CISS451: Cryptography

Abhishek Sharma (January 29, 2026)

Contents

Chapter 1

Classical ciphers

1.1 Classical ciphers: instructions

The subtitle: Stuff that you should not use anymore.

However some of these old stuff is important because their ideas are used in modern-day cryptography.

Here we go ...

1.2 Shift cipher debug: classical-cipher-shift-cipher.tex

Shift Cipher and the Caesar Cipher

A **shift cipher** encrypts each letter of the plaintext by moving it forward in the alphabet by a fixed number k, called the **key**, with $0 \leq k < 26$. In particular:

Definition 1.2.1. The **Caesar cipher** is the shift cipher with key $k = 3$. Under this cipher, encryption replaces each letter by the one three positions later, wrapping around from z to a, and decryption reverses the process.

For example, with $k = 3$ we have

$$a \mapsto d, \quad b \mapsto e, \quad c \mapsto f, \quad \ldots$$

and because the alphabet "goes in a circle," also

$$x \mapsto a, \quad y \mapsto b, \quad z \mapsto c.$$

Thus the word

`hello`

encrypts to

`khoor`.

In code, a single-character encryption function can be written as follows:

```
def E(x):
    i = ord(x) - ord('a')
    i = (i + 3) % 26
    return chr(ord('a') + i)
```

and equivalently in C++:

```
char E(char x)
{
    return (x - 'a' + 3) % 26 + 'a';
}
```

Because shifting by 26 positions returns each letter to itself, the only meaningful keys are

$$K = \{0, 1, 2, \ldots, 25\}.$$

Let P and C be two sets. A **cipher** consists of two maps

$$E\colon P \to C \quad \text{and} \quad D\colon C \to P$$

called the **encryption** and **decryption** respectively, satisfying

$$D\big(E(x)\big) = x \quad \text{for every } x \in P.$$

Here, elements of P are called **plaintexts** and elements of C are called **ciphertexts**.

Definition 1.2.2. A **symmetric cipher** (or **private-key cipher**) is given by two functions

$$E\colon K \times P \to C, \quad D\colon K \times C \to P,$$

such that for each key $k \in K$ one has

$$D\big(k,\ E(k, x)\big) = x \quad \text{for all } x \in P.$$

In other notation one often writes $E_k(x)$ for $E(k, x)$ and $D_k(y)$ for $D(k, y)$.

A symmetric cipher uses the same secret key for both encryption and decryption. By contrast:

Definition 1.2.3. An **asymmetric cipher** (or **public-key cipher**) employs a pair of distinct keys, an **encryption key** k and a **decryption key** k'. Its maps

$$E\colon P \to C \quad \text{and} \quad D\colon C \to P$$

must satisfy

$$D\big(E(x)\big) = x \quad \text{for all } x \in P,$$

when using E with the public key k and D with the private key k'. That is,

$$D_{k'}\big(E_k(x)\big) = x.$$

Here E_k may be shared openly, while k' remains confidential.

Kerckhoffs' principle tells us that the security of a cipher should rest solely on the secrecy of the key, not on the obscurity of the algorithm. Modern cryptography follows this by publishing algorithms and relying on rigorous analysis to ensure that only possession of the key allows decryption.

As a concrete example, the classical **shift cipher** (also known as the Caesar cipher) is a symmetric cipher where

$$K = \{0, 1, 2, \ldots, 25\}, \quad P = C = \{0, 1, 2, \ldots, 25\},$$

and for each key $k \in K$ we define

$$E_k(x) \; = \; x + k \; (\mathrm{mod}\, 26), \qquad D_k(y) \; = \; y - k \; (\mathrm{mod}\, 26).$$

It is immediate that

$$D_k\big(E_k(x)\big) \equiv x \pmod{26},$$

so decryption inverts encryption exactly.

In particular, choosing $k = 3$ yields the traditional Caesar cipher: encryption shifts each letter forward by three positions, and decryption shifts backward by three. “‘

1.3 Modular arithmetic debug: classical-cipher-modular-arithmetic.tex

Number thoery studies whole numbers and thier nature but is not limited to them. Infact, it is only a very small part of number theory. Number theory is huge in CS and is one of the most important topics. The problem statements might seem simple and easy enough to understand however the proof, concpepts and techniques requrie intense knowledge of mathematics.

Gaus said ‘Math is the queen of science and number theory is the queen of mathemactics‘

We will cover Elementary Number Theory. The name sounds misleading and the topic is not elementary at all. We only will study a small portion of number theory. The part we will cover will involve the part of whole numbers(integers)

However, in research in number thoery requires real numbers, complex numbers , geometry, complex analysis etc.

Let’s look at modular arthematic. Modular Arthematic is used in cyptography, data compression and error correction codes. In modular arithmetic, we perform calculations within a fixed range of numbers, where the result ”wraps around” when it exceeds that range. It’s often described as ”clock arithmetic” because it behaves similarly to how hours wrap around on a 12-hour clock (after 12 comes 1, not 13).

The set of integers $\{..., -3, -2, -1, 0, 1, 2, 3, ...\}$ denoted $\mathbb{Z}$ has two operations $+$ and $\cdot$. In terms of the algebraic structure (i.e. the operations), $\mathbb{Z}$ is known as a **commutative ring**. Basically a commutative ring is a set of “things” with two operations, addition and multiplication, with rules that look like the addition and multiplication rules for $\mathbb{Z}$. For instance one such rule in $\mathbb{Z}$ is

$$a(b+c) = ab + ac$$

This same rule holds true for $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ and polynomials with coefficients in $\mathbb{Z}$.

$\mathbb{Z}$ is known as a communtative ring.

Think of it in this way, as a function that would work for the argument it takes.

You can think of $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ as subclasses of `CommutativeRing`. Therefore if you have a function

```
// Base abstract class
class CommutativeRing
{
public:
    virtual CommutativeRing* add(const CommutativeRing& other) = 0;
    virtual CommutativeRing* multiply(const CommutativeRing& other) = 0;
    virtual bool equals(const CommutativeRing& other) = 0;
    virtual CommutativeRing* additiveInverse() = 0;
};

class Z : public CommutativeRing {
private:
    int value;
public:
    // Implementation of ring operations for integers
};

class Q : public CommutativeRing
{
private:
    int numerator;
    int denominator;
public:
    // Implementation of ring operations for rationals

};
```

```
class Field : public CommutativeRing {
public:
    virtual Field* multiplicativeInverse() = 0;
    // A field has multiplicative inverses for non-zero elements
};

// Q, R, and C would inherit from Field instead of CommutativeRing
```

then `f` can work with `x` if `x` is a object of $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$.

Another way we can talk about commutative ring. Let's talk about an example for a ring, Let's say we have a set X such that X has elements a and b. X is a ring that satisfies the condition, $a - b = a + b$

this means that if there is a element c such that $c \in X$ then c will satisfy the condition. On a real note, we know that the only numbers that would statisfy that condition is 0.

The idea is to create an abstract general solution that we can apply to the whole set or the whole ring.

A **commutatative ring** R is a set of "things" with two operations abstractly denoted by $\oplus$ and $\odot$ ("addition" and "multiplication") Furthermore there are two special "things" in R which we will call 0_R and 1_R. The properties satisfied by $R, 0_R, 1_R, \oplus, \odot$ (of course there must be something satisfied by them!) are as follows. For $\oplus$ the properties are:

(a) $r \oplus s$ is in R for all r, s in R
(b) $(r \oplus s) \oplus t = r \oplus (s \oplus t)$ for all r, s, t in R
(c) $r \oplus 0_R = r = 0_R \oplus r$ for all r in R
(d) For all r in R there is something in R which we will call r' such that $r \oplus r' = 0_R = r' \oplus r$

For $\odot$ the properties are:

(a) $r \odot s$ is in R for all r, s in R
(b) $(r \odot s) \odot t = r \odot (s \odot t)$ for all r, s, t in R
(c) $r \odot 1_R = r = 1_R \odot r$ for all r in R
(d) $r \odot s = s \odot r$ for all r, s in R

The property involving both $\oplus$ and $\odot$ is

(a) $r \odot (s \oplus t) = r \odot s \oplus r \odot t$

Commutative ring $(R, +, \cdot, 0, 1)$ satisfies:

- $(R, +)$ is an abelian group.
- $(R, \cdot)$ is a commutative monoid.
- Distributivity: $r\,(s + t) = r\,s + r\,t$.

Example: $\mathbb{Z}$ under usual + and ×.

Residue ring $\mathbb{Z}/n\mathbb{Z}$:

$$x \equiv y \pmod{n} \iff n \mid (x - y), \quad \{0, 1, \ldots, n-1\},$$

operations performed in $\mathbb{Z}$ then reduced mod n.

Case $n = 26$:

$$\{0, \ldots, 25\}, \quad 3 + 25 = 28 \equiv 2, \quad 7 \cdot 4 = 28 \equiv 2 \pmod{26}.$$

($\mathbb{Z}/26\mathbb{Z}$ is not a field.)

Caesar cipher:

$$a \mapsto 0, \ldots, z \mapsto 25, \quad E(p) = (p + k) \bmod 26, \quad D(c) = (c - k) \bmod 26.$$

Commutative ring $(R, +, \cdot, 0, 1)$ satisfies:

- $(R, +)$ is an abelian group.
- $(R, \cdot)$ is a commutative monoid.
- Distributivity: $r\,(s + t) = r\,s + r\,t$.

Example: $\mathbb{Z}$ under usual + and ×.

Residue ring $\mathbb{Z}/n\mathbb{Z}$:

$$x \equiv y \pmod{n} \iff n \mid (x - y), \quad \{0, 1, \ldots, n - 1\},$$

operations performed in $\mathbb{Z}$ then reduced mod n.

Case $n = 26$**:**

$$\{0, \ldots, 25\}, \quad 3 + 25 = 28 \equiv 2, \quad 7 \cdot 4 = 28 \equiv 2 \pmod{26}.$$

($\mathbb{Z}/26\mathbb{Z}$ is not a field.)

Caesar cipher:

$$a \mapsto 0, \ldots, z \mapsto 25, \quad E(p) = (p + k) \bmod 26, \quad D(c) = (c - k) \bmod 26.$$

Commutative ring $(R, +, \cdot, 0, 1)$ satisfies:

- $(R, +)$ is an abelian group.
- $(R, \cdot)$ is a commutative monoid.
- Distributivity: $r\,(s + t) = r\,s + r\,t$.

Example: $\mathbb{Z}$ under usual + and ×.

