory and any consecutive $2t$ zeros would have done, with appropriately shifted windows into the DFT and inverse DFT matrix for $\mathbf{H}$ and $\mathbf{G}$, respectively. This also shows why a Reed-Solomon code can recover from any pattern of $N - K = 2t$ erasures. The procedure described in Figure 2.1 can be adapted to correct erasures so there is no need to invert a $K \times K$ matrix to recover from erasures. This is described in principle but without details in Blahut's book, and the technique has been fully developed, refined an implemented by Talay Cheema in 2017 in the course of a 4th year project here at the Department of Engineering. Talay's implementation was for GF(2^m) where m is in the order of magnitude of 2-300 ("packets" of 2-300 bits) and he definitely holds the world record for the largest Reed-Solomon encoder/decoder ever built.

2.3 Problems for Chapter 2

Problems marked as $\star$ are typical exam questions. Problems marked 😎 are borrowed with thanks from Jim Massey's 1990s lectures at ETH Zurich.

Problem 2.3.1: Single error detecting code (😎)

Let $\mathbf{V}$ be the set of all vectors $\mathbf{b} = [b_1, b_2, \ldots, b_N]$ in GF$(q)^N$ such that $b_1 + b_2 + \cdots + b_N = 0$, where $n \geq 2$.

(a) Show that $\mathbf{V}$ is a *proper* subspace of GF$(q)^N$, i.e., a subspace not equal to the parent vector space GF$(q)^N$. Note that it follows that $\dim(\mathbf{V}) \leq N - 1$.

(b) Show that $\mathbf{g}_1 = [1, 0, \ldots, 0, -1], \mathbf{g}_2 = [0, 1, \ldots, 0, -1], \ldots, \mathbf{g}_{N-1} = [0, 0, \ldots, 1, -1]$ are a basis of $\mathbf{V}$ so that $\dim(\mathbf{V}) = N - 1$. Thus, $\mathbf{V}$ is an $(N, K = N - 1)$ q-ary linear code.

(c) Show that the code $\mathbf{V}$ has minimum distance $d_{\min} = 2$.
Hint: Show first that there are codewords $\mathbf{b}$ and $\mathbf{b}'$ with $d(\mathbf{b}, \mathbf{b}') = 2$. Then show that for any distinct codewords $\mathbf{b}$ and $\mathbf{b}'$, $d(\mathbf{b}, \mathbf{b}') \geq 2$.

(d) Write the explicit equation for the codeword digits b_i, $1 \leq i \leq N$, in terms of the information digits a_i, $1 \leq i \leq N-1$, when the linear encoder G determined by the basis in b) is used.

Note: You have shown that $\mathbf{V}$ is a *single-error-detecting code.* For any $N \geq 2$, this code is an *optimum* single-error-detecting code in the sense that it has the maximum number of codewords, q^{N-1}, of any (N, K) linear code with $d_{\min} \geq 2$.

Problem 2.3.2: Binary equidistant linear codes (😎)

Consider the sequence of binary codes for $m = 2, 3, 4, \ldots$ whose systematic encoding matrices are defined recursively by

$$G_2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad G_m = \begin{bmatrix} 1 & 0 \cdots 0 & 1 \cdots 1 \\ 0 & & \\ \vdots & G_{m-1} & G_{m-1} \\ 0 & & \end{bmatrix}.$$

(a) Write out all the codewords in the code generated by G_2. What is $d_{\min}$?

(b) Write out all the codewords in the code generated by G_3. What is $d_{\min}$?

(c) Show, by induction on m, that all the non-zero codewords in the code corresponding to G_m have Hamming weight exactly 2^{m-1}. *Hint:* Let $[a_1 a_2 \ldots a_m]$ be the information vector and consider separately the cases where $a_1 = 0$ and $a_1 = 1$.

(d) *Note:* A q-ary (N, K) linear code in which all non-zero codewords have the same Hamming weight is called an *equidistant code.* (Why is this terminology appropriate?)

Problem 2.3.3: Binary Hamming codes (😎)

Let H_m, where $m \geq 2$, be the $m \times (2^m - 1)$ matrix whose columns $\mathbf{c}_1^T, \mathbf{c}_2^T, \ldots, \mathbf{c}_{2^m-1}^T$ are such that $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2^m-1}$ are all the distinct non-zero vectors in $\mathrm{GF}(2)^m$.

(a) Explicitly give appropriate choices for the matrices H_2 and H_3.

(b) Show that no pair of columns of H_m are linearly dependent. (*Hint:* Remember that the only scalars are 0 and 1.) Show that there are triplets of columns of H_m that are linearly dependent.

(c) Show that the rows of H_m are linearly independent.
Hint: Consider m special columns of the matrix H_m.

(d) You have now proved that H_m is a reduced parity-check matrix of a binary (N, K) linear code with $d_{\min} = 3$. What are the parameters N and K of this code?

(e) In terms of $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2^m-1}$, find the syndrome $\mathbf{s}$ relative to H_m for the error pattern $\mathbf{e}$ of weight one whose "error" is in position i. Find $\mathbf{s}$ for the error pattern $\mathbf{e} = \mathbf{0}$. Explain now how to implement a single-error-correcting syndrome decoder for this binary linear code.

Note: The codes of this problem are called the *binary Hamming codes*, in honor of R. W. Hamming, who discovered them around 1948.

