TCS-TR-B-08-04

TCS Technical Report

Course Notes on Complexity and Cryptography

by

Thomas Zeugmann

Division of Computer Science

Report Series B

April 16, 2008

Hokkaido University
Graduate School of
Information Science and Technology

Email: thomas@ist.hokudai.ac.jp

Phone: +81-011-706-7684
Fax: +81-011-706-7684

Abstract

The main purpose of this course is an introductory study of computational complexity and cryptography.

The first part introduces the concept of computational complexity by looking at the basic arithmetic operations, i.e., addition, subtraction, multiplication and division. Then matrix multiplication is touched.

In order to prepare everything we need later for public-key cryptography, we continue with number theoretic problems and study several algorithms including modular exponentiation, primality testing and taking discrete roots.

In the following, we introduce well-known complexity classes, look at complete problems and finish with probabilistic complexity classes. There is also an appendix comprising additional material that had to be omitted due to the introductory character of this course, but may be worth to be known.

The second part is devoted to cryptography. After a short historical sketch we deal with public-key cryptography in some more detail, look at authentication and cryptographic protocols, and finish with a more detailed study of digital signatures. Again, there is an appendix containing material for further reading.

There will be a midterm problem set and a final report problem set each worth 100 points. So your grade will be based on these 200 points.

Note that the course is demanding. But this is just in line with William S. Clark's encouragement

`Boys, be ambitious !`

Of course, nowadays, we would reformulate this encouragement as

`Girls and Boys, be ambitious !`

Recommended Literature

The references given below are *mandatory*.

(1) **丸岡章**：計算理論とオートマトン言語理論、 Information & Computing – 106, サイエンス社, 2005
ISBN 978-4-7819-1104-5

(2) **黒澤馨、尾形わかな、 電子情報通信レクチャーシリーズ**D-8、 **現代暗号の基礎数理、 電子情報通信学会編**

(3) **J。ホップクロフト、J。ウルマン**：オートマトン言語理論　計算論 I, Information & Computing – 3, サイエンス社, 1984
ISBN 4-7819-0374-6

(4) **J。ホップクロフト、J。ウルマン**：オートマトン言語理論　計算論 II, Information & Computing – 3, サイエンス社, 1984
ISBN 4-7819-0432-7

There are some additional references to the literature given in some lectures. So please look there, too.

Contents

Part 1: Complexity

Lecture 1: Introduction

The history of algorithms goes back, approximately, to the origins of mathematics at all. For thousands of years, in most cases, the solution of a mathematical problem had been equivalent to the construction of an algorithm that *solved* it. The ancient development of algorithms culminated in Euclid's famous *Elements*.

Euclid's *Elements* form one of the most beautiful and influential works of science in the history of humankind. Its beauty lies in its logical development of geometry and other branches of mathematics. It has influenced all branches of science but none so much as mathematics and the exact sciences. The Elements have been studied 24 centuries in many languages starting, of course, in the original Greek, then in Arabic, Latin, and many modern languages.

A larger part of Euclid's *Elements* deals with the problem to construct geometrical figures by using only ruler and compass. Over the centuries, often quite different constructions have been proposed for certain problems. Then, in the 19th century we already find a first spirit of *complexity theory* when mathematicians started to compare different algorithms solving the same construction problem by counting the number of applications of ruler and compass.

For having an example, we shall look at the following problem. We wish to construct a triangle $\triangle\,\mathsf{ABC}$ from the following information. Given are the length $\overline{\mathsf{AB}}$ of the side AB, the length of the median going through B, and we also know that the length of the median going through A is twice as large as the length of the median going through C, i.e., $\mathfrak{m}_\mathfrak{a} = 2\mathfrak{m}_\mathfrak{c}$. That is all we know about the triangle $\triangle\,\mathsf{ABC}$.

So, please try to prove or to disprove that $\triangle\,\mathsf{ABC}$ can be constructed by using compass and ruler, only. If you find a construction for the wanted triangle, please count the number of applications of compass and ruler.

In classical geometry there have been also a couple of construction problems around that nobody could solve by using only ruler and compass. The most famous of these problems are the trisection of an angle, squaring the circle and duplicating the cube. Another important example is the question which regular $\mathfrak{n}$-gons are constructible by using only ruler and compass. The latter problem was only resolved by Gauss in 1798.

However, even after Lindemann's proof in 1882 of the impossibility to square the circle, it took roughly another 50 years before modern computability theory started. The main step still to be undertaken was to formalize the notion of algorithm. The famous impossibility results obtained for the classical geometrical problems "only" proved that there is no particular type of algorithm solving them. Here the restriction is to use ruler and compass, only.

But there have been other problems around that could not be solved despite enormous efforts of numerous mathematicians. For example, the design of an algorithm deciding whether or not a given Diophantine equation has an integral solution (Hilbert's

10th problem) remained unsolved until 1967 when it was shown by Matijasevič [7] that there is no such algorithm. So, modern computation theory starts with the question:

Which problems can be solved algorithmically ?

In order to answer it, first of all, the *intuitive notion of an algorithm has to be formalized mathematically.* Starting from different points of views Turing [12], Church [1], and Gödel [2] have given their formalizations (i.e., the Turing machine, the λ–calculus, the recursive functions). However, all these notions are equivalent, i.e., each computation in formalism-1 can be simulated in formalism-2 and vice versa. Since then, many other proposals had been made to fix the notion of an algorithm (e.g., by Post [8] and by Markov [6]). This led to the well-known "Church Thesis," i.e., "*the intuitive computable functions are exactly the Turing machine computable ones.*"

After the theory explaining which problems can and cannot be solved algorithmically had been well-developed, the attention has turned to the qualitative side, i.e., *How "good" a problem can be solved?* First influential papers in this direction were written by Rabin [9] and [10], as well as by Hartmanis and Stearns [4]; the latter paper gave the field its title: *computational complexity.* The central topic studied here was to clarify what does it mean to say that a function f is more difficult to compute than a function g. Alternatively, one could also ask what does it mean that a function f is more difficult to program than a function g.

Of course, this also involves the problem of how to analyze a given algorithm with respect to its complexity. For the time being, we shall deal with the *time complexity* of algorithms. This can be done in a machine-dependent way, i.e., just putting the algorithm into code and compiling it. Then, we may measure the amount of CPU time used in dependence on the input to the algorithm. But this approach has a serious drawback, that is, it is hard to compare different methods solving the same problem by different persons that do not have identical computers.

So, it is better to favor a *machine-independent* approach. Since the basic operation performed by a computer is a bit operation, we may just count the number of bit operations needed.

This course mainly deals with *sequential* computations. That is, within our underlying model, one and only one bit operation can be performed in one time step. So, we identify the time complexity with the number of bit operations to be performed.

For many problems often quite different algorithms are known that solve them. Thus, one has to compare these algorithms in order to make a qualified choice which algorithm to use. This is, however, easier said than done. In general, we distinguish between the *best-case*, *worst-case* and *average-case* analysis of algorithms. As these names suggest, the best-case deals with those inputs on which the algorithm achieves its fastest running time. The worst-case analysis provides an upper bound which is never exceeded independently of the inputs the algorithm is run with. So a worst-case analysis is of particular importance for all applications that are safety critical.

For example, an autopilot must work under all conditions in real-time. On the other hand, it may well be that those inputs, on which the considered algorithm achieves its worst-case, occur very seldom in practice. Provided the application is not safety critical, one should prefer the algorithm achieving the better average-case behavior.

Last but not least, for several applications one also needs guarantees that there is no algorithm solving a problem quickly on any input. We shall study such applications in the second part of our course when dealing with cryptography. But even without going into any detail here, it should be clear that deciphering a *password* must be as difficult on average as in the worst-case.

We aim to express the complexity as a functions of the *size* of the input. This assumes that inputs are made by using any reasonable representation for the data on hand. For doing this, it is advantageous to neglect constant factors. Intuitively, this means that we are aiming at results saying that the running time of an algorithm is proportional to some function.

1.1. Notations and Definitions

For formalizing this approach, we need the following notations and definitions. By $\mathbb{N}$ we denote the set of all natural numbers including zero. Furthermore, we set $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. The set of all integers is denoted by $\mathbb{Z}$. We use $\mathbb{Q}$ and $\mathbb{R}$ for the set of all rational numbers and real numbers, respectively. The non-negative real numbers are denoted by $\mathbb{R}_{\geqslant 0}$. For all real numbers y we define $\lfloor y \rfloor$, the *floor function*, to be the greatest integer less than or equal to y. Similarly, $\lceil y \rceil$ denotes the smallest integer greater than or equal to y, i.e., the *ceiling function*.