Residue ring $\mathbb{Z}/n\mathbb{Z}$:

$$x \equiv y \pmod{n} \iff n \mid (x - y), \quad \{0, 1, \ldots, n - 1\},$$

operations performed in $\mathbb{Z}$ then reduced mod n.

Case $n = 26$:

$$\{0, \ldots, 25\}, \quad 3 + 25 = 28 \equiv 2, \quad 7 \cdot 4 = 28 \equiv 2 \pmod{26}.$$

($\mathbb{Z}/26\mathbb{Z}$ is not a field.)

Caesar cipher:

$$a \mapsto 0, \ldots, z \mapsto 25, \quad E(p) = (p + k) \bmod 26, \quad D(c) = (c - k) \bmod 26.$$

Commutative ring $(R, +, \cdot, 0, 1)$ satisfies:

- $(R, +)$ is an abelian group.
- $(R, \cdot)$ is a commutative monoid.
- Distributivity: $r\,(s + t) = r\,s + r\,t$.

Example: $\mathbb{Z}$ under usual + and ×.

Residue ring $\mathbb{Z}/n\mathbb{Z}$:

$$x \equiv y \pmod{n} \iff n \mid (x - y), \quad \{0, 1, \ldots, n - 1\},$$

operations performed in $\mathbb{Z}$ then reduced mod n.

Exercise 1.3.1.

(a) True or false: $100 \equiv 74 \pmod{26}$
(b) True or false: $-3 \equiv 133 \pmod{26}$
(c) True or false: $-20 \equiv 21 \pmod{26}$
(d) True or false: $7 \equiv 3 \pmod{3}$
(e) True or false: $17 \equiv -3 \pmod{5}$
(f) True or false: $21 \equiv 11 \pmod{8}$
(g) True or false: $42 \equiv 0 \pmod{7}$

(Go to solution, page **??**) □

Exercise 1.3.2.

(a) Simplify $100 \pmod{26}$.
(b) If you have very simple calculator with +, −, ∗, / how would simplify $131246845 \pmod{26}$? You have 10 seconds ... the clock is ticking ...

(c) Simplify $33 \pmod{26}$.
(d) Solve $2x + 1 \equiv 0 \pmod{26}$ by brute force. Use Python or C++.
(e) Solve $10x + 20 \equiv 4x + 78 \pmod{26}$ by brute force. Use Python or C++.
(f) Simplify $10x + 20 \equiv 4x + 78 \pmod{26}$ first, and then solve it by brute force. Do you get the same results as in the previous part?
(g) Solve $42x^5 + 10x + 1 \equiv 73 \pmod{3}$.
(h) Solve $(1000000x + 2)^3 \equiv 2 \pmod{3}$.

(Go to solution, page **??**) □

Exercise 1.3.3.

(a) If $x \equiv 1 \pmod{7}$ and $x \equiv 5 \pmod{13}$, what can you tell me about x?
(b) If $x \equiv 1 \pmod{7}$ and $x \equiv 5 \pmod{13}$, what can you tell me about $x \pmod{7 \cdot 13}$?
(c) $x \equiv 2 \pmod{7 \cdot 13}$ what can you tell me about $x \pmod{7}$ and $x \pmod{13}$?
(d) True or false: $x^3 \equiv x^0 \pmod{3}$ since the 3 in the exponent can be replaced by 0.
(e) In $\mathbb{Z}$, is it true that $(x + y)^2 = x^2 + y^2$? Try some values for x and y.
(f) In $\mathbb{Z}/2$, is it true that $(x + y)^2 = x^2 + y^2 \pmod{2}$? Try all values for x and y.
(g) In $\mathbb{Z}$, is it true that $(x + y)^3 = x^3 + y^3$? Try some values for x and y.
(h) In $\mathbb{Z}/3$, is it true that $(x + y)^3 \equiv x^3 + y^3 \pmod{3}$? Try all values for x and y.
(i) In $\mathbb{Z}/n$, is it true that $(x + y)^n \equiv x^n + y^n \pmod{n}$? Can you prove your claim.

(Go to solution, page **??**) □

Exercise 1.3.4. Just like for boolean values, you can write down the complete behavior of the boolean and and boolean or and boolean not operators (these are called truth tables), you can also complete specify the complete behavior of addition in mod 26, "negative of" in mod 26, multiplication in mod 26, and also multiplicative inverse mod 26. The multiplicative inverse of x in mod 26 is just the number y mod 26 such that

$$xy \equiv 1 \pmod{26}$$

The multiplicative inverse of $x \pmod{26}$ is written $x^{-1} \pmod{26}$ – this is an integer mod 26!!! It's not a fraction in $\mathbb{R}$!!! Sometimes $x^{-1} \pmod{26}$ might not exist. In that case write None. Write down these 4 tables.

Addition table for $\mathbb{Z}/26$:

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0																										
1																										
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16																										
17																										
18																										
19																										
20																										
21																										
22																										
23																										
24																										
25																										

In the above, when I write 5, I meant of course 5 (mod 26).

Multiplication table for $\mathbb{Z}/26$:

×	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0																										
1																										
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16																										
17																										
18																										
19																										
20																										
21																										
22																										
23																										
24																										
25																										

Negative of table for $\mathbb{Z}/26$:

$x \pmod{26}$	$-x \pmod{26}$
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Multiplicative inverse table for $\mathbb{Z}/26$:

$x \pmod{26}$	$x^{-1} \pmod{26}$
0	None
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

It should be clear that $0 \pmod{26}$ does not have an inverse.

(Go to solution, page **??**) □

1.3.2 a) Simplify $100 \pmod{26}$. $100 = 26 \cdot 3 + 22$ Therefore, $100 \equiv 22 \pmod{26}$

b) Simplify $131246845 \pmod{26}$ using basic operations. $131246845 \div 26 = 5047955.57...$ So $131246845 = 26 \cdot 5047955 + 15$ Therefore, $131246845 \equiv 15 \pmod{26}$

c) Simplify $33 \pmod{26}$. $33 = 26 \cdot 1 + 7$ Therefore, $33 \equiv 7 \pmod{26}$

d) Solve $2x + 1 \equiv 0 \pmod{26}$ by brute force.

```
for x in range(26):
    if (2*x + 1) % 26 == 0:
        print(f"Solution: x = {x}")
```

The solution is $x = 13$ because $2 \cdot 13 + 1 = 27 \equiv 1 \pmod{26}$

e) Solve $10x + 20 \equiv 4x + 78 \pmod{26}$ by brute force.

```
for x in range(26):
    if (10*x + 20) % 26 == (4*x + 78) % 26:
        print(f"Solution: x = {x}")
```

The solution is $x = 1$.

f) Simplify $10x + 20 \equiv 4x + 78 \pmod{26}$ first, then solve.
$10x - 4x \equiv 78 - 20 \pmod{26}$
$6x \equiv 58 \pmod{26}$
$6x \equiv 6 \pmod{26}$ (since $58 = 26 \cdot 2 + 6$)
$x \equiv 1 \pmod{26}$
So $x = 1$ is the solution, which matches the brute force approach.

g) Solve $42x^5 + 10x + 1 \equiv 73 \pmod{3}$.
$42 \equiv 0 \pmod{3}$, $10 \equiv 1 \pmod{3}$, $1 \equiv 1 \pmod{3}$, $73 \equiv 1 \pmod{3}$
The equation becomes: $0 \cdot x^5 + 1 \cdot x + 1 \equiv 1 \pmod{3}$
$x + 1 \equiv 1 \pmod{3}$
$x \equiv 0 \pmod{3}$
Therefore $x = 0$ is the solution.

h) Solve $(1000000x + 2)^3 \equiv 2 \pmod{3}$.
$1000000 \equiv 1 \pmod{3}$, so $(1000000x + 2) \equiv (x + 2) \pmod{3}$
We need to solve $(x + 2)^3 \equiv 2 \pmod{3}$ true For $x = 0$: $(0 + 2)^3 = 8 \equiv 2 \pmod{3}$ false For $x = 1$: $(1 + 2)^3 = 27 \equiv 0 \pmod{3}$ false For $x = 2$: $(2 + 2)^3 = 64 \equiv 1 \pmod{3}$ Therefore $x = 0$ is the solution.

1.3.3 a) If $x \equiv 1 \pmod{7}$ and $x \equiv 5 \pmod{13}$, then $x = 7k + 1 = 13m + 5$ for some integers k and m.

b) By the Chinese Remainder Theorem, there is a unique solution modulo $7 \cdot 13 = 91$.
$x \equiv 1 \cdot 13 \cdot (13^{-1} \pmod{7}) + 5 \cdot 7 \cdot (7^{-1} \pmod{13}) \pmod{91}$
$13^{-1} \pmod{7} = 6$ and $7^{-1} \pmod{13} = 2$
$x \equiv 1 \cdot 13 \cdot 6 + 5 \cdot 7 \cdot 2 \pmod{91} \equiv 78 + 70 \equiv 57 \pmod{91}$

c) If $x \equiv 2 \pmod{91}$ where $91 = 7 \cdot 13$, then:
$x \equiv 2 \pmod{7}$ and $x \equiv 2 \pmod{13}$

d) False. $x^3 \not\equiv x^0 \pmod{3}$ in general.
For instance, if $x = 2$, then $2^3 = 8 \equiv 2 \pmod{3}$ while $2^0 = 1 \pmod{3}$.

e) In $\mathbb{Z}$, $(x + y)^2 = x^2 + 2xy + y^2 \neq x^2 + y^2$ in general.
For example, $(1 + 2)^2 = 9 \neq 1^2 + 2^2 = 5$.

f) In $\mathbb{Z}/2\mathbb{Z}$, let's try all values:
$(0+0)^2 \equiv 0^2 + 0^2 \equiv 0 \pmod{2}$ t $(0+1)^2 \equiv 1^2 + 0^2 \equiv 1 \pmod{2}$t
$(1+0)^2 \equiv 1^2 + 0^2 \equiv 1 \pmod{2}$ f $(1+1)^2 \equiv 0^2 + 1^2 \equiv 1 \pmod{2}$ t
So it's not true that $(x+y)^2 \equiv x^2 + y^2 \pmod{2}$ for all values.

g) In $\mathbb{Z}$, $(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3 \neq x^3 + y^3$ in general.
For example, $(1+1)^3 = 8 \neq 1^3 + 1^3 = 2$.

h) In $\mathbb{Z}/3\mathbb{Z}$, $(x+y)^3 \equiv x^3 + y^3 \pmod{3}$ is true.
$(x+y)^3 = x^3+3x^2y+3xy^2+y^3 \equiv x^3+0+0+y^3 \equiv x^3+y^3 \pmod{3}$

i) In $\mathbb{Z}/n\mathbb{Z}$, $(x+y)^n \equiv x^n + y^n \pmod{n}$ is true.
From binomial theorem: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$
For $1 \leq k \leq n-1$, $\binom{n}{k}$ is divisible by n
Therefore, $(x+y)^n \equiv x^n + y^n \pmod{n}$