Problem 2.3.4: Non-binary Hamming codes (😎)

Let H_m, where $m \geq 2$, be the $m \times \frac{q^m-1}{q-1}$ matrix whose columns $\mathbf{c}_1^T, \mathbf{c}_2^T, \ldots, \mathbf{c}_L^T$ ($L = \frac{q^m-1}{q-1}$) are such that $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_L$ are all the distinct non-zero vectors in $\mathrm{GF}(q)^m$ whose *first non-zero component is a 1.*

(a) Explicitly give appropriate choices of the matrices H_2 and H_3 for $q = 3$.

(b) Show that no pair of columns of H_m are linearly dependent. Show that there are triplets of columns of H_m that are linearly dependent.

(c) Show that the rows of H_m are linearly independent.
Hint: Consider m special columns of the matrix H_m.

(d) You have now proved that H_m is a reduced parity-check matrix of a q-ary (N, K) linear code with $d_{\min} = 3$. What are the parameters N and K of this code?

Note: The codes of this problem, when $q \neq 2$, are called the *non-binary Hamming codes* although they were actually first described by M. J. E. Golay in 1949. Golay, who received his diploma in electrical engineering from the ETH Zürich in 1924, was an active and prolific researcher until his death in April 1989.

Problem 2.3.5: Reed Solomon Codes ($\star$)

The following questions invite you to play with the concept of Reed Solomon Codes from the simplest examples to longer codes.

(a) Construct a Reed Solomon Code over GF(5).

(i) Find an element α of multiplicative order 4.

(ii) Construct the full DFT matrix of length 4

$$\mathbf{F} = \begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 \\ \alpha^0 & \alpha^2 & \alpha^0 & \alpha^2 \\ \alpha^0 & \alpha^3 & \alpha^2 & \alpha^1 \end{bmatrix},$$

then retain its first two rows to serve as the parity-check matrix of your Reed Solomon code.

(iii) What is the code dimension? How does it relate to the dimensions of the parity-check matrix? Compute a systematic encoder matrix for the code.

(iv) How many errors can this code correct?

(v) Construct the full inverse DFT matrix

$$\mathbf{F}^{-1} = \frac{1}{4}\begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^0 & \alpha^3 & \alpha^2 & \alpha^1 \\ \alpha^0 & \alpha^2 & \alpha^0 & \alpha^2 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 \end{bmatrix}$$

where you musn't forget that division by 4 in front of the matrix is to be evaluated over GF(5).

(vi) The received word at the output of a channel is $(0, 0, 2, 2)$. Multiply the received word by the DFT matrix.

(vii) Construct the recurrence relation / linear feedback shift register (LFSR) of length 1 that generates the syndrome and reconstruct the remaining 2 symbols of the DFT of the error sequence by computing the next 2 symbols out of the LFSR.

(viii) First assume that the encoder was a frequency domain encoder, where the information symbols are simply the non-zero symbols of the DFT of each codeword. What are the information symbols in this case?

(ix) Now assume that a systematic encoder was used, where the information symbols are the first $K = 2$ symbols of the codeword. You now need to recover the codeword in the time domain. You can do this in two ways:

1) subtract the error sequence from the received sequence in the frequency domain to obtain the codeword in the frequency domain. Now take the inverse DFT to recover the codeword in the time domain and hence the information symbols in its systematic part.
2) take the inverse DFT of the frequency domain error sequence you recovered via the LFSR construction, then substract it from the received sequence to obtain the codeword. This approach has the added benefit that you can immediately verify that the error sequence has the assumed Hamming weight.

(b) Construct a 1-dimensional Reed-Solomon Code of length 3 over GF(4). Repeat the steps above. Note that every entry in the DFT, inverse DFT and parity-check matrix is either a binary polynomial of degree 2 or less, or a binary string of length 2. Note also that the term in front of the inverse DFT matrix is $\frac{1}{3 \text{ modulo } 2}$ which in this case is simply 1. The received word is $(0, 0, X)$.

(c) Construct a 3-dimensional Reed-Solomon code of length 5 over GF(16). The received word is $(X^3 + X^2, 1, 1, X, X + 1)$.

(d) Construct a 2-dimensional Reed-Solomon code of length 6 over GF(7). The received word is (1,3,6,3,2,3).

(e) Construct a 2-dimensional Reed-Solomon code of length 4 over GF(9) using the primitive polynomial $\pi(X) = 1 + X + 2X^2$ to define multiplication. The received word is $(1, 2X, 0, 2X)$.

*N*ote: there are a lot of examples to work through in this problem and you are not necessarily expected to solve them all. Stop if and when you get bored.

Problem 2.3.6: Partial Fractions and Blahut's Theorem

Show that a rational function $f(.)$ over any field $\mathcal{F}$ given as a partial fraction expansion of the form

$$f(z) = \frac{c_1}{1 - \alpha_1 z^{-1}} + \frac{c_2}{1 - \alpha_2 z^{-1}} + \ldots + \frac{c_k}{1 - \alpha_k z^{-1}},$$

where all $\alpha_i, i = 1, \ldots, k$ are non-zero and distinct, and $c_i \neq 0$ for $i = 1, \ldots, k$, then any rational representation of $f(.)$

$$f(z) = \frac{C(z)}{D(z)},$$

has a denominator polynomial $D(z)$ of degree at least k.

Problem 2.3.7: Reverse Blahut Theorem

Prove that the Hamming weight of the discrete Fourier transform of a vector is equal to the linear complexity of its periodic repetition.