Next, we define several notions used to express growth rates of functions.

Definition 1.1. *Let $g\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be any function. We define the following sets*

(1) $O(g(n)) = \{f \mid f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0},$ there exist constants $c, n_0 > 0$ such that $0 \leqslant f(n) \leqslant cg(n)$ for all $n \geqslant n_0\}$,

(2) $\Omega(g(n)) = \{f \mid f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0},$ there exist constants $c, n_0 > 0$ such that $0 \leqslant cg(n) \leqslant f(n)$ for all $n \geqslant n_0\}$,

(3) $o(g(n)) = \{f \mid f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0},$ for every constant $c > 0$ there exists a constant $n_0 > 0$ such that $0 \leqslant f(n) \leqslant cg(n)$ for all $n \geqslant n_0\}$,

(4) $\Theta(g(n)) = \{f \mid f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0},$ there exist constants $c_1, c_2,$ and $n_0 > 0$ such that $0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ for all $n \geqslant n_0\}$.

To indicate that a function f is a member of $O(g(n))$, we write $f(n) = O(g(n))$. We adopt this convention to $\Omega(g(n))$, $o(g(n))$ and $\Theta(g(n))$. Note that the O-notation expresses an ***asymptotic upper bound*** while the Ω-notation expresses an ***asymptotic lower bound***. Looking at the definition above, we see that the Θ-notation establishes an ***asymptotic tight bound***.

The main difference between the O-notation and the o-notation is that for $f(n) = O(g(n))$ the bound $0 \leqslant f(n) \leqslant cg(n)$ holds for *some* constant, but if $f(n) = o(g(n))$, the bound $0 \leqslant f(n) \leqslant cg(n)$ holds for *all* constants $c > 0$. To understand this difference, as well as the relations between O, Ω and Θ the following exercise helps.

Exercise 1. *Let* $f,\ g\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ *be any functions. Then we have*

(1) $f(n) = o(g(n))$ *if and only if* $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.

(2) $f(n) = O(g(n))$ *if and only if* $g(n) = \Omega(f(n))$,

(3) $f(n) = \Theta(g(n))$ *if and only if* $f(n) = \Omega(g(n))$ *and* $f(n) = O(g(n))$.

Throughout this course, we write $\log n$ to denote the logarithm to the base 2, $\ln n$ to denote the logarithm to the base e (where e is the Euler number), and $\log_c n$ to denote the logarithm to the base c.

Now, we are ready to study the first algorithms. The basic arithmetic operations, i.e., *addition, subtraction, multiplication* and *division* are of fundamental importance. Therefore, we start with them.

1.2. Addition

For measuring the complexity of addition, we use the length of the numbers to be added as the basic complexity parameter. Thus, in the following we assume as input two n-bit numbers $a = a_{n-1} \cdots a_0$ and $b = b_{n-1} \cdots b_0$, where $a_i,\ b_i \in \{0,1\}$ for all $i = 0, \ldots, n-1$. The semantics of these numbers a and b is then $a = \sum\limits_{i=0}^{n-1} a_i 2^i$ and $b = \sum\limits_{i=0}^{n-1} b_i 2^i$. We have to compute the sum

$$s = a + b = \sum_{i=0}^{n} s_i 2^i \ , \quad s_i \in \{0,1\} \text{ for all } i = 0, \ldots, n \ .$$

Note that s is a number having at most $n+1$ bits. We use $\wedge$ and $\vee$ we denote the logical *AND* and *OR* operation, respectively. By $\oplus$ we denote the Boolean *EX-OR* function, i.e.,

$\oplus$	0	1
0	0	1
1	1	0

Using essentially the well-known school method for addition, we can express the s_i and the carry bits c_0 and c_{i+1}, $i = 0, \ldots, n-1$ as follows:

$$\begin{aligned} s_n &= c_n \ , \\ s_i &= a_i \oplus b_i \oplus c_i \ , \quad \text{where} \\ c_0 &= 0 \\ c_{i+1} &= (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i) \ . \end{aligned}$$

Since we want to count the number of binary operations to be performed, we have to make a decision here. Either we can consider $\oplus$ also as a basic binary operation or we restrict ourselves to allow exclusively the logical *AND*, *OR* and the logical negation as basic binary operations. In the latter case, we have to express $\oplus$ as

$$x \oplus y = (x \wedge \bar{y}) \vee (\bar{x} \wedge y) \quad x, y \in \{0, 1\} ,$$

i.e., it takes 5 bit basic operations (2 negations, 2 times *AND* and one time *OR*) to express $\oplus$. So, the choice we make will only affect the constant and can thus be neglected here.

Summarizing our results, we thus obtain the following theorem.

Theorem 1.1. *The addition of two numbers* a *and* b *each having at most* n *bits can be performed in time* $O(n)$.

More precisely, if we have to express $\oplus$ by using negations, *AND* and *OR*, then the time complexity of adding two n bit numbers can be bounded by $15n$. If $\oplus$ itself is considered to be a basic binary operation, then we get the bound $7n$.

Subtraction can be handled within the scope of integer addition. For representing integers, we need an extra bit for representing the sign. We shall call such representations *AS*-numbers, where A stands for "absolute value" and S for "sign."

Definition 1.2. *The* ***value*** *of an AS-number* $x = (x_{n-1}, \ldots, x_0) \in \{0, 1\}^n$ *is defined as*

$$V_{AS}(x) = (-1)^{x_{n-1}}(x_{n-2}2^{n-2} + \cdots + x_1 2 + x_0) .$$

When dealing with addition of two AS-numbers x, y having at most n bits each, we distinguish the following cases.

Case 1. $x_{n-1} = y_{n-1}$

That means x and y have the same sign. Hence, the sign of the sum is the same as the common sign of x and y. Moreover, the absolute value of the sum equals the sum of the absolute values of x and y. Thus, we can directly use addition algorithm presented above.

Case 2. $x_{n-1} \neq y_{n-1}$

Now, we have to compare the absolute values of x and y. The sign of the sum is equal to sign of the number having the bigger absolute value. Without loss of generality, let x' be the bigger absolute value and y' the smaller one. Hence, the absolute value of the sum is $x' - y'$.

Consequently, we have to deal with two additional problems. First, we have to show that subtraction of two n bit numbers can be performed in time $O(n)$. Second, we have to prove that comparison of absolute values of two n bit numbers can be done in time $O(n)$, too. We leave it as an exercise to show these two results, since you

should have already some familiarity with these subjects (from technical computer science and/or electrical engineering).

Exercise 2. *The subtraction of two AS-numbers* a *and* b *each having at most* n *bits can be performed in time* $O(n)$.

Next, we turn our attention to multiplication. For the sake of presentation, we deal here only with the multiplication of natural numbers.

1.3. Multiplication

In the following we assume as input two n-bit numbers $a = a_{n-1} \cdots a_0$ and $b = b_{n-1} \cdots b_0$, where $a_i,\ b_i \in \{0,1\}$ for all $i = 0, \ldots, n-1$. Again, the semantics of these numbers a and b is then $a = \sum_{i=0}^{n-1} a_i 2^i$ and $b = \sum_{i=0}^{n-1} b_i 2^i$.

We have to compute the product ab. Taking into account that $a,\ b < 2^n$, one easily estimates $ab < 2^n \cdot 2^n = 2^{n+n} = 2^{2n}$. So, the product ab has at most $2n$ bits.

First, we apply the usual school method for multiplication. Multiplication of two bits can be realized by the *AND* function. Thus, first we need n^2 many applications of *AND* to form the n summands and n costless shifts. Next, we have to add these n numbers. Using the same ideas as above, one easily verifies that this iterated sum takes another $O(n^2)$ many bit operations. Thus we have the following theorem.

Theorem 1.2. *The usual school algorithm for multiplying two numbers* a *and* b *each having at most* n *bits can be performed in time* $O(n^2)$.

Next, we ask whether or not we can do, at least asymptotically, better. The affirmative answer will be provided by our next theorem which goes back to Karatsuba and Ofman [5]. For establishing it, we shall apply the method of *divide et impera* (*divide and conquer* in English) for sequential computations. The general idea of the method *divide et impera* is as follows. The problem of size n is divided into a certain number of independent subproblems having of the same type but having lower size. The solution of the original problem is obtained combining the solutions of the subproblems in an appropriate manner. If this technique is applied recursively to the subproblems until problems of sufficiently small size arise, then the best effect will result. However, the number of subproblems obtained in each recursive step is also crucial and requires often tricky constructions.