1.3.4 Tables for modular arithmetic mod 26:
Addition: $a + b \equiv (a+b) \pmod{26}$
Negation: $-a \equiv 26 - a \pmod{26}$
Multiplication: $a \cdot b \equiv (a \cdot b) \pmod{26}$
Multiplicative inverse mod 26: $1^{-1} \equiv 1 \pmod{26}$ $3^{-1} \equiv 9 \pmod{26}$ $5^{-1} \equiv 21 \pmod{26}$ $7^{-1} \equiv 15 \pmod{26}$ $9^{-1} \equiv 3 \pmod{26}$ $11^{-1} \equiv 19 \pmod{26}$ $15^{-1} \equiv 7 \pmod{26}$ $17^{-1} \equiv 23 \pmod{26}$ $19^{-1} \equiv 11 \pmod{26}$ $21^{-1} \equiv 5 \pmod{26}$ $23^{-1} \equiv 17 \pmod{26}$ $25^{-1} \equiv 25 \pmod{26}$ For all other values of a, $a^{-1} \equiv \text{None} \pmod{26}$

Exercise 1.3.5. Solve

$$3x = 8$$

in $\mathbb{Z}$. Can you? Now solve

$$3x \equiv 8 \pmod{26}$$

(Go to solution, page **??**) □

Exercise 1.3.6.

(a) Solve

$$3x = 1$$

in $\mathbb{Z}$. Of course you can't! Now solve

$$3x = 1$$

in $\mathbb{Q}$. Of course you can! We would say that $\frac{1}{3}$ is the multiplicative inverse of 3 in $\mathbb{Q}$. Also, we would say that, in $\mathbb{Z}$, 3 is not invertible.

What about

$$3x \equiv 1 \pmod{26}$$

Can you? If you can then you have found a multiplicative inverse of 3 (mod 26).

(b) Now try to solve

$$ax \equiv 1 \pmod{26}$$

for $a \equiv 0, 1, 2, ..., 25 \pmod{26}$. Which a's have multiplicative inverses? Is there a pattern to a's with multiplicative inverses mod 26 and who which do not?

(c) Now try

$$ax \equiv 1 \pmod{N}$$

where N is a positive integer and $a \equiv 0, 1, 2, ..., N-1 \pmod{N}$; say you try $N = 4, 5, 6, 7, 8$. Notice something? See a pattern? Is there a pattern to a's with multiplicative inverses mod 26 and who which do not?

(Go to solution, page **??**) □

Exercise 1.3.7. Solve

$$3x = 13$$

in $\mathbb{Z}$. Of course you can't! Now solve

$$3x = 13$$

in $\mathbb{Q}$. Of course you can! Do you realize that you solved it using the multiplicative inverse of 3 in $\mathbb{Q}$? What about

$$3x \equiv 13 \pmod{26}$$

Are you going to use brute force and try all values (mod 26)? (HINT: You had better not.) (Go to solution, page **??**) □

Exercise 1.3.8. Solve

$$3x - 5 \equiv 10x + 20 \pmod{26}$$

by brute force. Next, simplify the above first before solving it. Do you get the same solutions? (Go to solution, page **??**) □

Exercise 1.3.9. Solve

$$3x + 12 \equiv 10x + 23 \pmod{26}$$

by brute force. Next, simplify the above first before solving it. (Are you sure it's really simplifed? HINT: Multiplicative inverse.) Do you get the same solutions? (Go to solution, page **??**) □

Exercise 1.3.10.

(a) Can you solve

$$x^2 \equiv 1 \pmod{26}$$

Of course one solution is 1 (mod 26). Can you find the other one in 1 second? Are there just two? Or are there more than two solutions? These are (of course) square roots of 1, but in mod 26 and not $\mathbb{R}$.

(b) What about

$$x^2 \equiv 2 \pmod{26}$$

(c) Keep going ... try to solve

$$x^2 \equiv a \pmod{26}$$

for all cases. Draw a table for the square roots of $a \pmod{26}$ for all a's.

(Go to solution, page **??**) □

Exercise 1.3.11. Continuing the previous exercise ...

(a) Next, try to solve

$$x^2 \equiv a \pmod{N}$$

for all $a \pmod{N}$ for at least 3 values of N. See any patterns?

(b) The above is probably too tough. What if you try

$$x^2 \equiv 2 \pmod{p}$$

for primes p? Try a few primes (maybe 20-30) and make a table. Notice a pattern?

(Go to solution, page **??**) □

Exercise 1.3.12.

(a) What are all the possible values of $x^2 \pmod 4$?
(Note: The key thing to note is that $x \pmod 4$ can take all the values 0, 1, 2, 3 mod 4. But the squares $x^2 \pmod 4$ can only certain values mod 4.)
(b) What are all the possible values of $x^2 \pmod 5$?
(c) What about squares mod 7?
(d) What about squares mod 15? (Be efficient.)
(e) Prove that there are no integer solutions to

$$5x^2 - 8y^2 = 3$$

(HINT: Prove this by contradiction. Take mod 4 ... and ... Checkmate!) Without modular arithmetic, there's no clear way to solve this problem! Go number theory!)

(Go to solution, page **??**) □

Exercise 1.3.13.

(a) Solve

$$5x^2 + y^2 = 3$$

(HINT: You don't really need number theory for this one. Why? But if you want to, imitate the solution for the previous problem.)
(b) Solve

$$11y^2 - 5x^2 = 3$$

(HINT: This is just a slight change from the previous problem. *But* now you need number theory. Try mod 4. If it does not work, try mod 5. Repeat.)
(c) Solve

$$y^2 - 5x^2 = 2$$

(d) What about this one:

$$x^2 - 5y^2 = 1$$

[Aside. Integer solutions to $x^2 - dy^2 = 1$ has been studied since at least 400BC. This equation appear the Cattle Problem of Archimedes:

> If thou art diligent and wise, O stranger, compute the number of cattle of the Sun, who once upon a time grazed on the fields of the Thrinacian isle of Sicily, divided into four herds of different colours, one milk white, another a glossy black, the third yellow and the last dappled. In each herd were bulls, mighty in number according to these proportions: Understand, stranger, that the white bulls were equal to a half and a third of the black together with the whole of the yellow, while the black were equal to the fourth part of the dappled and a fifth, together with, once more, the whole of the yellow. Observe further that the remaining bulls, the dappled, were equal to a sixth part of the white and a seventh, together with all the yellow. These were the proportions of the cows: The white were precisely equal to the third part and a fourth of the whole herd of the black; while the

> black were equal to the fourth part once more of the dappled and with it a fifth part, when all, including the bulls went to pasture together. Now the dappled in four parts8 were equal in number to a fifth part and a sixth of the yellow herd. Finally the yellow were in number equal to a sixth part and a seventh of the white herd. If thou canst accurately tell, O stranger, the number of cattle of the Sun, giving separately the number of well-fed bulls and again the number of females according to each colour, thou wouldst not be called unskilled or ignorant of numbers, but not yet shall thou be numbered among the wise. But come, understand also all these conditions regarding the cows of the Sun. When the white bulls mingled their number with the black, they stood firm, equal in depth and breadth, and the plains of Thrinacia, stretching far in all ways, were filled with their multitude. Again, when the yellow and the dappled bulls were gathered into one herd they stood in such a manner that their number, beginning from one, grew slowly greater till it completed a triangular figure, there being no bulls of other colours in their midst nor none of them lacking. If thou art able, O stranger, to find out all these things and gather them together in your mind, giving all the relations, thou shalt depart crowned with glory and knowing that thou hast been adjudged perfect in this species of wisdom.

If W, X, Y, Z represents the number of white, black, yellow, dappled bulls, you will get a systems of 7 linear equations, the first two being

$$W = (1/2 + 1/3)X + Z$$
$$X = (1/4 + 1/5)Y + Z$$

together with some constraints such as $W + X$ must be a square. After some manipulations, it can be shown that the equation to solve looks like

$$x^2 - 410286423278424y^2 = 1$$

What was Archimedes thinking? You are find information on the Archimedes Cattle Problem on the web.]

(Go to solution, page **??**) □

Exercise 1.3.14. Consider

$$5x^2 - 8y^2 = 3z^2$$

where we are only interested in finding integer solutions. Of course $(0, 0, 0)$ is a solution, but that's easy. (Right?) In general, an equation like

$$ax^p + by^q = cz^r$$

where $p, q, r > 0$ always has $(0, 0, 0)$ as a solution. $(0, 0, 0)$ is sometimes called the trivial solution. So we might as well assume $(x, y, z) \neq (0, 0, 0)$, i.e., assume they are not all zero. By the way the polynomial

$$5x^2 - 8y^2 - 3z^2$$

is a sum of terms where the number of variables appearing in each term is the same, i.e., 2. Such polynomial are is said to be homogeneous of degree 2.

How many solutions can you find? Is there any at all? Is there a finite number of solutions? If there are infinitely many solutions, can you write them down?

If not all of them, maybe an infinite family of them. For instance it would be nice to say: "For any integer n, $x = n, y = 2n, z = n + 5$ is a solution". This would be a 1–parameter family of solutions.

(a) What's the first thing you should do?
(b) Prove that if x, y, z is a solution, then nx, ny, nz is also a solution for any integer n.
(c) In case you can't see a solution right away, do the following. If (x, y, z) are solutions, show that x and z must be even. (HINT: mod 4.) This is helpful since we won't have to check the odd x or odd z caaes. That cuts down a brute force search down by 3/4.
(d) Continuing the above: let $x = 2a$, $y = b$, and $z = 2c$, substitute, and you'll get a new equation in a, b, c.
(e) The equation in a, b, c is easier than the one in x, y, z. Why? Because a solution in the equation in a, b, c for the above question would correspond to a solution x, y, z in the original equatison and a, b, c solution is smaller. Can you now find some solutions for the equation in a, b, c?
(f) I won't go further. But for those who want to solve this probably complete, let me just say that once you have one one solution, you can parametrize all solutions using two integer parameters.

(Go to solution, page **??**) □

Exercise 1.3.15. Find all the integer solutions to $3x^2 + 4y^2 = 5z^2$. (Hint: mod 3.) (Go to solution, page **??**) □

Exercise 1.3.16. Find all solutions to

$$y^2 = x^3 + x + 1 \pmod{26}$$

(No this is not a random, idle question. It's actually very important.) (Go to solution, page **??**) □

Exercise 1.3.17. (This "idle" problem is way more important than you think.)