Hint: this is easy and follows the same outline as the proof of the direct theorem.

Problem 2.3.8: Finite Length vs. Periodic Repetition in Blahut's Theorem

Our statement of Blahut's theorem is for finite sequences of weight $w < d/2$. Our proof is incomplete in that it shows that the linear complexity of the periodic repetition of $\mathbf{S}$ is w irrespective of whether $w < d/2$, but says nothing about the linear complexity of the finite sequence $\mathbf{S}$. Complete the proof.

Chapter 3

Introduction to Cryptology

This chapter is an aside in our course: cryptology is a vast discipline with a very active research community at the intersection between mathematics, computer science and engineering. It would be unreasonable to aim to teach you this topic in depth within the last 2 hours of our course. On the other hand, there are many intersections between information theory, coding and crypyology: some areas of cryptology rely on information theoretic arguments; some techniques from coding theory can be used as primitives in cryptographic or cryptanalytic protocols; many cryptographic algorithms rely on the same mathematical fundamentals that we've had to learn in the first chapter of this course in order to tackle algebraic coding. In view of these intersections, this last short chapter on cryptology serves the following purpose:

- give you a bird's eye view of cryptology as a field so you get a rough idea of what it is and what its sub-disciplines signify;
- focus on aspects of cryptology that intersect with information theory
- focus briefly on one method that intersects with what we learned about Reed Solomon coding;
- introduce two well established public key cryptosystems that rely on the number theory we've learned at the beginning of this course.

By the end of this chapter, you should be equipped with sufficient knowledge of cryptography to inspire you to read up and study more about the topic. You may have sufficient knowledge to start implementing some of the techniques discussed, but would need to read up and complete your patchy knowledge in order to implement them fully (e.g. we will not discuss methods to generate large random primes that are necessary for a secure implementation of the public key cryptosystems we introduce.)

Having just studied algebraic coding for communications, the following dictionary may be a useful help in getting to grips with cryptologic terminology:

Communications	Cryptology
code	cipher
encode	encipher
decode	decipher
information	plaintext
codeword	ciphertext

Many in public debate would use the word “code” and “coding” to signify the transmission of secret messages, but this is not the norm among cryptologists.

Finally, it is worth pointing out that cryptologists, tired of using abstract terminology to refer to the origin and destination of messages (e.g. “from A to B”), have developed a colourful cast of characters to discuss their protocols:

Alice: the origin of a secret message, i.e., the person who wants to transmit a secret message.

Bob: the destination of a secret message, i.e., the intended recipient of the message.

Eve: the enemy *cryptanalyst* (definition to follow) who attempts to intercept a secret message.

3.1 Classifications of Cryptology

In this section, we give an overview of the various areas encompassed under the field of cryptology. The field can be divided into two areas of interest:

Cryptography: from the Greek κρυπτος, kryptos, meaning “hidden” and γραφειν, graphein, meaning “to write” is the area concerned with ensuring secrecy;

Cryptanalysis: is the area concerned with breaking secrecy, with finding out secrets that have been hidden using cryptographic methods.

Traditionally, cryptographers would have been thought as the “good guys” working for government agencies while cryptanalysts would have been the “bad guys”, hackers trying to steal rightfully hidden secrets. Nowadays the situation is often reversed, with gangsters and terrorists using cryptography to hide their communication and government agencies using cryptanalysis to uncover their secrets.

The possible attacks pursued by cryptanalysts are classified as follows:

Ciphertext only attack: the cryptanalyst only has access to the ciphertext and bases an attack on knowledge of the ciphertext only;

Known plaintext attack: the cryptanalyst knows a plaintext/ciphertext pair, e.g., such as the “cribs” used in the breaking of the enigma naval code used by the Germans in

World War II, where the British crytanalists had guessed that some encrypted messages were being headed with stereotypical messages or contained weather reports, in some cases the same weather reports that were being sent unencrypted to lesser naval units.

Chosen plaintext attack: the cryptanalyst can cause a chosen plaintext to be encrypted. This is often the case in modern attacks when the enemy cryptanalyst can use impersonation to cause a chosen plaintext to be encrypted.

Chosen ciphertext attack: this is similar to the chosen plaintext attack, except that the cryptanalysts can cause chosen encrypted messages to be decrypted and bases their attack on these pairs.

While it is well known to most that cryptography aims to keep secrets hidden, secrecy is not the only aim of cryptographic protocols. Indeed, cryptography can be depicted as having the following dual aims:

Secrecy: how to write a message so no-one except its intended recipient can read it.

Authenticity: how to write a message so anyone reading it is confident about who authored the message.

The aims are dual in the sense that it is possible to devise cryptographic protocols that ensure one without the other, and in cases where you need both secrecy and authenticity you will often need to implement two parallel methods to achieve each aim independently of the other.

A further classification of cryptographic methods focuses on the requirement for a shared secret key between origin and destination:

Secret key cryptosystems: rely on the existence of a secret key known only to origin and destination as a basis for ensuring the secrecy and authenticity of communications.

Public key cryptosystems: do not assume the existence of such a secret key, and aim instead to establish a secret channel in full view of the enemy cryptanalyst using a public channel.