Theorem 1.3. *There is an algorithm for multiplying two numbers* a *and* b *each having at most* n *bits that can be performed in time* $O(n^{\log 3})$.

Proof. Without loss of generality we let $n = 2^k$. Then there exist numbers $a_0,\ a_1$ and $b_0,\ b_1$ such that

$$a = a_0 2^{n/2} + a_1 \quad \text{and } b = b_0 2^{n/2} + b_1\ .$$

Then, it holds:

$$\begin{aligned} ab &= (a_0 2^{n/2} + a_1)(b_0 2^{n/2} + b_1) \\ &= a_0 b_0 2^n + (a_0 b_1 + a_1 b_0) 2^{n/2} + a_1 b_1 \end{aligned}$$

But a direct application of this reduction to the problem of multiplying two n bit numbers to *four* multiplications of numbers having at most $n/2$ bits and three additions as well as two shift operations does not help. Fortunately, Karatsuba and Ofman [5] have discovered a way to reduce the number of problems having half of the size of the original problem from four to *three* by observing that

$$(a_0b_1 + a_1b_0) = (a_0 + a_1)(b_0 + b_1) - (a_0b_0 + a_1b_1) \ .$$

Using the latter equation, it is immediately clear that we only have to compute the three products a_0b_0, a_1b_1, and $(a_0 + a_1)(b_0 + b_1)$.

Therefore, we directly arrive at the following algorithm. For each of the steps displayed below perform the computation sequentially.

Improved Multiplication

(1) $s_0 = a_0 + a_1$, $s_1 = b_0 + b_1$

(2) $p_0 = a_0b_0$, $p_1 = a_1b_1$, $p_2 = s_0s_1$

(3) $t = p_0 + p_1$

(4) $u = p_2 - t$, $\hat{u} = u2^{n/2}$

(5) $v = a_0b_02^n + a_1b_1$

(6) $p = v + \hat{u}$

The correctness of the algorithm above is obvious by the arguments provided before displaying it.

Next, we estimate the complexity of our improved multiplication.

Let $A(n)$ be the time for adding two n bit numbers, and

let $M(n)$ be the time for multiplying two n bit numbers.

Then, we can estimate the time complexity as follows:

(1) $2A(n/2)$

(2) $2M(n/2) + M(n/2 + 1)$

(3) $A(n)$

(4) $A(n + 2)$ and a costless shift.

(5) This step is costless, since it is only a concatenation of bits.

(6) $A\left(\frac{3}{2}n\right)$, since the $n/2$ lower bits of $\hat{u}$ are all 0.

Now, the main idea is to apply our improved multiplication algorithm recursively to itself. The only disturbing point here is that we have a subproblem (computing $s_0 s_1$) which is the multiplication of two numbers having $n/2+1$ bits (and not only $n/2$ as desired). We resolve it as follows.

Let $s_0 = x_0 2 + x_1$ and $s_1 = y_0 2 + y_1$, where $x_1,\ y_1 \in \{0,1\}$ and $x_0,\ y_0$ are $n/2$ bit numbers. Then

$$\begin{aligned} s_0 s_1 &= (x_0 2 + x_1)(y_0 2 + y_1) \\ &= 4x_0 y_0 + 2(x_0 y_1 + x_1 y_0) + x_1 y_1 \end{aligned}$$

Now, the multiplication of $x_0 y_0$ is a multiplication of $n/2$ bits numbers, i.e., it has costs $M(n/2)$. The remaining multiplications, i.e., $x_0 y_1$, $x_1 y_0$ and $x_1 y_1$ can be directly realized by using $n+1$ AND gates, since $x_1,\ y_1 \in \{0,1\}$. Additionally, we have to include the costs for addition, i.e., $A(n) + A(n/2) + 1$. Therefore, we arrive at

$$M(n/2+1) \leqslant M(n/2) + A(n) + A(n/2) + 1 \ .$$

Since addition can be realized by a sequential algorithm taking time $O(n)$, we can conclude that there is a constant $\hat{c} > 0$ such that

$$M(n/2+1) \leqslant M(n/2) + \hat{c}n \ .$$

Consequently, there is a constant $c > 0$ such that

$$M(n) = \begin{cases} c & , \quad \text{if } n = 1 \\ 3M(n/2) + cn \ , & \quad \text{if } n > 1 \ . \end{cases}$$

Now, it suffices to show that $M(n) = 3cn^{\log 3} - 2cn$ is a solution of the recursive equation displayed above. This is shown inductively. For the induction base, we directly get

$$T(1) = 3c1^{\log 3} - 2c = c \ .$$

Assuming the induction hypothesis for m, we perform the induction step from m to $2m$ as follows. Recall that $n = 2^k$, thus this induction step is justified. We have to show that $M(2m) = 3c(2m)^{\log 3} - 2c(2m)$.

$$\begin{aligned} M(2m) &= 3M(m) + 2cm \\ &= 3\left(3cm^{\log 3} - 2cm\right) + 2cm \\ &= 9cm^{\log 3} - 6cm + 2cm \\ &= 9cm^{\log 3} - 4cm \\ &= 9cm^{\log 3} - 2c(2m) \\ &= 3c2^{\log 3} m^{\log 3} - 2c(2m) \\ &= 3c(2m)^{\log 3} - 2c(2m) \ . \end{aligned}$$

Consequently, $M(n) = O(n^{\log 3})$. This proves the theorem. ∎

Recall that $\log 3 \approx 1.59$. Thus, the Karatsuba/Ofman [5] algorithm is indeed much faster than the usual school method. Intuitively, the improvement is due to the fact that multiplication is more complex than addition. Hence, performing three instead of four multiplications results in roughly 25% saving of time.

Please think about possible implementations of the Karatsuba/Ofman [5] algorithm including the problem that n is not a power of 2.

Moreover, it should be noted that one can do even better. In 1971, Schönhage and Strassen [11] have found a multiplication algorithm that takes time $O(n \log n \log\log n)$ for computing the product of two n bit numbers. Very roughly speaking, their algorithm works via the fast Fourier transformation. However, their algorithm is only asymptotically faster, and n must be very large for achieving an improvement in practice. We therefore omit this algorithm here.

Since the technique of *divide et impera* is very important for the design of efficient algorithms, it is only natural to ask if we can say something about the general solvability of recursive equations arising in the analysis of the algorithms obtained. The affirmative answer is provided by the following theorem.

Theorem 1.4. *Let a, b and c be positive natural numbers. Then the recursive equation*

$$T(n) = \begin{cases} b\,, & \text{if } n = 1 \\ aT\left(\frac{n}{c}\right) + bn\,, & \text{for all } n > 1\,, \end{cases}$$

where n is a power of c, has the following solution

$$T(n) = \begin{cases} O(n)\,, & \text{if } a < c \\ O(n \log n)\,, & \text{for all } a = c \\ O(n^{\log_c a})\,, & \text{for all } a > c\,. \end{cases}$$

Proof. Let n be a power of c, i.e., $n = c^k$. First, we show that

$$T(n) = bn \cdot \sum_{i=0}^{\log_c n} \left(\frac{a}{c}\right)^i$$

is a solution of the recursive equation given in the theorem.

This is done inductively. For the induction basis let $n = 1$. Then, $\log_c 1 = 0$ and

$$\begin{aligned} T(1) &= b && \text{by definition} \\ &= b \cdot \left(\frac{a}{c}\right)^0 && \text{multiplying by 1} \\ &= b \cdot \sum_{i=0}^{0} \left(\frac{a}{c}\right)^0\,. \end{aligned}$$

This shows the induction basis.

Next, assume the induction hypothesis (abbr. IH) that

$$T(m) = bm \cdot \sum_{i=0}^{\log_c m} \left(\frac{a}{c}\right)^i$$

is a solution of the recursive equation for $n = m$. The induction step has to be done from m to cm, i.e., we have to show that

$$T(cm) = bcm \cdot \sum_{i=0}^{\log_c(cm)} \left(\frac{a}{c}\right)^i .$$

This is done as follows.

$$\begin{aligned} T(cm) &= aT(m) + bcm \quad \text{by definition} \\ &= abm \cdot \sum_{i=0}^{\log_c m} \left(\frac{a}{c}\right)^i + bcm \quad \text{by the IH} \\ &= bm \cdot \sum_{i=0}^{\log_c m} \frac{a^{i+1}}{c^i} + bcm = bcm \cdot \sum_{i=0}^{\log_c m} \left(\frac{a}{c}\right)^{i+1} + bcm \\ &= bcm \cdot \sum_{i=0}^{(\log_c m)+1} \left(\frac{a}{c}\right)^i . \end{aligned}$$

Furthermore, $1 = \log_c c$ since $c^1 = c$. Thus, $(\log_c m)+1 = \log_c m + \log_c c = \log_c(cm)$ (product law for logarithms). Thus, we finally have

$$T(cm) = bcm \cdot \sum_{i=0}^{\log_c(cm)} \left(\frac{a}{c}\right)^i ,$$

and the induction step is shown.