(a) Compute

$$2^n \pmod{26}$$

for $n = 0, 1, 2, 3, \ldots.$ What do you notice? Can you tell me what is

$$2^{1000} \pmod{26}$$

(b) Now try $3^n \pmod{26}$ for $n = 0, 1, 2, 3, ...$ Can you tell me what is $3^{1000} \pmod{26}$
(c) Now try $4^n \pmod{26}$ for $n = 0, 1, 2, 3, ...$ Can you tell me what is $4^{1000} \pmod{26}$
(d) Now try $5^n \pmod{26}$ for $n = 0, 1, 2, 3, ...$ Can you tell me what is $5^{1000} \pmod{26}$
(e) Of course since $\mathbb{Z}/26$ is finite, you would expect $a^n \pmod{26}$ to be finite even though $n = 0, 1, 2, ...$ runs through an infinite set. Which a would give you most distinct value of $a^n \pmod{26}$ as you run through all values for n? And do you notice a pattern in the finite collection of all the $a^n \pmod{26}$?

(Go to solution, page **??**) □

Exercise 1.3.18. Continuing the previous question ... What about $a^n \pmod{p}$ when p is a prime? How many values do you get as you run through $n = 0, 1, 2, ...$? First try it for $a = 2$: try about 20 to 50 primes p. Do you see a pattern? Once you have spotted the pattern, try $a = 3$, etc. (The above is a very serious research problem for you.) (Go to solution, page **??**) □

Exercise 1.3.19.

(a) What is $10 \pmod{3}$? (when simplified).
(b) What is $100 \pmod{3}$? (when simplified).
(c) What is $1000 \pmod{3}$? (when simplified).
(d) What is $5342 \pmod{3}$? (when simplified).
(e) What is $534603142235187 \pmod{3}$? (when simplified).
(f) What is the general fact here?

(Go to solution, page **??**) □

Exercise 1.3.20.

(a) What is $10 \pmod{9}$? (when simplified).
(b) What is $100 \pmod{9}$? (when simplified).
(c) What is $1000 \pmod{9}$? (when simplified).
(d) What is $9142 \pmod{9}$? (when simplified).
(e) What is $96331420351887 \pmod{9}$? (when simplified).
(f) What is the general fact here?

(Go to solution, page **??**) □

Exercise 1.3.21.

(a) What is 10 (mod 11)? (when simplified).
(b) What is 100 (mod 11)? (when simplified).
(c) What is 1000 (mod 11)? (when simplified).
(d) What is 9142 (mod 11)? (when simplified).
(e) What is 80432440556787 (mod 11)? (when simplified).
(f) What is the general fact here?

(Go to solution, page **??**) □

Exercise 1.3.22. Although the algebraic manipulations on mod N seems to be very similar to the algebraic manipulations in $\mathbb{Z}$ (or even $\mathbb{R}$), certain "bizarre" things do happen. In $\mathbb{Z}$ (in fact in $\mathbb{Q}$ and $\mathbb{R}$ as well), you have the implication

$$xy = 0 \implies x = 0 \text{ or } y = 0$$

Is it true that

$$xy \equiv 0 \pmod{26} \implies x \equiv 0 \pmod{26} \text{ or } y \equiv 0 \pmod{26}$$

For each N, check when

$$xy \equiv 0 \pmod{N} \implies x \equiv 0 \pmod{N} \text{ or } y \equiv 0 \pmod{N}$$

holds. And when it does not, give one counterexample. Do you see a pattern?
(Go to solution, page **??**) □

Exercise 1.3.23. (Dr.Liow's magic formula) Suppose I want to simplify 23532 (mod 26). With a calculator or C++ or python, we see quickly that

$$23532 \equiv 2 \pmod{26}$$

I claim that you can use this magic formula: Suppose the digits of a 5–digit number is *edcba*. For instance $edcba = 23532$ where $e = 2, d = 3, c = 5, b = 3, a = 2$. Then

$$edcba \equiv ba + 2(-2c + 6d - 5e) \pmod{26}$$

For instance when $edcba = 23532$, we have

$$edcba \equiv 32 + 2(-2 \cdot 5 + 6 \cdot 3 - 5 \cdot 2) \pmod{26}$$

and

$$32 + 2(-2 \cdot 5 + 6 \cdot 3 - 5 \cdot 2) \equiv 6 + 2(-2) \equiv 2 \pmod{26}$$

which is easier to work with than the much larger 23532. So ... is

$$edcba \equiv ba + 2(-2c + 6d - 5e) \pmod{26}$$

really true for any 5-digit number $edcba$? Write a program to test all cases. Next, write a proof (that does not involve checking all cases like a program).

(Go to solution, page **??**) □

Exercise 1.3.24. Continuing the previous question:

(a) I further claim that if the number is an 8–digit number $hgfedcba$, then

$$hgfedcba \equiv ba + 2((-2c + 6d - 5e) + (2f - 6g + 5h)) \pmod{26}$$

This is also the same as saying

$$hgfedcba \equiv ba + 2((2(f - c) + 6(d - g) + 5(h - e)) \pmod{26}$$

Is this true? Write a program to check. If it's true, prove it.

(b) Can you conjecture a general formula? Can you prove it?

(Go to solution, page **??**) □

Exercise 1.3.25.

(a) Solve the following

$$\begin{aligned} 9x + 5y &\equiv 3 \pmod{26} \\ 5x + 7y &\equiv 1 \pmod{26} \end{aligned}$$

First solving by writing a program that performs a brute force search for solutions. Next, try to solve it algebraically by hand.

(b) What about this one:

$$\begin{aligned} 3x - y &\equiv 2 \pmod{26} \\ 2x + 19y &\equiv 14 \pmod{26} \end{aligned}$$

(c) And this one:

$$\begin{aligned} 9x + y &\equiv 2 \pmod{26} \\ 19x + 5y &\equiv 7 \pmod{26} \end{aligned}$$

(d) Write a program that solves

$$\begin{aligned} ax + b &\equiv c \pmod{26} \\ dx + ey &\equiv f \pmod{26} \end{aligned}$$

for a, b, c, d, e, f in $\mathbb{Z}/26$ by brute force search. Then write a program that randomly picks a, b, c, d, e, f in $\mathbb{Z}/26$ and ask you to solve it. Print all the cases where a, b, c, d, e, f provides a linear system of two equations or two unknowns where there is no solution. Do you notice a pattern in these degenerate cases?

(Go to solution, page **??**) □

Exercise 1.3.26. Notice that earlier on, we wrote down multiplicative inverses of elements in $\mathbb{Z}/26$. Some elements do not have inverses. Can you tell if there's something common among them?

(a) Compute the multiplicative inverses of elements in $\mathbb{Z}/3$.
(b) Compute the multiplicative inverses of elements in $\mathbb{Z}/5$.
(c) Compute the multiplicative inverses of elements in $\mathbb{Z}/7$.
(d) Compute the multiplicative inverses of elements in $\mathbb{Z}/11$.
(e) Do you notice something special about the above cases? How many elements have multiplicative inverse?
(f) Compute the multiplicative inverses of elements in $\mathbb{Z}/6$.
(g) Compute the multiplicative inverses of elements in $\mathbb{Z}/10$.
(h) Compute the multiplicative inverses of elements in $\mathbb{Z}/14$.
(i) Compute the multiplicative inverses of elements in $\mathbb{Z}/15$.
(j) In the above 4 cases is there something special about values which are invertible? Don't see the pattern? In the above, the modulus are all products of two distinct primes.

(Go to solution, page **??**) □

Exercise 1.3.27. Suppose you want to write a random number generator. Say the numbers are to be in the range [0,256). (You can try a bigger range later – don't be too ambitious for now.) Of course use (reasonable) formula.

(a) Go ahead and try this one:

$$h(n) = (2n+5) \pmod{256}$$

Starting with a seed value of 1, the next value is $h(1) = 7$. And the value after that is $h(7) = 19$. And the next is $h(19) = 43$. What are all the possible values you can obtain using this h?

(b) Now of course you do want to have lots of values. For instance you might want to use this hash function to build a hashtable. Or a cryptographic hash (which we have no covered yet). But in any case, you hope that h covers *all* the values in $[0, 255]$. Was the function above a good function?

(c) What if you use a different seed value? What if you start with 2? Or 3?

(d) Try to find a function that generates as many values in $[0, 255]$ as possible. Can you find one that covers the whole range of $[0, 255]$?

(Go to solution, page **??**) □

$(a) 10 \equiv 1 \pmod 3$

$(b) 100 \equiv 1 \pmod 3$

1.3.19 $(c) 1000 \equiv 1 \pmod 3$

$(d) 5342 \equiv 2 \pmod 3$

$(e) 534603142235187 \equiv 1 \pmod 3$

(f) For powers of 10: $10^n \equiv 1 \pmod 3$ for all $n \geq 1$. In general, the sum of digits of a number divided by 3 gives the same remainder as the number itself mod 3.

$(a) 10 \equiv 1 \pmod 9$

$(b) 100 \equiv 1 \pmod 9$

1.3.20 $(c) 1000 \equiv 1 \pmod 9$

$(d) 9142 \equiv 7 \pmod 9$

$(e) 96331420351887 \equiv 1 \pmod 9$

(f) For powers of 10: $10^n \equiv 1 \pmod 9$ for all $n \geq 1$. In general, the sum of digits of a number divided by 9 gives the same remainder as the number itself mod 9.

$(a) 10 \equiv 10 \pmod{11}$

$(b) 100 \equiv 1 \pmod{11}$

1.3.21 $(c) 1000 \equiv 10 \pmod{11}$

$(d) 9142 \equiv 1 \pmod{11}$

$(e) 80432440556787 \equiv 0 \pmod{11}$

(f) For powers of 10: $10^{2k} \equiv 1 \pmod{11}$ and $10^{2k+1} \equiv 10 \pmod{11}$ for integers $k \geq 0$. In general, alternating sum of digits (with appropriate weights) gives the remainder mod 11.

1.3.22 The statement $xy \equiv 0 \pmod{N} \Rightarrow x \equiv 0 \pmod{N}$ or $y \equiv 0 \pmod{N}$ holds true if and only if N is prime.

For non-prime N, we can find counterexamples. For $N = 26$: Let $x = 13$ and $y = 2$. Then $13 \cdot 2 \equiv 0 \pmod{26}$, but neither $13 \equiv 0 \pmod{26}$ nor $2 \equiv 0 \pmod{26}$.