The possibility of public key cryptosystems was a huge surprise to most when they were first proposed in the 1970s. The idea that we could generate secrecy “out of thin air” under full public scrutiny seemed almost like magic. Note however that most public key protocols in fact rely on the availability of authenticity to construct secrecy. It is not possible to generate secrecy in full public view if you are not able to ascertain the authorship of messages you receive.

The last classification that we will cover concerns the question whether the security of a cryptosystem can be proved mathematically:

Unconditional security: concerns itself with the study of methods and conditions for which there are mathematical proofs of the safety of a cryptographic protocol, so that even if the opponent has unlimited computing resources it is impossible for them to detect our secret communications.

Computational security: concerns itself with algorithms that ensure security (secrecy and authenticity) so that an opponent with limited computational resources using all currently known and published mathematical know-how would be unable to detect our secret communications.

Unconditional security was long not the focus of active research, with the only reference being Claude Shannon's 1949 paper "Communication Theory of Secrecy Systems", written to apply the principles of information theory that he had developed in his famous 1948 paper "A Mathematical Theory of Communications" to the problem of secret communications. This has changed in recent years and there is a very active community of researchers, mainly information theorists, who are studying contexts in which provable security is achievable, under the label "physical-layer security". Computational security on the other hand is the mainstream in cryptological applications and research. The main principle underlying the field is that, if anyone were clever enough to solve one of the seminal mathematical problems that secure protocols rely on, then they'd be much more likely to seek fame by publishing their result than to sit in a dark corner and use their solution to steal credit card numbers off the internet. Mathematicians can be fairly exotic and I for one can well imagine a mathematician who, having proved that $p = np$, would prefer to sit quietly on their seminal result and not to publish it.

We conclude this section with a statement of Auguste Kerchkhoffs'[1] principle:

Principle 3.1 (Kerchkhoff's principle) *The cipher should be designed so as to be secure when the enemy cryptanalyst knows all details of the enciphering process and deciphering process except for the secret key.*

In Kerchkhoffs' day, cryptology was not just a theory about secrecy but also a secret theory: nobody in their right mind would publish any details of devices and methods to ensure secrecy. Cryptographers and cryptanalysts worked for shady government agencies and were sworn to secrecy about their work. Kerchkhoffs' principle was visionary in the sense that it signified the birth of cryptology as a scientific discipline with the same level of public scrutiny and peer reviewing as is common within all scientific disciplines. Nowadays, no one would trust a cryptologist who claimed to have invented a strong cipher if they chose to keep the details of their cipher secret. A last hiccup of the old age of cryptology was the launch of the GSM "Global System for Mobile Communications" 2G standard in the 1990s, where the standardisation body had decided to keep the security features of the new standard confidential. Within months of deployment, cryptanalysts had found weaknesses in the security and published software that could be used to crack GSM security. Lessons were learned and subsequent wireless communications standards relied on published algorithms that had received sufficient public scrutiny to make them unlikely to be broken so easily.

[1] `https://en.wikipedia.org/wiki/Auguste_Kerckhoffs`

3.2 Information theoretic perfect secrecy à la Shannon

This section will re-visit what most of you learned in 3F7 Examples Paper 2, Question 10.

- communication setup
- Shannon's definition of perfect secrecy
- Shannon's theorem
- Vernam's cipher

3.3 Secret key cryptography

This Section will be taught using a few examples on the black board and there are currently no lecture notes for it.

3.3.1 Stream ciphers

- general definition and uses
- example
- linear complexity and link to Reed Solomon Codes

3.3.2 Block Ciphers

- general definition and uses
- Shannon's confusion and diffusion
- substitution vs transposition
- cascade structure and Maurer's theorem
- DES
- list of modern ciphers: AES, RC5, IDEA

3.4 Public Key Cryptography

Public-key cryptography relies on the concept of a *one-way function* or a *trapdoor one-way function.* The former is a mathematical function that is easy to compute in one direction, but whose inverse is known to be hard to compute, i.e., there is no known algorithm to compute is inverse in reasonable time. The latter is a function that, like a standard one-way function, has a hard to compute inverse if one doesn't know a secret "trapdoor", but can be computed easily if one knows the "trapdoor" (the trapdoor is typically a secret number.) We will see two examples of well known and established cryptosystems that are based on either a one-way function (the Diffie-Hellman protocol) or a trapdoor one-way function (the Rivest-Shamir-Adelmann protocol).

3.4.1 The Diffie-Hellman key distribution system

The Diffie-Hellman key distibution system operates in a cyclic group. It relies on the difficulty of inverting discrete exponentiation α^x for a given generator α and an unknown x for groups whose order has a large prime factor, e.g. the multiplicative group of GF(p) where p is prime and $p-1$ has a large prime divisor. The inverse of discrete exponentiation is called the *discrete logarithm* and satisfies all the usual properties of logarithms, i.e.,

$$\log_\alpha(n_1 \odot n_2) = \log_\alpha n_1 \oplus \log_\alpha n_2 \text{ and } \log_\alpha(n_1^{n_2}) = n_2 \odot \log_\alpha n_1 \tag{3.1}$$

where $\oplus$ and $\odot$ denote the addition and multiplication in GF(p). Finding an efficient algorithm to compute the discrete logarithm is a known hard unsolved mathematical problem. You'll have the opportunity to play with an algorithm that computes the discrete exponent α^x efficiently in Question 2 of the Examples paper, and Shanks' algorithm to compute the discrete logarithm, also called the "baby-step, giant-step" algorithm (one of the most efficient methods currently known) in Question 4 of the Examples Paper.