Next, we distinguish the following cases.

Case 1. $a < c$

Then $a/c < 1$ and thus $\sum_{i=0}^{\infty} \left(\frac{a}{c}\right)^i$ is a convergent series. Consequently, $T(n) = O(n)$.

Case 2. $a = c$

Then $a/c = 1$, and thus

$$\sum_{i=0}^{\log_c n} \left(\frac{a}{c}\right)^i = \log_c n .$$

Consequently, $T(n) = O(n \log_c n) = O(n \log n)$.

Case 3. $a > c$

Now, we need the following from calculus. First recall that

$$\sum_{i=0}^{m} x^i = \frac{x^{m+1} - 1}{x - 1} .$$

Thus, we obtain

$$T(n) = bn \cdot \sum_{i=0}^{\log_c n} \left(\frac{a}{c}\right)^i = bn \cdot \frac{\left(\frac{a}{c}\right)^{1+\log_c n} - 1}{\frac{a}{c} - 1}$$

$$
\begin{aligned}
&\leqslant \hat{k}bn \cdot \left(\left(\frac{a}{c}\right)^{1+\log_c n} - 1 \right) \quad \text{note that } \hat{k} \text{ is a constant} \\
&\leqslant \hat{k}bn \cdot \frac{a^{1+\log_c n}}{c^{1+\log_c n}} \\
&= \hat{k}b\frac{a}{c}a^{\log_c n} \quad \text{recall that } n = c^{\log_c n} \\
&= k'a^{\log_c n} \quad \text{where } k' = \hat{k}ba/c \\
&= k'e^{\log_c n \ln a} \\
&= k'e^{\ln n \ln a \cdot (1/\ln c)} \quad \text{since } \log_c n = \frac{\ln n}{\ln c} \\
&= k'e^{\ln n \log_c a} \quad \text{since } \frac{\ln a}{\ln c} = \log_c a \\
&= k'n^{\log_c a} = O(n^{\log_c a}) \; .
\end{aligned}
$$

This proves the theorem. ∎

Exercise 3. *Generalize Theorem 1.4 to the case where the recursive equation is given as*

$$T(n) = \begin{cases} b \; , & \text{if } n = 1 \\ aT\left(\frac{n}{c}\right) + f(n) \; , & \text{for all } n > 1 \; , \end{cases}$$

where n *is again a power of* c*, and function* $f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ *describes the costs for combining the results of the subproblems.*

Throughout this course, we shall see several applications of the *divide et impera* technique, and we shall discuss further issues at the appropriate places.

References

[1] A. CHURCH (1936), An unresolvable problem of elementary number theory, *Am. J. Math.* **58**, 345 – 365.

[2] K. GÖDEL (1931), Über formal unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme, *Monatshefte Mathematik Physik* **38**, 173 – 198.

[3] R.L. GRAHAM, D.E. KNUTH AND O. PATASHNIK (1989), *Concrete Mathematics* (Addison-Wesley, Reading, Massachusetts).

[4] J. HARTMANIS AND R.E. STEARNS (1965), On the computational complexity of algorithms, *Trans. Am. Math. Soc.* **117**, 285 – 306.

[5] A. KARATSUBA AND YU. OFMAN (1962), Multiplication of multidigit numbers on automata, *Doklady Akademii Nauk* **145**, 293 – 294.

[6] A.A. MARKOV (1954), Theoria Algorithmov, *Akad. Nauk SSSR, Math. Inst. Trudy* **42**.

[7] YU. V. MATIJASEVIČ (1970), The Diophantineness of enumerable sets. Dokl. Akad. Nauk SSSR, **191**, 279 – 282.

[8] E. POST (1943), Formal Reductions of General Combinatorial Decision Problems, *Am. J. Math.* **65**, 197 – 215.

[9] M.O. RABIN (1959), Speed of computation and classification of recursive sets, *in* Third Convention Sci. Soc., Israel, pp. 1 – 2.

[10] M.O. RABIN (1960), Degrees of difficulty of computing a function and a partial ordering of recursive sets, *in* Technical Report No. 1, University of Jerusalem.

[11] A. SCHÖNHAGE AND V. STRASSEN (1971), Schnelle Multiplikation großer Zahlen, *Computing* **7**, 281 – 292.

[12] A.M. TURING (1936/37), On computable numbers with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* **42**, 230 – 265.

Lecture 2: Complexity of Division and Matrix Multiplication

In this lecture we study the complexity of division, compare the complexity of division to the complexity of multiplication and then we shall look at the complexity of matrix multiplication.

2.1. Division

Next, we shall deal with division. So, let natural numbers a and d be given. Actually, there are two versions of division, i.e., computing the quotient and computing integers q, r such that $a = qd + r$ and $0 \leqslant r < d$, respectively. While the second version (division with remainder) nicely fits into the class of problems studied so far (n bit numbers given as input are transformed into output numbers having $O(n^c)$ bits for a constant $c > 0$ independently of n), the first version still does not, since the quotient may have infinitely many bits. Thus, in this case, it is only meaningful to require the computation of a sufficiently precise approximation. Therefore, we first define what is meant by approximation.

Definition 2.1. *Let x be any number. We say that $\tilde{x}$ is an **approximation of** x **with precision** 2^{-c} (precise for c bits) provided $|x - \tilde{x}| \leqslant 2^{-c}$.*

Then, the division problem is defined as follows.

Division

Input: Numbers $a,\ d \in \mathbb{N}$ each having at most n bits.

Problem: Compute the quotient a/d with precision 2^{-n}.

When dealing with division, the first observation is that we can split the problem into two subproblems, i.e., computing the inverse d^{-1} of d with precision 2^{-2n} and then multiplying this approximation of d^{-1} and a. In the following, we use $\tilde{d}^{-1}$ to denote the approximation of d^{-1} with precision 2^{-2n}. The justification for the observation made is provided by the following lemma.

Lemma 2.1. *Let $a,\ d$ be n bit numbers. Suppose we have computed the inverse d^{-1} of d with precision 2^{-2n}. Then $a\tilde{d}^{-1}$ is an approximation of a/d with precision 2^{-n}.*

Proof. Let $\tilde{d}^{-1}$ be the approximation of d^{-1} with precision 2^{-2n}. By assumption, $a < 2^n$, and thus

$$\begin{aligned} \left| \frac{a}{d} - a\tilde{d}^{-1} \right| &= |a||d^{-1} - \tilde{d}^{-1}| \\ &\leqslant 2^n \cdot 2^{-2n} = 2^{-n} \,. \end{aligned}$$

∎

First, we deal with the computation of d^{-1}. One possible approach would be to use the fact

$$\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i$$

provided $|x| < 1$.

Note that we can assume, without loss of generality, that $1/2 \leqslant d < 1$. For seeing this, let $d = \sum_{i=0}^{n-1} d_i 2^i$ and let d_k the highest non-zero bit, i.e., $d_k \neq 0$, $k \leqslant n-1$ and $d_{k+1} = \cdots = d_{n-1} = 0$ provided $k < n-1$. Then, $\hat{d} =_{df} d2^{-(k+1)}$ satisfies $1/2 \leqslant \hat{d} < 1$. Thus, $\hat{d}$ can be computed from d by applying a simple shift operation. However, for determining k one needs $O(n)$ bit operations in the worst case. Now, since the inverse of $2^{-(k+1)}$ is 2^{k+1} knowing the inverse of $\hat{d}$ with precision 2^{-2n} is all we need. So, we can set $x = 1 - d$.

However, it remains to ask how many summands we actually have to compute to achieve the desired approximation. The answer is provided by the following lemma.