1.3.26 (a-d) In $\mathbb{Z}/p\mathbb{Z}$ where p is prime, all non-zero elements have multiplicative inverses.

(e) When the modulus is prime, all non-zero elements have multiplicative inverses.

(f-i) In $\mathbb{Z}/n\mathbb{Z}$ where n is composite, an element a has a multiplicative inverse if and only if $\gcd(a, n) = 1$, i.e., a is coprime to n.

1.3.27 (a) The function $h(n) = (2n + 5) \bmod 256$ has a cycle of length 128. Starting with seed 1: $1 \rightarrow 7 \rightarrow 19 \rightarrow 43 \rightarrow 91 \rightarrow 187 \rightarrow 123 \rightarrow \ldots$ This function can only generate 128 distinct values (the even numbers + 5).

(b) This is not a good hash function as it only covers half the range.

(c) Different seeds with the same parity will generate the same cycle.

(d) A function of the form $h(n) = (an + b) \bmod 256$ can cover the entire range if and only if $\gcd(a, 256) = 1$, e.g., $h(n) = (3n + 7) \bmod 256$.

1.4 Attacks debug: classical-cipher-attacks.tex

We know that a algorithm consists of two algorithms: the encryption function E_k and the decryption function D_k, both indexed by a secret key k. When Alice wants to send a plaintext message m to Bob, she computes

$$c = E_k(m)$$

and transmits the ciphertext c . Bob then recovers the original message by computing

$$D_k(c) \ = \ D_k\big(E_k(m)\big) \ = \ m.$$

An adversary (we'll call her Eve) observing this exchange might try to learn either:

Ciphertext-only Eve sees one or more ciphertexts

$$c_1, c_2, \dots, c_n$$

and her goal is to learn either m_i or k from them alone.

Known-plaintext Eve obtains pairs

$$(m_1, c_1), (m_2, c_2), \dots, (m_t, c_t),$$

where each $c_i = E_k(m_i)$. Using these she attempts to recover k.

Chosen-plaintext Eve can submit any plaintexts m' and the she will get the correspondings $E_k(m')$. She then tries to infer k (or decrypt other ciphertexts).

Chosen-ciphertext Eve may submit arbitrary ciphertexts c' to decrypt and obtain $D_k(c')$. From these responses she attempts to recover k (or to learn decryptions of other ciphertexts).

Why these distinctions matter. It is important to understanf the different types of attacka and the increase in the strength of the attacks. Later Eve will have a lot new tool that she can use to attack.

1.5 Attacking the Shift Cipher debug: classical-cipher-attacking-shift-cipher.tex

The shift cipher encrypts a letter x by shifting it by a key k (with $0 \leq k \leq 25$) in the alphabet:

$$E_k(x) \equiv (x+k) \bmod 26, \ D_k(x) \qquad \equiv (x-k) \bmod 26.$$

Since there are only 26 possible keys, a brute-force attack tries all decryptions $D_0, \ldots, D_{25}$.

However, frequency analysis offers a more efficient method:

- Compute the frequency of each ciphertext character.
- Assume the most frequent symbol corresponds to "e."
- Derive the candidate key and perform decryption.
- Verify by checking readability or further statistics.

This heuristic can fail if "e" isn't the most common symbol in the sample.

Here are some useful statistics for English:

3-gram Probabilities:

- the: 0.0181 and: 0.0073 ing: 0.0072
- ent: 0.0042 ion: 0.0042 her: 0.0036
- for: 0.0034 tha: 0.0033 nth: 0.0033
- int: 0.0032 ere: 0.0031 tio: 0.0031
- ter: 0.0030 est: 0.0028 ers: 0.0028
- ati: 0.0026 hat: 0.0026 ate: 0.0025
- all: 0.0025 eth: 0.0024 hes: 0.0024
- ver: 0.0024 his: 0.0024 oft: 0.0022
- ith: 0.0021 fth: 0.0021 sth: 0.0021
- oth: 0.0021 res: 0.0021 ont: 0.0020

4-gram Probabilities:

- tion: 0.31 nthe: 0.27 ther: 0.24
- that: 0.21 ofth: 0.19 fthe: 0.19
- thes: 0.18 with: 0.18 inth: 0.17
- atio: 0.17 othe: 0.16 tthe: 0.16
- dthe: 0.15 ingt: 0.15 ethe: 0.15
- sand: 0.14 sthe: 0.14 here: 0.13

- `thec`: 0.13 `ment`: 0.12 `them`: 0.12
- `rthe`: 0.12 `thep`: 0.11 `from`: 0.10
- `this`: 0.10 `ting`: 0.10 `thei`: 0.10
- `ngth`: 0.10 `ions`: 0.10 `andt`: 0.10

- Use scoring functions (e.g. log-likelihood) to rank key candidates quickly.
- Compare ciphertext n-gram distributions against language models.
- Apply hill-climbing or genetic algorithms to search the key space efficiently.
- Combine frequency analysis with n-gram scoring for robust results.

Exercise 1.5.1.

(a) What is $10 \pmod{11}$? (when simplified).
(b) What is $100 \pmod{11}$? (when simplified).
(c) What is $1000 \pmod{11}$? (when simplified).
(d) What is $9142 \pmod{11}$? (when simplified).
(e) What is $80432440556787 \pmod{11}$? (when simplified).
(f) What is the general fact here?

(Go to solution, page **??**) □

Exercise 1.5.2. Although the algebraic manipulations on mod N seems to be very similar to the algebraic manipulations in $\mathbb{Z}$ (or even $\mathbb{R}$), certain "bizarre" things do happen. In $\mathbb{Z}$ (in fact in $\mathbb{Q}$ and $\mathbb{R}$ as well), you have the implication

$$xy = 0 \implies x = 0 \text{ or } y = 0$$

Is it true that

$$xy \equiv 0 \pmod{26} \implies x \equiv 0 \pmod{26} \text{ or } y \equiv 0 \pmod{26}$$

For each N, check when

$$xy \equiv 0 \pmod{N} \implies x \equiv 0 \pmod{N} \text{ or } y \equiv 0 \pmod{N}$$

holds. And when it does not, give one counterexample. Do you see a pattern? (Go to solution, page **??**) □

$(a) 10 \equiv 10 \pmod{11}$

$(b) 100 \equiv 1 \pmod{11}$

1.5.1 $(c) 1000 \equiv 10 \pmod{11}$

$(d) 9142 \equiv 1 \pmod{11}$

$(e) 80432440556787 \equiv 0 \pmod{11}$

(f) For powers of 10: $10^{2k} \equiv 1 \pmod{11}$ and $10^{2k+1} \equiv 10 \pmod{11}$ for integers $k \geq 0$. In general, we can use a digit-by-digit computation where each digit position has a weight based on its place value modulo 11.

1.5.2 The statement $xy \equiv 0 \pmod{N} \Rightarrow x \equiv 0 \pmod{N}$ or $y \equiv 0 \pmod{N}$ holds true if and only if N is prime.

For non-prime N, we can find counterexamples. For $N = 26$: Let $x = 13$ and $y = 2$. Then $13 \cdot 2 \equiv 0 \pmod{26}$, but neither $13 \equiv 0 \pmod{26}$ nor $2 \equiv 0 \pmod{26}$.

The pattern is that in modular arithmetic modulo N, the implication holds if and only if N is prime. For composite N, we can always find factors a and b of N where $a < N$ and $b < N$ such that $ab \equiv 0 \pmod{N}$, but neither $a \equiv 0 \pmod{N}$ nor $b \equiv 0 \pmod{N}$.

1.6 Affine cipher debug: classical-cipher-affine-cipher.tex

Suppose we use $\mathbb{Z}/26$ instead of a to z. For the shift cipher, we have

$$E(k, x) = x + k \bmod 26, \quad D(k, x) = x - k \bmod 26.$$

We can think of these shifts in terms of numbers where the alphabets represent the numbers. This way we can use different mathematical operations on top of these numbers to create formulas and new algorithms for encryption and decryption.

The Affine Cipher looks something like this:

$$E(a, b, x) = ax + b \bmod 26.$$

This increases the key from just k to the pair (a, b), so more effort is required to break the cipher.

Key Features

- Key becomes a pair (a, b) instead of a single shift.
- Security relies on a being coprime with 26.
- b can be any integer modulo 26.

Decryption Function

We want

$$D((a, b), E((a, b), x)) = x \pmod{26}.$$

Assume a decryption of the form

$$D((a, b), y) = c\,y + d \bmod 26.$$

Then

$$c(ax + b) + d \equiv x \pmod{26},$$

which gives two conditions:

$$\begin{aligned} c\,a &\equiv 1 \pmod{26}, \\ c\,b + d &\equiv 0 \pmod{26}. \end{aligned}$$

The first condition holds when $c = a^{-1} \pmod{26}$, which requires $\gcd(a, 26) =$

1. The valid values for a are

$$1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.$$

Once c is known, we solve the second condition for d:

$$d \equiv -c\,b \pmod{26}.$$

Hence the decryption function simplifies to

$$D((a, b), x) = a^{-1}(x - b) \bmod 26.$$

Exercise 1.6.1. Suppose $a = 1$. What c's would make $ca \equiv 1 \pmod{26}$? What if $a = 2$? What about $a = 3$? Etc. (Go to solution, page **??**) □

Exercise 1.6.2. How many integers $0, 1, 2, ..., 25$ have multiplicative inverses mod 26? (Go to solution, page **??**) □

Exercise 1.6.3. What is the size of the key space for the affine cipher? In the worse case how many tries must Eve attempt before breaking an affine cipher? Compare this with the shift cipher. (Go to solution, page **??**) □

Exercise 1.6.4. This is easy: Show that $\phi(26) \cdot 26 = 312$. (Go to solution, page **??**) □

Exercise 1.6.5. Solve the above for a and b. (Go to solution, page **??**) □

Exercise 1.6.6. Now let's abstract the above. Suppose A, B, C, D are numbers and a and b satisfy the equations

$$\begin{aligned} Aa + b &= B \\ Ca + b &= D \end{aligned}$$

Solve for a and b in terms for A, B, C, D. If this can be done, then of course you can write a C++ function to return a and b directly. (Go to solution, page **??**) □

Exercise 1.6.7. But wait! a is not arbitrary! Remember that a must be invertible in mod 26. Recall that this means there is some integer c such that $ac \equiv 1 \pmod{26}$. Therefore it would be helpful if you have a function that will determine if a is invertible mod 26. How would you do that? (Go to solution, page **??**) □

1.6.1 For $ca \equiv 1$ (mod 26): When $a = 1$: $c = 1$ When $a = 2$: $c = 13$ When $a = 3$: $c = 9$ When $a = 4$: No solution (not coprime to 26) When $a = 5$: $c = 21$ Solution exists iff $\gcd(a, 26) = 1$.