The operation of the Diffie Hellman protocol is illustrated in Figure 3.1. The system is used to agree on a secret key x_{AB} over a public authenticated communication channel. The secret key is then typically converted to an appropriate format (e.g. binary) and used as a seed in a stream cipher or as a secret key in a block cipher.

The system components are:

Modulus p and generator α: these are published and define the cryptosystem.

Alice and Bob's secret keys x_A and x_B: these need to be chosen using a strong random number generator whose operation cannot be predicted by a potential attacker. Many successful attacks on implementations of the Diffie-Hellman protocol have taken advantage of weaknesses in the implementation of the random number generator used by the communicating parties.

Alice and Bob's public keys $y_A = \alpha^{x_A}$ and $y_B = \alpha^{x_B}$: these are computed by Alice and Bob using an efficient implementation of discrete exponentiation ("square and multiply": see Question 2 in the Examples Paper) They are published by the communicating parties and accessible to anyone. In practical implementation, there is need

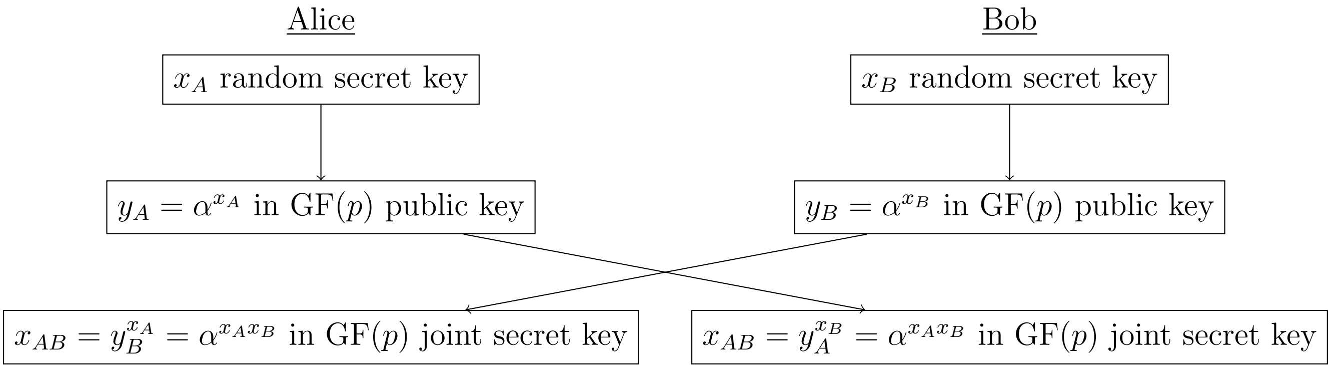
Alice
Bob
x_A random secret key
x_B random secret key
$y_A = \alpha^{x_A}$ in GF(p) public key
$y_B = \alpha^{x_B}$ in GF(p) public key
$x_{AB} = y_B^{x_A} = \alpha^{x_A x_B}$ in GF(p) joint secret key
$x_{AB} = y_A^{x_B} = \alpha^{x_A x_B}$ in GF(p) joint secret key

Figure 3.1: Diffie-Hellman key distribution system

for a trusted authentication authority to publish the public keys in a manner that cannot be tampered with and vouches for their authenticity, i.e., guarantees that the public key belongs indeed to the person listed in the directory as its originator.

Alice and Bob's joint secret $\alpha^{x_A x_B}$: this is computed locally by each recipient using efficient exponentiation, and kept secret.

This technique would allow two people in broad daylight and in full view of others to agree on a secret key that only they know by the end of the protocol, though all those watching are able to hear all communication between them and have access to excellent computing facilities. This was seen as very surprising and counter-intuitive when the method was published and many see Diffie and Hellman's paper as the "bombshell" that started modern cryptology. There are some claims that versions of public-key cryptography were in fact known before to secret researchers working for the British intelligence agency GCHQ but never published.

If you want to learn more details and try to implement Diffie-Hellman, you would need to spend some time investigating:

1. discrete log algorithms known so far and for which special cases their complexity becomes manageable (you will see one such method in the examples paper but there are others.)

2. techniques to choose p and α so as to ensure that currently known discrete log algorithms retain a high complexity

3. algorithms to select large random numbers in GF(p) safely and as unpredictably as possible.

3.4.2 The Rivest-Shamir-Adelman public-key cryptosystem

The Rivest-Shamir-Adleman (RSA) public-key cryptosystem also uses discrete exponentiation but in a different way from Diffie-Hellman. To begin, RSA is an encryption protocol