Lemma 2.2. *Let d^{-1} be the exact inverse of d where $1/2 \leqslant d < 1$. Furthermore, let $b_{2n+1} = \sum_{i=0}^{2n+1} (1-d)^i$. Then, we have $|d^{-1} - b_{2n+1}| \leqslant 2^{-2n}$.*

Proof. First, since $1/2 \leqslant d < 1$ we can conclude that $0 < 1 - d \leqslant 1/2$. Thus, the geometrical series $\sum_{i=0}^{\infty} (1-d)^i$ is absolutely convergent and from calculus we know that

$$\sum_{i=0}^{\infty} (1-d)^i = \frac{1}{1-(1-d)} = \frac{1}{d} \; .$$

Thus, $\sum_{i=0}^{\infty} (1-d)^i$ is the *exact* inverse of d. Hence, we get

$$\begin{aligned}
|d^{-1} - b_{2n+1}| &= \left| \sum_{i=0}^{\infty} (1-d)^i - \sum_{i=0}^{2n+1} (1-d)^i \right| \\
&= \left| \sum_{i=2n+2}^{\infty} (1-d)^i \right| \\
&\leqslant \sum_{i=2n+2}^{\infty} |(1-d)^i| \leqslant \sum_{i=2n+2}^{\infty} \left(\frac{1}{2}\right)^i \\
&= \frac{1 - 1 + \left(\frac{1}{2}\right)^{2n+2}}{1 - \frac{1}{2}} = 2^{-(2n+1)} \\
&< 2^{-2n} \; .
\end{aligned}$$

∎

However, the resulting algorithm is quite slow compared to multiplication. Can we do any better?

The affirmative answer is obtained as follows. We can avoid to compute b_{2n+1} by evaluating the sum given above if we use the well-known Newton procedure for computing zeros of functions. Recall that for a given differentiable function f and x_* with $f(x*) = 0$, one can compute x_* with any desired precision by using

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} \ ,$$

provided x_0 is appropriately chosen. That is, then we know from calculus that

$$\lim_{k\to\infty} x_k = x_* \ .$$

In order to apply this method to our setting of computing the inverse, we *cannot* use $f(x) = xd - 1$. In this case, we would get $f'(x) = d$, and thus for computing the iteration we already should know d^{-1}.

So, we set $f(x) = d - 1/x$, and obtain thus that the sequence $(b_k)_{k\in\mathbb{N}}$ defined by

$$b_k = (2 - b_{k-1}d)b_{k-1}$$

converges to d^{-1} provided b_0 is appropriately chosen. Also, we should know that Newton's procedure converges quadratically. Thus, $O(\log n)$ iterations will suffice. For the sake of completeness, and for justifying our choice of b_0 we include the following theorem here.

Theorem 2.3. *Let d be such that $1/2 \leqslant d < 1$, let $b_0 = 1$ and let $b_k = (2 - b_{k-1}d)b_{k-1}$ for all $k \geqslant 1$. Then we have*

$$b_k = \sum_{i=0}^{2^k-1} (1-d)^i \ .$$

Proof. We prove the theorem by induction. For the induction base $k = 1$ we directly obtain

$$b_1 = (2 - b_0d)b_0 = 1 + (1 - d) = \sum_{i=0}^{2^1-1} (1-d)^i \ .$$

The induction step from k to $k+1$ is derived as follows.

$$\begin{aligned} b_{k+1} &= (2 - b_kd)b_k = 2b_k - b_kdb_k \\ &= 2b_k - \sum_{i=0}^{2^k-1} (1-d)^i \cdot d \cdot \sum_{i=0}^{2^k-1} (1-d)^i \quad \text{by the induction hypothesis} \\ &= 2b_k + \sum_{i=0}^{2^k-1} (1-d)^i(-1 + (1-d)) \sum_{i=0}^{2^k-1} (1-d)^i \end{aligned}$$

$$
\begin{aligned}
&= 2b_k - \sum_{i=0}^{2^k-1}(1-d)^i \sum_{i=0}^{2^k-1}(1-d)^i + \sum_{i=0}^{2^k-1}(1-d)^{i+1}\sum_{i=0}^{2^k-1}(1-d)^i \\
&= 2b_k - b_k - \sum_{i=1}^{2^k-1}(1-d)^i \sum_{i=0}^{2^k-1}(1-d)^i + \sum_{i=1}^{2^k-1}(1-d)^i \sum_{i=0}^{2^k-1}(1-d)^i \\
&\quad + (1-d)^{2^k}\sum_{i=0}^{2^k-1}(1-d)^i \\
&= b_k + \sum_{i=0}^{2^k-1}(1-d)^{2^k+i} \\
&= \sum_{i=0}^{2^k-1}(1-d)^i + \sum_{i=2^k}^{2^{k+1}-1}(1-d)^i \quad \text{by the induction hypothesis} \\
&= \sum_{i=0}^{2^{k+1}-1}(1-d)^i \,.
\end{aligned}
$$

∎

The latter theorem essentially shows that the number of correct bits is doubled in each iteration. Summarizing the results obtained so far, now we can prove the following theorem.

Theorem 2.4. *There is an algorithm which, on input two number* a *and* d *having at most* n *bits, computes the quotient* a/d *with precision* 2^{-n} *using time* $O(M(n)\log n)$.

Proof. First, using Theorem 2.3 we can compute the inverse d^{-1} of d with precision 2^{-2n} by using $\lceil \log 2n \rceil + 1$ many iterations. In each iteration we have to perform two multiplications and one addition. The two multiplications require time $O(M(n))$ and the addition needs time $O(n)$. Of course, we have to truncate the result of each iteration to the $2n$ leading bits.

Furthermore, by Lemma 2.1 it then suffices to multiply a and the approximate inverse. This requires another multiplication of two numbers having at most $2n$ bits. Thus, the overall time complexity is $O(M(n)\log n)$. ∎

Does this mean that division is more complex than multiplication? We may be tempted to answer this question affirmatively, but some care has to be taken here. Of course, we can try to prove a lower bound on the number of bit operations needed to perform division. Provided we could show this lower bound to be $\Omega(M(n)\log n)$, we are done in the sense that we then know no improvement is possible. But this is much easier said than done. Proving non-trivial lower bounds is a very hard task (and we still have almost no experience in doing so). Second, we could try to find another

method for division. But again, this is easier said than done. Nevertheless, you are encouraged to try it.

Finally, it is well possible that we have already collected all good ideas needed, but failed to put them together in the right way. Maybe, we have been a bit to generous. The point here is that we always perform multiplications and additions of $2n$ bit numbers. So, let us continue the lecture by asking what happens if we ignore the bits that are anyhow not correct. There is some hope that this would not matter much, since the Newton procedure is known to be self-correcting. We give it a try. The following table shows the result of our iteration when performed on $2n$ bit numbers (left) and with iterates that use 2, 4, 8 bits and so on (right). In this example, we set $d = 0.66$ and assume that we need the first 8 digits to be correct. Thus, we have to compute 1.5151515 (we perform the calculation here in decimal notion).

x_1	1.34	1.3
x_2	1.494904	1.484
x_3	1.5148809	1.5145110
x_4	1.5151514	1.5151512
x_5	1.5151515	1.5151515

This looks pretty good. Finally, we shall resolve the complexity of division (relative to multiplication).

2.2. Comparing the Complexity of Division and Multiplication

So far, we have shown that for $b_0 = 1$ the sequence $(b_k)_{k\in\mathbb{N}}$ defined by the iteration

$$b_{k+1} = (2 - b_k d)b_k = 2b_k - b_k^2 d \text{ for all } k \in \mathbb{N} \tag{2.1}$$

converges to the inverse of d. Moreover, we have seen that we have to compute only the first $\lceil \log 2n \rceil + 1$ members of the sequence $(b_k)_{k\in\mathbb{N}}$ for obtaining d^{-1} with precision 2^{-2n}. However, if all computations are performed with $O(n)$ bits then we arrive at complexity $O(M(n) \log n)$ for computing the approximative inverse of d.

Let $I(n)$ denote the time needed to compute the inverse of any n bit number with precision 2^{-2n}. Our goal is to show that there exists a constant $c > 0$ such that $I(n) \leqslant M(n)$ for all $n \in \mathbb{N}$. For showing this, we first observe that we can make the following reasonable assumption

$$a^2 M(n) \geqslant M(an) \geqslant aM(n) \text{ for all } a \geqslant 1 \ . \tag{2.2}$$

Now, perform the computation of the b_k's defined by (2.1) as follows. We take the highest k bits of $2b_k$ right from the dual point and calculate $b_k^2 d$ up to the $2k$ bits right from the dual point (by using only the highest k bits of d right from the dual point and setting the remaining bits to zero). Finally, we calculate b_{k+1} up to the highest $2k$ bits right from the dual point.

Of course, now one has to show that the sequence $(b_k)_{k\in\mathbb{N}}$ computed in the way described above still converges to the inverse of d and that we still have to compute only the first $\lceil \log 2n \rceil + 1$ members of it. This is left as an exercise. However, since the Newton method is self-correcting as we know from numerics, this should be intuitively clear. Moreover, we leave all further technical details out here, since we are only interested in the general insight that for sequential computations division and multiplication have the same complexity up to some constant.