1.6.2 Numbers coprime to 26 in $\{0, 1, ..., 25\}$: $\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$ Total: 12 integers have multiplicative inverses mod 26.

1.6.3 Affine cipher key space: 12 choices for a × 26 choices for $b = 312$ possible keys. Worst case for Eve: 312 attempts. Compared to shift cipher's 26 keys, affine cipher key space is 12 times larger.

1.6.4 $\phi(26) = 12$ (count of integers coprime to 26) $\phi(26) \cdot 26 = 12 \cdot 26 = 312$

1.6.6 Given: $Aa + b = B$ $Ca + b = D$
Solution: $a = \frac{D-B}{C-A}$ (when $C \neq A$) $b = \frac{BC-AD}{C-A}$

1.7 Vigenere cipher debug: classical-cipher-vigenere-cipher.tex

The shift cipher and the substitution cipher map each letter using a single alphabet: they are *monoalphabetic.* A polyalphabetic cipher uses multiple alphabets to encrypt.

The Vigenère Cipher is a sequence of shifts determined by a key vector. Given a key vector $\{k_1, k_2, \ldots, k_t\}$, each plaintext letter x_i is shifted by $k_{((i-1) \bmod t)+1}$:

$$c_i = x_i + k_{((i-1) \bmod t)+1} \pmod{26}.$$

Example

Key: $\{4, 5, 6, 7, 8\}$; Plaintext: "HELLO WORLD"

- H shifts by 4
- E shifts by 5
- L shifts by 6
- L shifts by 7
- O shifts by 8
- (space remains unchanged)
- W shifts by 4 (key restarts)
- O shifts by 5
- R shifts by 6
- L shifts by 7
- D shifts by 8

Notice that repeated letters receive different shifts: the two L's in "HELLO" shift by 6 then 7.

Key Properties

- Period t equal to key length.
- Security improves with longer, random keys.
- Encryption and decryption both use the same key sequence.

Exercise 1.7.1. Write a function (in your favorite programming language). If returns a hashtable of frequencies of strings of length 1. (Go to solution, page ??) □

Exercise 1.7.2. Generalize the above: Write a function (in your favorite programming language) that accepts a string s and returns a hashtable of where the key-value pair (k, v) has a string length k for key and the value v is the count of occurrences of k in s. (Go to solution, page **??**) □

Exercise 1.7.3. Now modify the above: Write a function (in your favorite programming language) that accepts a string s and returns a hashtable of where the key-value pair (k, v) has a string length k for key and the value v is a list of index positions in s where the k was found. (Go to solution, page **??**) □

Exercise 1.7.4. Using the above, write a function that computes a probabilistic guess on the length of the key when given a string encrypted using the Vigenere cipher. (Go to solution, page **??**) □

Exercise 1.7.5. Approximate $I(s) = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}$ where $p_i = f_i/n$'s are the probabilities taken from our table of letter probabilities. (Go to solution, page **??**) □

Exercise 1.7.6. Suppose s is a random string, i.e. $f_i/n = 1/26$. Compute $I(s) = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}$.

(Go to solution, page **??**) □

Exercise 1.7.7. Write a function `I` that accepts a string of lowercase letters or a list of numbers 0..25 and computes the I–value as described above. (Go to solution, page **??**) □

Exercise 1.7.8. Using the probabilities from our table of letter probabilities, complete the following table:

Relative shift	M
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

(Go to solution, page **??**) □

Exercise 1.7.9. Here's another example. Say this is the plaintext:

```
This is a test. This tutorial is aimed at getting
familiar with the bare bones of LaTeX. First, ensure
that you have LaTeX installed on your computer (see
Installation for instructions of what you will
need). We will begin with creating the actual source
LaTeX, and then take you through how to feed this
through the LaTeX system to produce quality output,
such as postscript or PDF.
```

We use the key of `fun` to obtain the ciphertext

```
ybvxcffnrxngmcfyogtlvffvxuvryqfntjngnhtkuznfvfljnnuy
brguejvbsyftzyfnrczvwmgjhfzlrybnysbzbnayyfnrccaxnnqf
riiadihwwbrjhyyexyrnhfyuyqugniakienhfylhhnvthftzjmug
dihbcyqhrjxjjqvqfojavsqvybpwynycalnujupyonqmbzlpjfny
ykfhqybrsnnpyltogmlbzaumijyisjyqybvxnuwihlbgmyyfnrcm
lxnrrnbulbiopjkhffvysbznczfzwufmctmgxwenjgtlciz
```

The key `fun` gives us the shifts $(5, 20, 13)$. Let's try to re-discover the shifts.

The (sorted) incidence values I for up to length 6 are

```
Average I-values     keylength
-------------------- ---------
0.06467025914470374  3
0.06262443438914027  6
0.04968484284445197  1
0.04938482570061517  4
0.04908514013749339  5
0.048510313216195575 2
```

Therefore we conjecture that the length is 3. (It shouldn't be surprising that multiples of 3 will also give high I-values, right?) In more details, for the case of testing a keylength of 3, the 3 strings are:

```
yxfxmytfxrfjnknfnygjstfcwjzyyzafcxqidwryxnyqnknyhttm
dbqjjqjsywyljyqzjyfysptmzmyjyxwlmfcxruijfyzzzftxnti

bcnncolfuynnhuflnbuvyznzmhlbsbyncnfiiwjyyhuuiihlnhzu
ichxqfaqbycnuomlfyhbnyolaiiybnibynmnnlokfsnnwmmwjlz

vfrgfgvvvqtgtzvjurebfyrvgfrnbnyranrahbherfygaefhvfjg
hyrjvovvpnaupnbpnkqrnlgbujsqvuhgyrlrbbphvbcfucgegc
```

with respective I-values

```
0.0651056539121
0.0687226346849
0.0601824888371
```

The I–values are computed with

$$\frac{\sum_{i=0}^{25} f(i)f(i-1)}{n(n-1)}$$

where f is the frequency function for the string (for instance $f(0)$ is the frequency of `a`) and n is the length of the string. The average of the above 3 I-values for keylength 3 is 0.0646702591447.

Next we compute the M-values. This means that for each distinct pair of strings (from the 3), let f_1 and f_2 be their frequencies and n, n' be their length, we compute

$$\frac{\sum_{i=0}^{25} f_1(i)f_2((i-g) \bmod 26)}{nn'}$$

with all possible relative shifts $g = 0, 1, 2, ..., 25$. ($f(0)$ means the frequency of `a` in the first string, etc.)

```
M-values        Index of 1st string Index of 2nd string Relative shift g
--------------- ------------------- ------------------- ----------------
0.0498633235932                   1                   2                0
0.0344990102743                   1                   2                1
0.0310114054105                   1                   2                2
```

0.0330851164106	1	2	3
0.0452446036384	1	2	4
0.0342162315016	1	2	5
0.0333678951833	1	2	6
0.0332736355924	1	2	7
0.0331793760015	1	2	8
0.0291262135922	1	2	9
0.0398718069564	1	2	10
0.0707889527759	1	2	11
0.044584786502	1	2	12
0.0329908568197	1	2	13
0.0328023376379	1	2	14
0.0415684795928	1	2	15
0.0334621547742	1	2	16
0.0401545857291	1	2	17
0.0454331228202	1	2	18
0.031576962956	1	2	19
0.0286549156377	1	2	20
0.0325195588651	1	2	21
0.0441134885475	1	2	22
0.0446790460929	1	2	23
0.0419455179565	1	2	24
0.0379866151381	1	2	25
0.0355035217971	1	3	0
0.0272225395012	1	3	1
0.0371216447744	1	3	2
0.0509232819341	1	3	3
0.046925566343	1	3	4
0.0324576432515	1	3	5
0.0341709499334	1	3	6
0.0503521797068	1	3	7
0.0397867885018	1	3	8
0.0293165810013	1	3	9
0.0390253188654	1	3	10
0.0426422996383	1	3	11
0.0329335617742	1	3	12
0.0278888254331	1	3	13
0.0438796877974	1	3	14
0.0348372358652	1	3	15
0.028745478774	1	3	16
0.0399771559109	1	3	17
0.0667237768894	1	3	18
0.0402627070246	1	3	19
0.029602132115	1	3	20
0.0323624595469	1	3	21
0.0442604226156	1	3	22
0.0345516847516	1	3	23
0.0413097277746	1	3	24
0.037216828479	1	3	25
0.0413097277746	2	3	0
0.035313154388	2	3	1
0.0286502950695	2	3	2
0.0434989529792	2	3	3
0.0369312773653	2	3	4
0.0317913573196	2	3	5
0.0440700552065	2	3	6
0.0655815724348	2	3	7
0.0368360936608	2	3	8
0.031505806206	2	3	9
0.0355987055016	2	3	10
0.0369312773653	2	3	11
0.037216828479	2	3	12
0.0395012373882	2	3	13
0.0356938892062	2	3	14

```
0.0318865410242                   2                   3               15
0.0336950314106                   2                   3               16
0.0343613173425                   2                   3               17
 0.055396916048                   2                   3               18
0.0403578907291                   2                   3               19
  0.04016752332                   2                   3               20
0.0357890729107                   2                   3               21
0.0484485056158                   2                   3               22
0.0339805825243                   2                   3               23
0.0326480106606                   2                   3               24
0.0328383780697                   2                   3               25
```

We sort and list 6 rows with the highest M-values:

```
M-values        Index of 1st string Index of 2nd string Relative shift g
--------------- ------------------- ------------------- ----------------
0.0707889527759                   1                   2               11
0.0667237768894                   1                   3               18
0.0655815724348                   2                   3                7
 0.055396916048                   2                   3               18
0.0509232819341                   1                   3                3
0.0503521797068                   1                   3                7
```

For the first row, if the M-value using index 1 and index 2 string (i.e., z_1, z_2) with a relative shift of 11. The computed M–value is 0.0707.... In other words if f_1 and f_2 denote the frequency data of z_1 and z_2 respectively, then the M–value shown in the first row above is

$$\frac{\sum_{i=0}^{25} f(i) f'((i-11) \bmod 26)}{n_1 n_2}$$

where n_1 and n_2 are the lengths of z_1, z_2 respectively and $f_1(0)$ is the frequency of a in z_1, $f_1(1)$ is the frequency of b in z_1, etc. From the top 2 entries of M-values:

```
M-values        Index of 1st string Index of 2nd string Relative shift g
--------------- ------------------- ------------------- ----------------
0.0707889527759                   1                   2               11
0.0667237768894                   1                   3               18
```

we get

$$\begin{aligned} k_1 - k_2 &\equiv 11 \pmod{26} \\ k_1 - k_3 &\equiv 18 \pmod{26} \end{aligned}$$

or

$$\begin{aligned} k_2 &\equiv k_1 - 11 \equiv k_1 + 15 \pmod{26} \\ k_3 &\equiv k_1 - 18 \equiv k_1 + 8 \pmod{26} \end{aligned}$$

At this point, even though we don't have the 3 shifts yet, we at least know that the shifts are probably of the form

$$(k_1, \quad (k_1 + 15) \bmod 26, \quad (k_1 + 8) \bmod 26)$$

There is now *one* unknown. Now trying different values of k_1, we see that when $k_1 = 5$, we get

$$k_1 = 5, \quad k_2 = 20, \quad k_3 = 13$$

which corresponds to the word `fun`. Vóila! ... we have discovered the key.