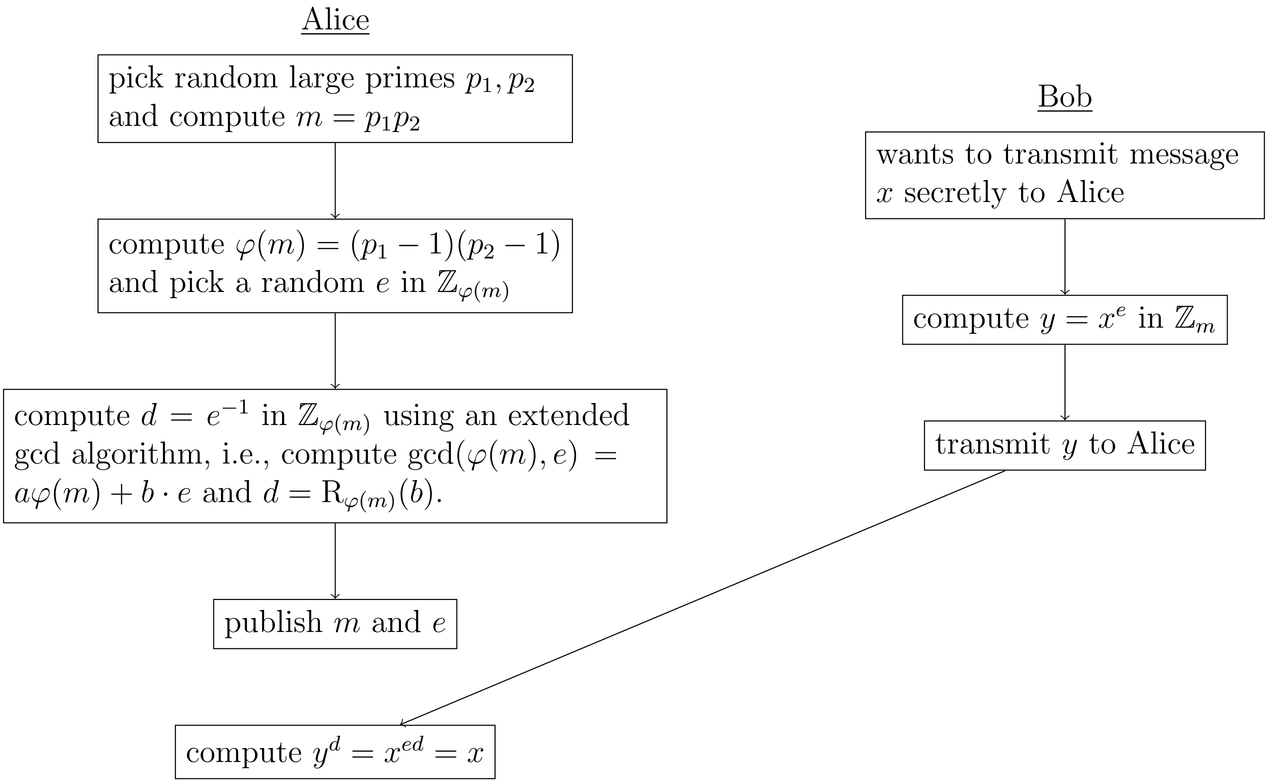
Alice
pick random large primes p_1, p_2 and compute $m = p_1 p_2$
compute $\varphi(m) = (p_1 - 1)(p_2 - 1)$ and pick a random e in $\mathbb{Z}_{\varphi(m)}$
compute $d = e^{-1}$ in $\mathbb{Z}_{\varphi(m)}$ using an extended gcd algorithm, i.e., compute $\gcd(\varphi(m), e) = a\varphi(m) + b \cdot e$ and $d = \mathrm{R}_{\varphi(m)}(b)$.
publish m and e
compute $y^d = x^{ed} = x$
Bob
wants to transmit message x secretly to Alice
compute $y = x^e$ in $\mathbb{Z}_m$
transmit y to Alice

Figure 3.2: The Rivest-Shamir-Adleman (RSA) public-key cryptosystem

unlike Diffie-Hellman which is a key agreement protocol. While the secret key in Diffie-Hellman is in the exponent of α^x and the base α is a system constant, in RSA the plaintext is the base and the secret key will be the exponent. RSA relies not on the difficulty of computing a discrete logarithm but on the difficulty of factoring a large number m, or, equivalently, to find Euler's function $\varphi(m)$ and hence to invert exponentiation in $\mathbb{Z}_m$. The difficulty of inverting a discrete logarithm also helps but only in as much as knowing an algorithm to compute a discrete logarithm would enable a known plaintext attack on RSA.

To understand the operation of RSA, you need to remember that, when considering the calculus of exponentials in $\mathbb{Z}_m$, that all operations in the bases occur in $\mathbb{Z}_m$ whereas all operations in the exponents occur modulo $\varphi(m)$, i.e., in $\mathbb{Z}_{\varphi(m)}$. For example, the following operations in $\mathbb{Z}_m$ illustrate this

$$x^{e_1} \odot_m x^{e_2} = x^{e_1 \oplus_{\varphi(m)} e_2} \text{ and } (x^{e_1})^{e_2} = x^{e_1 \odot_{\varphi(m)} e_2}. \tag{3.2}$$

A flowchart of RSA is drawn in Figure 3.2. The system enables Bob (or anyone else) to transmit secret messages that only Alice can read. Its security relies on the fact that only someone who knows $\varphi(m)$ which can only be obtained from the secret primes p_1, p_2 can find $d = e^{-1}$ in $\mathbb{Z}_{\varphi(m)}$ and hence invert the encryption operation x^e in $\mathbb{Z}_m$.

The protocol enables a user Alice to publish a public key that would allow anyone to transmit messages that only Alice can read. Again, in order to deploy such a system in the real world, there is need for a trusted authentication authority to publish a directory of public keys in a manner that cannot be tampered with, and warrants that the public key listed under Alice's name in the directory belongs indeed to the real user Alice who is the intended recipient of the secret communication.

The system components are:

Alice's initial secrets p_1, p_2: these need to be chosen using reliable and unpredictable generators of large random primes, as any predictability in the selection of p_1 and p_2 could be used to reduce the search space in an attack.