Then we arrive at the following theorem.

Theorem 2.5. *There exists a constant* $c > 0$ *such that* $I(n) \leqslant c \cdot M(n)$ *for all* $n \in \mathbb{N}$.

Proof. Performing the calculation of b_{k+1} as described above directly yields

$$I(2k) \leqslant I(k) + \frac{5}{2}M(2k) + c_1 2k$$

for a suitably chosen constant c_1.

Next, we choose $c > 0$ such that

$$c \geqslant I(1)/M(1) \text{ and } c \geqslant 5 + 2c_1 \ .$$

Now, the proof is done by induction on k. For the induction base $k = 1$ everything is obvious. Next, assuming the induction hypothesis $I(k) \leqslant c \cdot M(k)$ we perform the induction step as follows, where in line three below, (2.2) is used, i.e., $M(2k) \geqslant 2M(k)$ and thus $M(k) \leqslant \frac{1}{2}M(2k)$. Moreover, in line four below we use the obvious inequality $2k \leqslant M(2k)$.

$$\begin{aligned}
I(2k) &\leqslant I(k) + \frac{5}{2}M(2k) + c_1 2k \\
&\leqslant c \cdot M(k) + \frac{5}{2}M(2k) + c_1 2k \\
&\leqslant \frac{c}{2}M(2k) + \frac{5}{2}M(2k) + c_1 2k \\
&\leqslant \frac{c}{2}M(2k) + \frac{5}{2}M(2k) + c_1 M(2k) \\
&\leqslant \left(\frac{c}{2} + \frac{5}{2} + c_1\right) M(2k) \\
&\leqslant \left(\frac{c}{2} + \frac{c}{2}\right) M(2k) \quad \text{by the choice of } c \\
&= c \cdot M(2k) \ .
\end{aligned}$$

This completes the induction step, and we are done. ∎

2.3. Complexity of Matrix Multiplication

Next, we turn our attention to another fundamental problem, i.e., matrix multiplication. Matrix multiplication is needed in numerous applications involving linear

algebra. Therefore, studying the complexity of matrix multiplication is not only of theoretical interest but also of fundamental practical importance.

In order to achieve as much generality as possible, we shall consider matrix multiplications for any matrices defined over a ring.

Let us first recall the definition of a ring.

Definition 2.2. *Let* S *be any non-empty set containing at least two distinguished elements* 0 *and* 1*, and let* $+$ *and* $\cdot$ *be binary operations over* S *(that is* $+ : S \times S \to S$ *and* $\cdot : S \times S \to S$*). Then* $R = (S, +, \cdot, 0, 1)$ *is a* ***ring*** *with identity element provided that for all* $a, b, c \in S$ *the following properties hold.*

(1) $(a + b) + c = a + (b + c)$ *and* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
(*i.e.,* $+$ *and* $\cdot$ *are associative*).

(2) $(a + b) = (b + a)$ ($+$ *is commutative*).

(3) $(a + b) \cdot c = a \cdot c + b \cdot c$ *and* $a \cdot (b + c) = a \cdot b + a \cdot c$ (*laws of distributivity*).

(4) $a + 0 = 0 + a = a$ (0 *is the neutral element with respect to* $+$).

(5) $a \cdot 1 = 1 \cdot a = a$ (1 *is the identity element with respect to* $\cdot$).

(6) *For each* $a \in S$ *there exist an element* $-a \in S$ *such that* $a+(-a) = (-a)+a = 0$ ($-a$ *is called the additive inverse of* a).

Since we only consider rings with identity, we refer to a ring with identity as to a ring for short.

Furthermore, if R is a ring and the operation $\cdot$ is commutative, then we say that the ring R is ***commutative***. Finally, if R is a commutative ring and if for every $a \in S \setminus \{0\}$ there exist an element $a^{-1} \in S$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$, then R is said to be a ***field***.

So, let $R = (S, +, \cdot, 0, 1)$ be a commutative ring with 1. Furthermore, let $n \in \mathbb{N}^+$. We consider the set M_n of all $n \times n$ matrices over R. Moreover, let

$$I_n = \begin{pmatrix} 1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & \cdot & \cdot & \cdot & 0 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & \cdot & \cdot & 0 & 1 \end{pmatrix}$$

and

$$0_n = \begin{pmatrix} 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ 0 & \cdot & \cdot & \cdot & 0 \end{pmatrix}$$

Addition and multiplication of matrices from M_n is defined as usual by using the addition and multiplication from the underlying ring R. We denote the resulting operations by $+_n$ and $\times_n$, respectively. That is, for $A = (a_{ij})$ and $B = (b_{ij})$, $i, j = 1, \ldots, n$, the sum $A +_n B$ is the $n \times n$ matrix C and the product $A \times_n B$ is the $n \times n$ matrix D defined by

$$A +_n B \quad =_{df} \quad C, \qquad \text{where } c_{ij} = a_{ij} + b_{ij} \quad i, j = 1, \ldots, n$$

$$A \times_n B \quad =_{df} \quad D, \qquad \text{where } d_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \quad i, j = 1, \ldots, n$$

The following important property is stated as an exercise.

Exercise 4. *Let* $\mathcal{M}_n = (M_n, +_n, \times_n, 0_n, I_n)$. *Then we have:* $\mathcal{M}_n$ *is a ring if and only if* R *is ring.*

Note that we did not require the ring R to be a commutative one in Exercise 4. Furthermore, it should be noted that the matrix multiplication $\times_n$ as defined above, is *not* commutative for $n > 1$, even if the multiplication $\cdot$ in the underlying ring R is commutative. In the following, we often omit the subscript n in $+_n$ and $\times_n$, i.e., we just write $+$ and $\times$, when there is no possibility of confusion. Moreover, we often just write AB instead of $A \times B$ to simplify notation.

Next, we continue with a very useful technical result. Let R be a commutative ring with 1, and let $\mathcal{M}_n$ be the ring of all $n \times n$ matrices over R. Assume n to be even. Then, we can divide any $n \times n$ matrix A into 4 matrices A_{11}, A_{12}, A_{21}, A_{22} of size $n/2 \times n/2$, i.e.,

$$A = \left(\begin{array}{ccc|ccc} a_{11} & \cdots & a_{1,\frac{n}{2}} & a_{1,\frac{n}{2}+1} & \cdots & a_{1n} \\ \vdots & & & & & \vdots \\ a_{\frac{n}{2}1} & \cdots & a_{\frac{n}{2},\frac{n}{2}} & a_{\frac{n}{2},\frac{n}{2}+1} & \cdots & a_{\frac{n}{2}n} \\ \hline a_{\frac{n}{2}+1,1} & \cdots & a_{\frac{n}{2}+1,\frac{n}{2}} & a_{\frac{n}{2}+1,\frac{n}{2}+1} & \cdots & a_{\frac{n}{2}+1,n} \\ \vdots & & & & & \vdots \\ a_{n1} & \cdots & a_{n,\frac{n}{2}} & a_{n,\frac{n}{2}+1} & \cdots & a_{nn} \end{array}\right) = \left(\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right)$$

Furthermore, let $R_{2,n/2}$ be the ring of all 2×2 matrices with elements from $M_{n/2}$. Then, the multiplication and addition of matrices from M_n is equivalent to the multiplication and addition of the corresponding 2×2 matrices from $R_{2,n/2}$. That is, for $A, B \in M_n$ and A_{11}, A_{12}, A_{21}, A_{22} B_{11}, B_{12}, B_{21}, $B_{22} \in R_{2,n/2}$ we have

$$A + B \quad = \quad \left(\begin{array}{cc} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \end{array}\right) \tag{2.3}$$

$$A \times B \quad = \quad \left(\begin{array}{cc} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{array}\right) \tag{2.4}$$

The proof of the latter statement is an elementary exercise and therefore here omitted.

Now, we are ready to deal with the complexity of matrix multiplications. What we are going to count here is the number of the arithmetic ring operations, i.e., additions and multiplications in the underlying ring R. Our first theorem establishes the starting point by analyzing the obvious matrix multiplication algorithm that is based on the definition of matrix multiplication.