Note that without any of the above techniques, the key space can be any string, i.e., the size of the key space is infinite. The first step was finding (probabilistically) the length of the key. Once this is found, say the (probable) length of the key is m, then the (probable) size of the key space becomes 26^m. The second step reduces the search down to search for k_1 (basically the search for the first letter of the key) which is in a search space of size 26. (Go to solution, page **??**) □

Exercise 1.7.10. (Go to solution, page **??**) □

Exercise 1.7.11. Let $x = \langle x_1, x_2, ..., x_n \rangle$ $y = \langle y_1, y_2, ..., y_n \rangle$ be two sequences of real numbers. Let

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

and likewise for $\|y\|$. Also, let

$$x \cdot y = \sum_{i=1}^{n} x_i y_i$$

Note that

$$x \cdot x = \|x\|^2$$

(a) Show that

$$|x \cdot y| \leq \|x\| \cdot \|y\|$$

and equality holds iff $y = cx = \langle cx_1, ..., cx_n \rangle$ for some $c \in \mathbb{R}$.

(b) In particular if x is made up of real numbers ≥ 0 which are not all zeros and not all the same and y is a permutation of x, then $x \cdot y$ is maximum iff $y = x$ (i.e., the permutation used to create y is the identity permutation that does not move the values of x).

The above is the justification of the method used in breaking Vigenère's cipher. (Go to solution, page ??) □

1.7.2 Function to count occurrences of substrings of different lengths:

```
function substringCount(s):
    counts = {}
    for length in range(1, len(s)):
        for i in range(len(s) - length + 1):
            substring = s[i:i+length]
            counts[substring] = counts.get(substring, 0) + 1
    return counts
```

1.7.3 Function to find index positions of substrings:

```
function substringPositions(s):
    positions = {}
    for length in range(1, len(s)):
        for i in range(len(s) - length + 1):
            substring = s[i:i+length]
            if substring not in positions:
                positions[substring] = []
            positions[substring].append(i)
    return positions
```

1.7.4 Function to guess Vigenere key length using repeated substring analysis:

```
function guessKeyLength(ciphertext):
    positions = substringPositions(ciphertext)
    distances = []

    # Look for repeated substrings of length 3 or more
    for substring, pos_list in positions.items():
        if len(substring) >= 3 and len(pos_list) > 1:
            # Calculate distances between occurrences
            for i in range(len(pos_list) - 1):
                distances.append(pos_list[i+1] - pos_list[i])

    # Find the GCD of these distances
    if distances:
        return findGCD(distances)
    return 0
```

1.7.5 To approximate Index of Coincidence $I(s) = \frac{\sum_{i=0}^{25} f_i(f_i-1)}{n(n-1)}$:

```
function indexOfCoincidence(text):
    n = len(text)
    freqs = [0] * 26
    for char in text:
        freqs[ord(char) - ord('a')] += 1

    sum_fi_fi_minus_1 = sum(f * (f - 1) for f in freqs)
    return sum_fi_fi_minus_1 / (n * (n - 1))
```

1.7.6 For a random string, each letter has probability $p_i = 1/26$, so: $I(s) = \sum_{i=0}^{25} \frac{f_i(f_i-1)}{n(n-1)} = \sum_{i=0}^{25} \frac{f_i}{n} \cdot \frac{f_i-1}{n-1}$

For large n, $\frac{f_i}{n} \approx \frac{1}{26}$ and $\frac{f_i-1}{n-1} \approx \frac{1}{26}$

Therefore, $I(s) \approx \sum_{i=0}^{25} \frac{1}{26} \cdot \frac{1}{26} = \frac{26}{26^2} = \frac{1}{26} \approx 0.0385$

1.7.7 Function to calculate index of coincidence:

```
function indexOfCoincidence(input):
    n = len(input)
    if n <= 1:
        return 0

    # Handle both string and list input
    if isinstance(input[0], str):
        freqs = [0] * 26
        for char in input:
            freqs[ord(char.lower()) - ord('a')] += 1
    else:  # list of numbers
        freqs = [0] * 26
        for num in input:
            freqs[num] += 1

    sum_fi_fi_minus_1 = sum(f * (f - 1) for f in freqs)
    return sum_fi_fi_minus_1 / (n * (n - 1))
```

1.8 Substitution cipher debug: classical-cipher-substitution-cipher.tex

Deciphering a substitution cipher is a challenging task. Frequency analysis only takes us so far; many letters have similar frequencies, making full decryption difficult.

Key Techniques

- Use frequency analysis to identify likely candidates for ’e’, ’t’, ’a’, etc.
- Apply digram (two-letter) analysis to known letters, treating decryption like solving a crossword puzzle.
- Examine common n-grams (”th”, ”he”, ”in”) and adjust mappings iteratively.
- Refine by re-examining patterns after each correct substitution.

Example Cipher and Approach

```
FMRZVGBYIZBVGCTGDRBBSHJVGCZMVNMVGOFRDCMZVRJF
DMZSBLSNRYABIRZGJNVGCZRBFRBCRBKBJKBCTGDMTZLGV
TVRBFRBCRBBWRBFMNWIBFRBCZAVGJLMFMGBSZCBCWVGMJ
FJBZBSVZWVGCZFJNZBVJWBJNBSAVZWKBCRBAZWIBZMRM
SSNWVGCBTVSYVCBSVGCZFJNZBVJCTGDLNFBTKWVGCZ
```

1. Frequency Analysis Most frequent letters: Z (25), B (22), C (18), V (17), G (15), R (15), F (12). Guess $Z \to e$ and mark all Z as e.

2. Digram Analysis Look for patterns like ”?e” and ”e?”. Assign $C \to t$ if ”te” appears often.

3. Trigram Analysis Identify ”the” patterns: if ”eR” recurs, map $R \to h$, then fill $G \to n$, $B \to a$, etc.

Continue iterating through frequency, digram, and trigram patterns until the plaintext emerges clearly.

1.9 Permutation cipher debug: classical-cipher-permutation-cipher.tex

A permutation cipher reorders the characters of the plaintext according to a fixed permutation, making the ciphertext appear scrambled.

Definition

Given a permutation σ on $\{1, 2, \ldots, n\}$, each plaintext block $(x_1, x_2, \ldots, x_n)$ is mapped to $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$.

Example

Let σ be a permutation of $\{1, 2, 3, 4, 5, 6, 7\}$ defined by:

$$\sigma : \quad 1\,2\,3\,4\,5\,6\,7 \;\rightarrow\; 4\,6\,1\,7\,5\,2\,3\;.$$

In cycle notation:

$$\sigma = (1\,4\,7\,3)(2\,6)(5).$$

Omitting fixed points, this is often written as $\sigma = (1\,4\,7\,3)(2\,6)$.

Key Properties

- The permutation must be invertible (a bijection).
- Encryption and decryption are mutual inverses: apply σ to encrypt, σ^{-1} to decrypt.
- Security depends on block size and secrecy of σ.

Exercise 1.9.1. Of course the process is tedious and error prone. So you know what to do:

- Write a program to encrypt and decrypt messages for the permutation cipher. For instance you can try this permutation

  $$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 5 & 4 & 1 \end{pmatrix}$$

  by entering `2,3,5,4,1`. Note that (mathematically speaking) permutations are written as bijection between sets $\{1, 2, 3, ..., m\}$. You might want to start with 0 instead. In that case, you should tell your user.

- From the above experience of computation-by-hand, we see that for breaking a permutation cipher, the program accepts a ciphertext and ask you for a permutation. Furthermore your program should allow you to enter a permutation partially. For instance to enter this permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ & 3 & & & 1 \end{pmatrix}$$

  the user enters `?,3,?,?,1` in your program. You also want to allow the user to modify a permutation.
- You also want the program to list commonly occurring digrams that appears in the ciphertext, especially including those that appear within substrings of length m (the length of the permutation).
- With all the above, you can then automate the process to imitate what I did in the previous example.

(Go to solution, page **??**) □

Exercise 1.9.2. Decrypt this permutation ciphertext:

```
rhetibbalhotnweerstthigalontakeinuntolfemsorywae
```

and write down the key. It's advisable to do some of these decryption by hand.

(Go to solution, page **??**) □

Exercise 1.9.3. Decrypt this permutation ciphertext:

```
uyashonlpotlsasdaskeathrsshcdp
```

and write down the key. It's advisable to do some of these decryption by hand. (Go to solution, page **??**) □

Exercise 1.9.4. Decrypt this permutation cipher:

```
konusyowairduathsiatirttleistmkewihsihenatprdmpevn
oanigoliorwackatmhfianboemlrteseeudganubtodatofeao
hispaxtinyioenedpipstcheiaatlelairywdhlitsisdntehe
owytmmhaotorlteedmnhieswwanogutyyf
```

and write down the key. It's advisable to do some of these decryption by hand. (Go to solution, page **??**) □

Exercise 1.9.5. Decrypt this permutation ciphertext:

```
wshiattesotbtfeistamiweseostwrhttmsoiefd
```

and write down the key. It's advisable to do some of these decryption by hand. (Go to solution, page **??**) □

Exercise 1.9.6. You have been intercepting ciphertexts sent between Alice and Bob. You know that they use the permutation cipher. You notice this fact: All the 1000 ciphertexts have length 720. Why is that? Or is that just coincidence? (Go to solution, page **??**) □

1.10 Hill cipher debug: classical-cipher-hill-cipher.tex

Hill's cipher Hill cipher, introduced by Lester S. Hill in 1929, represents an important advancement in cryptography by applying linear algebra to encryption. Rather than encrypting letters one at a time, it processes blocks of size n via matrix multiplication.