Alice's published key $m = p_1 p_2$ and e: e needs to be an invertible element of $\mathbb{Z}_{\varphi(m)}$, and so can be generated at will but checked so that $\gcd(e, \varphi(m)) = 1$, or generated using the Chinese Remainder Theorem using a factorisation of $\varphi(m)$ if available.

Alice's secrets $\varphi(m)$ and d: the latter is generated using an extended gcd algorithm (preferably Stein's algorithm in this context!)

The security of RSA relies on the generation of large primes, on the difficulty of factoring products of large primes[2], and on the difficulty of computing the discrete logarithm, because in a known plaintext attack where you know x and y, where $x = y^d$ in $\mathbb{Z}_m$, we could compute $d = \log_y x$ if we had an algorithm to compute the discrete logarithm efficiently.

3.5 Problems for Chapter 3

Questions marked as $\star$ are typical exam questions. Questions marked 😎 are borrowed with thanks from Jim Massey's 1990s lectures at ETH Zurich.

Problem 3.5.1: Chosen plaintext attack on an additive stream cipher (😎, $\star$)

By a chosen-plaintext attack on an additive stream cipher [where $Y_i = X_i + Z'_i$ in GF(2) and where the sequence $Z'_1, Z'_2, Z'_3, \ldots$ is called the running key], one generally means that the cryptanalyst is free to choose $X_1, X_2, \ldots, X_L$ for some specified L, but that $X_{L+1}, X_{L+2}, \ldots$are chosen by the sender. Of course, the cryptanalyst also observes the entire ciphertext sequence $Y_1, Y_2, \ldots$.

(a) Show that the chosen-plaintext attack on an additive stream cipher is equivalent to the cryptanalyst being told the first L digits of the running key.

[2] you'll remember Bill Gates' famous 1995 gaffe mentioned in an earlier footnote. This can be excused since he'd probably heard somewhere that factoring products of large primes was an important unsolved mathematical problem with repercussions in computer security, and he just forgot the "products of" part of the statement… We extend our sincere thanks to him for providing material for entertaining footnotes.

(b) Explain why a *necessary* condition for an additive stream cipher to be secure against a chosen-plaintext attack is that the running key have very large linear complexity for virtually all choices of the secret key.

Problem 3.5.2: Fast exponentiation (😎, $\star$)

The "brute force" calculation of α^x, where x is a positive integer, requires $x-1$ multiplications in whatever algebraic system the "number" lies. For instance, $\alpha^5 = (((\alpha\cdot\alpha)\cdot\alpha)\cdot\alpha)\cdot\alpha$. This can always be reduced to at most $2\lfloor\log_2 x\rfloor$ multiplications (where$\lfloor\cdot\rfloor$ denotes the "integer part" of the enclosed number, i.e., the largest integer equal or less than the enclosed number) by the trick of "square and multiply". For instance, $\alpha^5 = (\alpha^2)^2\cdot\alpha$ which requires only 3 multiplications since each squaring requires only one multiplication.

If $x < 2^n$ (or, equivalently, if $\lfloor\log_2 x\rfloor \le n-1$), then x can be written as an n-place radix-two number $[b_{n-1},\ldots,b_1,b_0]_2$, i.e.,

$$x = b_0 + 2b_1 + \cdots + 2^{n-1}b_{n-1}$$

where each b_i is 0 or 1. But then

$$\begin{aligned} \alpha^x &= \alpha^{b_0+2b_1+\cdots+2^{n-1}b_{n-1}} \\ &= (\alpha)^{b_0}\cdot(\alpha^2)^{b_1}\cdot\cdots\cdot(\alpha^{2^{n-1}})^{b_{n-1}} \\ &= \prod_{i:b_i=1}\alpha^{2^i}. \end{aligned} \tag{3.3}$$

The numbers $\alpha^2,\alpha^4,\ldots,\alpha^{2^{n-1}}$ can be formed with $n-1$ squarings. The product in (3.3) requires at most $n-1$ further multiplications, and hence $2(n-1)$ multiplications always suffice.

Use this fast exponentiation technique to compute the following:

(a) 2^{2957} in GF(3989). *Note:* 2 is primitive in GF(3989).

(b) 3^{72} in GF(257). *Note:* 3 is primitive in GF(257).

(c) 2^{72} in GF(257).

(d) 3^{11} in $\mathbb{Z}_{35}$. (This is what many people somewhat imprecisely call "3^{11} modulo 35".)

When we wish to consider the number α to be fixed, then we call computing α^x "exponentiation". In this case, we can precompute and store $\alpha,\alpha^2,\alpha^4,\ldots,\alpha^{2^{n-1}}$. The calculation of α^x via (3.3) then requires at most $\lfloor\log_2 x\rfloor$ multiplications.

(e) Use this fast method of exponentiation to compute $3^{128}, 3^{192}$ and 3^{47} in GF(257).

Problem 3.5.3: Diffie-Hellman Public-Key Distribution "Mini-System" (😎, $\star$)

Because $p = 47$ is a prime for which $p - 1$ has a large prime factor ($47 - 1 = 2 \cdot 23$), it might make a good choice for a Diffie-Hellman public-key distribution "mini-system", that uses a six-digit binary key. Suppose that you are user No. 1 in this system, that $\alpha = 5$ is specified, that your secret number is $x_1 = 13$ and hence that your public directory number is $y_1 = \alpha^{13} = 43$. You wish to send a message to user No. 2 whose public number is $y_2 = 33$. What six-digit binary key will you use in your conventional cryptosystem?