Theorem 2.6. *The usual algorithm for multiplying any two $n \times n$ matrices requires $2n^3 - n^2$ arithmetic operations.*

Proof. Let $A = (a_{ij})$ and let $B = (b_{ij})$, $i, j = 1, \ldots, n$ be any two $n \times n$ matrices. Then the product $C = (c_{ij}) = A \times B$ is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \ .$$

That is, the computation of every element c_{ij} requires n ring multiplications and $n-1$ ring additions. Since there are n^2 many elements c_{ij} which have to be computed, the total number of ring multiplications is n^3 and the total number of ring additions is $n^2(n-1)$. Thus the overall number of ring operations is $2n^3 - n^2$. ∎

The obvious question is, of course, whether or not we can do better. Strassen [2] discovered a method of multiplying two 2×2 matrices with elements from an arbitrary ring using only seven multiplications. By using this method recursively, he was able to provide an algorithm for matrix multiplication that works in time $O(n^{\log 7})$ which is of order approximately $n^{2.81}$.

We continue with Strassen's [2] result.

Theorem 2.7. *There is an algorithm for multiplying any two $n \times n$ matrices that requires $O(n^{\log 7})$ ring operations.*

Proof. Let $T(n)$ denote the number of ring operations needed for multiplying any two $n \times n$ matrices. Furthermore, let $A, B \in M_n$ be any two $n \times n$ matrices. First, we assume n to be a power of 2. Using equation (2.4) we can write

$$C = A \times B = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

But this approach alone does not help, since we have reduced the original problem of size n into eight subproblems of size $n/2$. Thus, the resulting recursive equation has the solution $T(n) = O(n^{\log 8}) = O(n^3)$ (cf. Theorem 1.4). At this point Strassen [2] discovered the following. Let

$$\begin{aligned} M_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ M_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_3 &= (A_{11} - A_{21})(B_{11} + B_{12}) \\ M_4 &= (A_{11} + A_{12})B_{22} \end{aligned}$$

$$\begin{aligned} M_5 &= A_{11}(B_{12} - B_{22} \\ M_6 &= A_{22}(B_{21} - B_{11}) \\ M_7 &= (A_{21} + A_{22})B_{11} \ , \end{aligned}$$

then an easy calculation shows that

$$\begin{aligned} C_{11} &= M_1 + M_2 - M_4 + M_6 \\ C_{12} &= M_4 + M_5 \\ C_{21} &= M_6 + M_7 \\ C_{22} &= M_2 - M_3 + M_5 - M_7 \ . \end{aligned}$$

Thus, we have reduced the original problem of size n to *seven* multiplications of matrices having size $n/2 \times n/2$ and 18 additions of $n/2 \times n/2$ matrices. Consequently, for $n \geqslant 2$ we directly get the recursive equation

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \ ,$$

which has the solution $T(n) = 7 \cdot 7^{\log n} - 6n^2$ (cf. Theorem 1.4), and hence $T(n) = O(7^{\log n}) = O(n^{\log 7})$.

If n is not a power of 2, then we embed each matrix in a matrix whose dimension is the next-higher power of 2. This at most doubles the dimension and thus increases the constant by at most a factor of 7. Hence, $T(n) = O(n^{\log 7})$ for all $n \in \mathbb{N}^+$. ∎

Note that Theorem 2.7 is only concerned with the functional growth rate of $T(n)$. But if we really like to know for what values of n Strassen's algorithm outperforms the usual matrix multiplication algorithm, we have to determine the constant hidden in the big-O notation more carefully. The simple embedding of A and B in the next-higher power of 2 will give too large a constant. This is left as an exercise.

Finally, we remark that Strassen's [2] algorithm is also not the best possible. After his pioneering paper, many researchers worked on even faster matrix multiplication algorithms. The currently best known algorithm achieves $O(n^{2.376})$ and is due to Coppersmith and Winograd [1].

Finally, it should be noted that the improved matrix multiplication algorithms are not widely used in numerical computations. In such computations, one usually used floating point numbers and thus round off errors are a serious matter of concern. But the numerical error control for the faster matrix multiplication algorithms is not sufficiently well understood til now. Of course, this may change in the future.

References

[1] D. COPPERSMITH AND S. WINOGRAD (1990), Matrix multiplication via Arithmetic Progressions, *Journal of Symbolic Computation* **9** 251 – 280.

[2] V. STRASSEN (1969), Gaussian Elimination is not Optimal, *Numerische Mathematik* **13**, 354 – 356.

Lecture 3: Complexity of Number Theoretic Problems

Now, we want to have a closer look at the complexity of several problems arising in number theory. A certain part of the material included below should already be known. It is mainly included for completeness and self-containment of this course.

Clearly, we cannot provide an exhaustive study of all interesting problems. Instead, we concentrate ourselves on problems that will be needed when dealing with cryptography.

First, we formally introduce some notations and definitions. Consider $m \in \mathbb{N}^+$ and $a \in \mathbb{Z}$. Then there are uniquely determined numbers q, R such that $a = qm + R$, where $0 \leqslant R < m$ (as we should remember from school). Quite often, one is not interested in a number a itself but in its remainder when divided by a number m. Now, in order to deal with the properties that exclusively depend on the remainder R when a is divided by m, it is very useful to introduce the following notation.

Let $m \in \mathbb{N}^+$, and let $a, b \in \mathbb{Z}$; we write $a \equiv b \bmod m$ if and only if m divides $a - b$ (abbr. $m|(a-b)$). Thus, $a \equiv b \bmod m$ if and only if a and b have the *same* remainder when divided by m.

If $a \equiv b \bmod m$ then we say that a is *congruent* b modulo m, and we refer to "$\equiv$" as to the *congruence relation*.

It is easy to see that "$\equiv$" is an equivalence relation, i.e., it is *reflexive, symmetric* and *transitive*. Thus, we may consider the equivalence classes

$$[a] = \{x \in \mathbb{Z} |\ a \equiv x \bmod m\} .$$

Consequently, $[a] = [b]$ iff $a \equiv b \bmod m$. Therefore, there are precisely the m equivalence classes $[0], [1], \ldots, [m-1]$. We set $\mathbb{Z}_m = \{[0], [1], \ldots, [m-1]\}$. Next, we define addition and multiplication of these equivalence classes by

$$\begin{aligned} [a] + [b] &= [a + b] \text{ and} \\ [a] \cdot [b] &= [a \cdot b] . \end{aligned}$$

Exercise 5. *Show that the definition of $+$ and $\cdot$ over $\mathbb{Z}_m$ are independent of the choice of the representation.*

Now, it is easy to see that $(\mathbb{Z}_m, +, \cdot)$ constitutes a commutative ring. Clearly, the neutral element for addition is $[0]$ and the identity element with respect to multiplication is $[1]$. It is left as an exercise to verify these assertions formally.

Moreover, by the definition of a ring, it is immediate that $(\mathbb{Z}_m, +)$ is an Abelian group. We refer to this group also as to $\mathbb{Z}_m$ for short.

Note, however, that in general $(\mathbb{Z}_m, +, \cdot)$ is *not* a field. For example, let $m = 6$ and consider $[2] \cdot [3] = [6] = [0]$. Thus, $[2]$ and $[3]$ are non-trivial divisors of $[0]$ and a field does not have non-trivial divisors of $[0]$.

In order to see under what circumstances $(\mathbb{Z}_m, +, \cdot)$ is a field, we have to answer the question under which conditions the multiplicative inverses do always exist. This question is answered by Theorem 3.4 below. But before we can present it, we have to establish some useful rules for performing calculations with congruences. This will be done in the following subsection of this lecture. We shall also look at the complexity of some of the more important algorithms provided. For doing this, we measure the length of the inputs by the number of bits needed to write the input down. Moreover, whenever dealing with elements from $\mathbb{Z}_m$, we assume that they are represented by their canonical representations, i.e., by $0, \ldots, m-1$.

3.1. Calculating in $\mathbb{Z}_m$

We start with basic properties of the congruence relation. This is done by the following theorem.

Theorem 3.1. *Let $m \in \mathbb{N}^+$, let $a, b, c, d \in \mathbb{Z}$ be any integers and let $n \in \mathbb{N}$. Then we have the following.*

(1) *If $a \equiv b \bmod m$ and $c \equiv d \bmod m$ then $a + c \equiv b + d \bmod m$.*

(2) *If $a \equiv b \bmod m$ and $c \equiv d \bmod m$ then $a - c \equiv b - d \bmod m$.*

(3) *If $a \equiv b \bmod m$ and $c \equiv d \bmod m$ then $ac \equiv bd \bmod m$.*

(4) *If $a \equiv b \bmod m$ then $a^n \equiv b^n \bmod m$.*

The proof of the latter theorem is left as an exercise. The importance of the latter theorem is easily explained. Having Theorem 3.1, we see that we can calculate with congruences almost as convenient as with equations. The main difference is division. Division cannot be used. Instead we should use the modular inverses, but as we have seen, they may not exist. Before we can deal thoroughly with modular inverses, we have to recall the extended Euclidean algorithm for computing the greatest common divisor of two numbers. First, we define this problem more formally.