Encryption Process

For an $n \times n$ key matrix K and plaintext vector $\mathbf{x}$ (converted A=0,...,Z=25),

$$\mathbf{c} = K\mathbf{x} \bmod 26.$$

A 3×3 example: $K = \begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$, $\mathbf{x} = \begin{pmatrix} 2 \\ 17 \\ 24 \end{pmatrix}$ ("CRY").

$$K\mathbf{x} = \begin{pmatrix} 444 \\ 538 \\ 689 \end{pmatrix} \equiv \begin{pmatrix} 2 \\ 18 \\ 13 \end{pmatrix} \bmod 26 = \begin{pmatrix} C \\ S \\ N \end{pmatrix}.$$

Decryption via Matrix Inversion

Requires K^{-1} in mod 26: compute $\det(K)$, adjugate, then multiply by $\det(K)^{-1}$ mod 26. Exists only if $\gcd(\det(K), 26) = 1$.

Key Requirements

- K must be square and invertible mod 26.
- $\det(K)$ must be coprime with 26.

Small Example (2×2)

Key $K = \begin{pmatrix} 9 & 4 \\ 5 & 7 \end{pmatrix}$, "LINEAR"→ numeric blocks [11,8], [13,4], [0,17]. First block:

$$K\begin{pmatrix} 11 \\ 8 \end{pmatrix} = \begin{pmatrix} 131 \\ 111 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 7 \end{pmatrix} \bmod 26 = B, H.$$

Results: "LINEAR"→"BHYCRX".

Advantages and Vulnerabilities

- Obscures single-letter frequencies via block processing.
- Vulnerable to known-plaintext attacks (solve for K with pairs).
- Precursor to modern block ciphers.

1.11 One-time pad cipher debug: classical-cipher-one-time-pad-cipher.tex

One-Time Pad

The one-time pad (OTP) offers perfect secrecy (information-theoretic security) when used correctly. Its key properties are:

- Use a truly random key stream, as long as the plaintext.
- Encrypt by bitwise XOR:

$$C_i = P_i \oplus K_i.$$

- Every possible plaintext of that length is equally likely given the ciphertext.
- An attacker with only the ciphertext gains zero information about the plaintext.
- Security does not rely on computational assumptions—no amount of computing power can break it.

Key Requirements

- The key must be generated from a truly random source.
- The key length must be at least as long as the message.
- Each key bit is used exactly once and never reused.
- The key must remain perfectly secret.

Practical Considerations

- Generating and distributing large, truly random keys is difficult.
- Secure key exchange and storage are the main challenges.
- In practice, OTPs are rare—often replaced by stream ciphers that approximate randomness.

Exercise 1.11.1. Suppose the ciphertext is zsdeasheir. (Of course you have to convert this to bits, say using ASCII code. Find a key so the the above is decrypted to killatfour. Find another key so that the above is decrypted to anapplepie. (Go to solution, page ??) □

There's a rumor that during the Cold War, Washington D.C. communicates with Moscow using one time pad.

(Using the concept of entropy of information theory, Claude Shannon can prove that the ciphertext contains no information about the plaintext, other than the length.)

1.12 Block and stream ciphers debug: classical-cipher-block-and-stream-cipher.tex

In cryptography, we have two primary approaches to encrypting data: stream ciphers and block ciphers. Let's explore these concepts with simple examples.

1.12.1 Stream Ciphers

Stream ciphers process data one unit (typically one byte or character) at a time. Think of them as applying a transformation to each element independently.

Example: Caesar Cipher

The Caesar cipher is a classic stream cipher where each letter is shifted by a fixed number of positions.

With a key of 3:

- Plain: "hello"
- Process: $h \to k$, $e \to h$, $l \to o$, $l \to o$, $o \to r$
- Cipher: "khoor"

Notice how each character is encrypted individually without considering its neighbors. Mathematically, we can express this as:

$$E_K(m_1 \parallel m_2 \parallel \ldots \parallel m_n) = E_K(m_1) \parallel E_K(m_2) \parallel \ldots \parallel E_K(m_n) \tag{1.1}$$

where $\parallel$ represents concatenation.

Example: One-Time Pad

Another stream cipher example is the one-time pad, where each character is combined (usually with XOR) with a random key character:

- Plain: "cat"
- Key: "xyz" (random)
- Process: $c \oplus x$, $a \oplus y$, $t \oplus z$
- Cipher: Result of each XOR operation

1.12.2 Block Ciphers

Block ciphers process fixed-size blocks of data as a single unit. This approach helps mask patterns in the original text.

Example: Hill Cipher

The Hill cipher uses matrix multiplication to encrypt blocks of characters:

With a 2×2 key matrix $K = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix}$ and using $A = 0$, $B = 1$, etc.:

- Plain: "help" → split into blocks: "he" and "lp"
- First block: "he" → $[7, 4]$
- Encrypted first block: $K \times \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 + 12 \\ 7 + 16 \end{pmatrix} = \begin{pmatrix} 26 \\ 23 \end{pmatrix}$ mod $26 = \begin{pmatrix} 0 \\ 23 \end{pmatrix}$
- This translates to "AX"
- Similarly for "lp"
- Cipher: Result after processing all blocks

Example: Modern Block Cipher Concept

Consider a simplified version of a modern block cipher:

- Plain: "The quick brown fox"
- Split into 4-character blocks: "The " + "quic" + "k br" + "own " + "fox"
- Each block undergoes multiple complex transformations
- Cipher: Result of transforming each block

1.12.3 Comparison and Applications

1.12.4 Real-World Context

Modern encryption systems like AES (Advanced Encryption Standard) are block ciphers, processing data in blocks of 128 bits. Stream ciphers like ChaCha20 are still used in scenarios requiring high speed and low resource usage, such as encrypting data on resource-constrained devices.

When implementing either type of cipher in practice, we must consider ad-

Stream Ciphers	**Block Ciphers**
Fast and simple	More complex but offer stronger security
Operate with minimal memory	Require padding for incomplete blocks
Vulnerable to statistical attacks when used improperly	Better at hiding patterns in the data

Table 1.1: Comparison of Stream and Block Ciphers

ditional factors like initialization vectors, padding methods, and operation modes to ensure security.

1.13 Linear feedback shift register debug: classical-cipher-lfsr.tex

LFSRs are efficient mechanisms for generating pseudorandom bit sequences. They operate by using the current state to deterministically calculate the next bit in a sequence.

Basic Concept

1. Start with an initial bit sequence (seed).
2. Apply a linear function (XOR) to selected bits.
3. Shift the register and append the new bit.

Mathematical Foundation

Exclusive-OR (XOR) in modulo 2 arithmetic:

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0.$$

Example

Consider a 4-bit LFSR with initial state $\mathbf{x}^{(0)} = (0, 1, 1, 1)$ and feedback rule:

$$x_{n+4} = x_n \oplus x_{n+2} \pmod 2.$$

Compute:

$$\begin{aligned} x_4 &= x_0 \oplus x_2 = 0 \oplus 1 = 1, \\ x_5 &= x_1 \oplus x_3 = 1 \oplus 1 = 0, \\ x_6 &= x_2 \oplus x_4 = 1 \oplus 1 = 0, \\ x_7 &= x_3 \oplus x_5 = 1 \oplus 0 = 1, \\ x_8 &= x_4 \oplus x_6 = 1 \oplus 0 = 1, \end{aligned}$$

sequence continues similarly. The output stream begins:

$$0111\,1001\,1001\,1\ldots$$

This register has period 7 (less than the maximum $2^4 - 1 = 15$).

Applications in Cryptography

- Generate pseudorandom sequences efficiently with minimal resources.

- Implementable in hardware or software.
- Certain feedback polynomials yield maximal period $2^n - 1$.
- Vulnerable if the structure and output bits are known (solvable via linear algebra).

Exercise 1.13.1. What if you define

$$x_1x_2x_3x_4 = 1011$$

and use

$$x_{n+5} \equiv x_{n+1} + x_{n+2} + x_{n+4} \pmod 2$$

for $n \geq 0$ instead? (Compare with the LSRF above). What sequence do you get? What is the period? (Go to solution, page **??**) □

Exercise 1.13.2. Compute all the possible degree 1 LSRF bit sequences. How many possibilities are there? What is the period for each of them? (Go to solution, page **??**) □

Exercise 1.13.3. Compute all the possible degree 2 LSRF bit sequences. How many possibilities are there? What is the period for each of them? (Go to solution, page **??**) □

Exercise 1.13.4. Compute all the possible degree 3 LSRF bit sequences. How many possibilities are there? What is the period for each of them? (Go to solution, page **??**) □

Exercise 1.13.5. Write a program that generated a sequence of integers with values or 0 and 1 using the LFSR method. You want to have a `c` array as a parameter. Your generator also need to remember bits of the sequence $x_1, ...$ needed to generate the next bits. In general, for a fixed k, you want to have an array c of size k and an array x also of size k. When you call your function, the next bit is placed in the array x, shifting the bits of x so that one bit is lost. So if you want the full sequence for analysis, you need a very long array to keep the bits before it's removed from x. For instance the main program might look like this:

```
x = [x1, x2, x3, x4, x5] # the initial bits
c = [c1, c2, c3, c4, c5] # the coefs of the linear relation
bits = [x1, x2, x3, x4, x5] # the full sequence
```

```
LSRF(c, x) # append rightmost value of x to the right of bits
LSRF(c, x) # append rightmost value of x to the right of bits
etc.
```

Besides putting the new bit into x, it's also a good idea to return that bit as well. Then the above becomes

```
x = [x1, x2, x3, x4, x5] # the initial bits
c = [c1, c2, c3, c4, c5] # the coefs of the linear relation
bits = [x1, x2, x3, x4, x5] # the full sequence
b = LSRF(c, x); append b to the right side of bits
b = LSRF(c, x); append b to the right side of bits
etc.
```

If you like you can also write an LFSR class. Then the above becomes

```
x = [x1, x2, x3, x4, x5] # the initial bits
c = [c1, c2, c3, c4, c5] # the coefs of the linear relation
LFRS = LRFSClass(c, x)
bits = [x1, x2, x3, x4, x5] # the full sequence
b = LSRF.run(); append b to the right side of bits
b = LSRF.run(); append b to the right side of bits
etc.
```

In the above, the code works with integers 0 and 1. For scenarios where there is a huge number of bits, the bits are packed into a register (say of size 64 bits). (Go to solution, page ??) □

Exercise 1.13.6. For a given initial sequence of bits and the sequence c, write a function that attempts to compute the length of the period. Try lots of examples and see if you can produce cases of extremely long periods. (Go to solution, page ??) □

Index

Bibliography