You will get some idea of the security of this system if you try by hand to find the secret number x_2.

Problem 3.5.4: Shank's Algorithm to Compute Discrete Logarithm (😎)

The fastest (in the sense of fewest operations in the cyclic group) of known algorithms for solving the general discrete logarithm problem is Shank's algorithm. [There are more complicated general algorithms that are just as fast but use less storage. For specific cyclic groups, there are often much faster algorithms.] Let α be a specific generator of a cyclic group of order n. The problem is to find x (where $0 \leq x < n$) when given $y = \alpha^x$, i.e., to find $\log_\alpha y$. Shank's algorithm exploits the fact that, for any positive integer d, x can be written uniquely as

$$x = qd + r$$

where $0 \leq r < d$ and $0 \leq q < n/d$. Note that for $d \approx \sqrt{n}$, there are about the same number of possible values of r and of q, namely about $\sqrt{n}$. Finding x is equivalent to finding q and r with $0 \leq r < d$ and $0 \leq q < n/d$ such that $\alpha^{qd+r} = y$ or, again equivalently, such that

$$\alpha^r = (\alpha^{-d})^q y.$$

***Shank's Algorithm:* (*α, n and y are inputs*)**

1) Choose a convenient positive integer d, $d \approx \sqrt{n}$.

2) Compute $\beta = \alpha^{-d} = \alpha^{n-d}$ by square-and-multiply.

3) Starting with $(0, 1)$, make a table with entries (r, α^r) for $r = 0, 1, \ldots, d - 1$ by adding 1 to the first member and multiplying the second member of the previous entry by α.

4) Sort the table on the second member of each entry to make easy retrieve by the second member possible.

   Note: You now have a table with entries $(\log_\alpha y, y)$ for all y for which $0 \leq \log_\alpha y < d$, that is easily accessed on y. Thus one can easily find $\log_\alpha y$ whenever this logarithm is small.

5) Set $q = 0$ and $t = y$.

6) If there is an entry in the table whose second member is t, then set r equal to its first member, set $x = qd + r$, and then stop. Otherwise, increase q by 1, modify t by multiplying it by β and return to step 6).

The bulk in the computation of Shank's algorithm occurs in step (3) and (6), which in the worst case takes about $2\sqrt{n}$ multiplications.

The element $\alpha = 7$ is a primitive element of GF(359), i.e., its multiplicative order is $n = 358$. Note that $358 = 2 \times 179$ and that 179 is a prime so we are working here in a GF(p) such that $p - 1$ has a large prime factor, which is a necessary condition for the discrete logarithm problem to be difficult. Find $\log_\alpha y$ by Shank's algorithm for each of the following cases:

(a) $y = 100$.

(b) $y = 2$.

(c) $y = 281$.

Problem 3.5.5: Rivest-Shamir-Adleman (RSA) Public-Key Cryptosystem (😎, $\star$)

Just to convince yourself that the Rivest-Shamir-Adleman (RSA) public-key cryptosystem decrypts as claimed, choose the plaintext X randomly in the range $1 \leq X < m$ and form the cryptogram $Y = R_m(X^e)$ where $(m, e) = (667, 191)$ are found in the public directory. Then decrypt as $R_m(Y^d)$ to recover X, where d is the number that only you can determine from your secret knowledge that $m = 23 \times 29$, i.e., $d = R_{\varphi(m)}(b)$ where $\varphi(m) = 22 \times 28 = 616$ and b is the number that you can find from Euclid's algorithm (see Fig. 1.1) such that $1 = \gcd(616, 191) = a \times 616 + b \times 191$. If that strikes you as too much work, use the simple numbers $m = 5 \times 7$, $e = 11$, and $\varphi(m) = 24$.

Problem 3.5.6: Breaking RSA knowing $\varphi(m)$ (😎, $\star$)

You are a cryptanalyst trying to break the RSA cryptosystem for which $m = 4003997$ and $e = 379$ are in the public directory. After some computation, you have discovered that $\varphi(m) = 3999996$.

(a) Find the deciphering exponent d.

(b) Now find the primes p_1 and p_2 such that $m = p_1 p_2$.

Problem 3.5.7: Cryptanalysis by repeated Encrypting of RSA (😎)

One of the "tricks" in the bag of the cryptanalyst is to try repeated encryption of the cryptogram, i.e., to form $E(\mathbf{Y}), E(E(\mathbf{Y}))$, etc. where E is the encrypting function. [Note that E is known and easily computable in a public-key cryptosystem (PKC).] It often happens for even a secure-looking cryptosystem that the plaintext will result from a small number of these encryptions. This would, of course, be a fatal flaw in a PKC.

Verify, in the lazy man's RSA PKC of Problem 3.5.5 ($m = 35$), that $E(Y) = X$ for any legitimate e [i.e., an e such that $0 < e < 24$ and $\gcd(e, 24) = 1$] so that this system is *completely insecure*.

Hint: Because $E(Y) = Y^e = (X^e)^e = X^{e^2}$ and $\varphi(m) = 24$, you need only show that $R_{24}(e^2) = 1$ for all legitimate e.

Note: By choosing p (and similarly q) such that $p-1$ has a large prime factor p' where $p' - 1$ again has a large prime factor p'', this type of attack can be prevented.