Problem 3.1. Greatest Common Divisor (abbr. gcd)

Input: Numbers $a, b \in \mathbb{N}$.

Problem: Compute the greatest number d dividing both a and b.

Since we are also interested in the complexity of the number theoretic problems we are dealing with, we have to say how we do present numbers. In the following, we always assume numbers to be represented in binary notation. Thus, we need $n = \lfloor \log a \rfloor + 1$ many bits to represent number a, and we refer to n as to the *length* of input a.

Theorem 3.2. *There exists an algorithm computing the gcd of two numbers a and b using at most $1.5 \log m + O(1)$ many divisions of numbers less than or equal to m, where $m = \max\{a, b\}$.*

Proof. We use the extended Euclidean algorithm. Without loss of generality, assume $a > b$. The following procedure computes $d = \gcd(a, b)$ as well as numbers $x, y \in \mathbb{Z}$ such that $d = ax + by$.

Procedure ECL: "Set $x_0 = 1,\ x_1 = 0,\ y_0 = 0,\ y_1 = 1$, and $r_0 = a,\ r_1 = b$.

Compute successively

$r_{i+1} = r_{i-1} - q_i r_i$, where $q_i = \lfloor \frac{r_{i-1}}{r_i} \rfloor$,

$x_{i+1} = x_{i-1} - q_i x_i$, and

$y_{i+1} = y_{i-1} - q_i y_i$ until $r_{i+1} = 0$.

Output $r_i,\ x_i,\ y_i$."

Claim A. *Procedure ECL computes* d, x, *and* y *correctly.*

We prove inductively that $r_0 x_i + r_1 y_i = r_i$ for $i \geqslant 0$. For $i = 0$ and $i = 1$ we directly obtain $r_0 x_0 + r_1 y_0 = r_0$ and $r_0 x_1 + r_1 y_1 = r_1$, respectively. Thus, we may assume the induction hypothesis for $i - 1$ and i for $i \geqslant 1$. By definition:

$x_{i+1} = x_{i-1} - q_i x_i$, and $y_{i+1} = y_{i-1} - q_i y_i$; thus

$$\begin{aligned} r_0 x_{i+1} + r_1 y_{i+1} &= r_0 x_{i-1} - r_0 q_i x_i + r_1 y_{i-1} - r_1 q_i y_i \\ &= \underbrace{r_0 x_{i-1} + r_1 y_{i-1}}_{= r_{i-1} \text{ by ind. hyp.}} - \underbrace{q_i (r_0 x_1 + r_1 y_i)}_{= r_i \text{ by ind. hyp.}} \\ &= r_{i-1} - q_i r_i = r_{i+1} \,. \end{aligned}$$

Furthermore,

$$r_{i+1} + r_i \leqslant r_{i-1} \text{ for all } i \geqslant 1. \tag{3.1}$$

This can be seen as follows. By construction, $r_{i+1} = r_{i-1} - q_i r_i$; hence $r_{i+1} + r_i = r_{i-1} + r_i(1 - q_i) \leqslant r_{i-1}$ provided $(1 - q_i) \leqslant 0$. The latter inequality obviously holds in accordance with q_i's definition. Moreover, all r_i are non-negative. Thus, there must be an $n \in \mathbb{N}$ such that $r_{n+1} = 0$.

It remains to show that $r_n = \gcd(a, b)$. Let $d = \gcd(a, b)$. As shown above, $r_n = r_0 x_n + r_1 y_n = a x_n + b y_n$. Thus d divides r_n. On the other hand, every divisor of r_n divides $a x_n + b y_n$. Since $r_{n+1} = 0$, we additionally know that $r_{n-1} = q_n r_n$. Therefore, r_n divides r_{n-1}, too. Consequently, $r_{n-2} = r_n + q_{n-1} r_{n-1}$ implies $r_n | r_{n-2}$. Iterating this argument directly yields r_n divides a and b. Thus, $r_n = d$. This proves Claim A, i.e., the correctness.

Claim B. *Procedure ECL uses at most* $1.5 \log m + O(1)$ *many divisions of numbers less than or equal to* m, *where* $m = \max\{a, b\}$.

By (3.1) we directly see that the number of divisions is maximal iff $r_{i+1} + r_i = r_{i-1}$ for all $i \geqslant 1$. Hence, the worst-case occurs if $a_0 = a_1 = 1$ and $a_\ell = a_{\ell-1} + a_{\ell-2}$ for

all $n \geqslant \ell \geqslant 2$, where $a = a_n$ and $b = b_{n-1}$, i.e., if a equals the nth member and b equals the $(n-1)$th member of the well-known Fibonacci sequence. Therefore, all we have to do is to estimate the size of the nth member of the Fibonacci sequence.

This leaves us with the problem to express the nth Fibonacci as a function solely depending on n. Fortunately enough, we have already successfully finished the course in calculus. There is a very powerful tool, called *generating functions* which seems appropriate to be used here. Therefore, we shortly recall the definition of generating functions and an important theorem from calculus.

3.2. Generating Functions and Fibonacci Numbers

Let $(a_n)_{n\in\mathbb{N}}$ be any sequence of real (or complex) numbers. Then

$$g(z) = \sum_{n=0}^{\infty} a_n z^n$$

is called ***generating function*** of $(a_n)_{n\in\mathbb{N}}$. The following theorem is often applied to generating functions.

Theorem 3.3. *Let $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ be any sequences such that that their generating functions have a radius $r > 0$ of convergence. Then*

$$\sum_{n=0}^{\infty} a_n z^n = \sum_{n=0}^{\infty} b_n z^n$$

if and only if $a_n = b_n$ for all $n \in \mathbb{N}$.

Moreover, recall that power series can be differentiated by differentiating their summands. Thus, we also know that

$$g'(z) = \sum_{n=0}^{\infty} n \cdot a_n z^{n-1} \ .$$

For more information about generating functions the interested reader is referred to Graham, Knuth, and Patashnik [1].

Now, let $(a_n)_{n\in\mathbb{N}}$ be the Fibonacci sequence. Thus, we have the generating function

$$g(z) = \sum_{n=0}^{\infty} a_n z^n$$

which we use as follows.

$$\begin{aligned} g(z) &= \sum_{n=0}^{\infty} a_n z^n = 1 + z + \sum_{n=2}^{\infty} a_n z^n \\ &= 1 + z + \sum_{n=2}^{\infty} (a_{n-1} + a_{n-2}) z^n \end{aligned}$$

$$\begin{aligned} &= 1 + z + \sum_{n=2}^{\infty} a_{n-1} z^n + + \sum_{n=2}^{\infty} a_{n-2} z^n \\ &= 1 + z + z \cdot \sum_{n=2}^{\infty} a_{n-1} z^{n-1} + z^2 \cdot \sum_{n=2}^{\infty} a_{n-2} z^{n-2} \\ &\quad (*\text{changing the summation indices yields}*) \\ &= 1 + z + z \left[\cdot \sum_{n=0}^{\infty} a_n z^n - 1 \right] + z^2 \cdot \sum_{n=0}^{\infty} a_n z^n \ . \end{aligned}$$

Next, we replace $\sum_{n=0}^{\infty} a_n z^n$ by $g(z)$ and obtain:

$$g(z) = 1 + z - z + zg(z) + z^2 g(z) = 1 + zg(z) + z^2 g(z) \ .$$

Hence, we arrive at

$$g(z) = \frac{1}{1 - z - z^2} \ .$$

Thus, we have found a representation of g as a rational function. All what is left for applying Theorem 3.3 is to develop this rational function in a power series. For that purpose, we have to calculate the zeros of the denominator. Solving

$$0 = z^2 + z - 1$$

directly yields

$$z_{0,1} = -\frac{1}{2} \pm \sqrt{\frac{1}{4} + 1} \ .$$

Next, we set

$$\alpha = \frac{-1 + \sqrt{5}}{2}$$

and

$$\hat{\alpha} = \frac{-1 - \sqrt{5}}{2} \ ,$$

and write

$$\frac{1}{1 - z - z^2} = \frac{1}{(z - \alpha)(\hat{\alpha} - z)} = \frac{A}{z - \alpha} + \frac{B}{\hat{\alpha} - z} \ .$$

Now, an easy calculation yields $A = B = -\frac{1}{\sqrt{5}}$, and consequently we have

$$g(z) = -\frac{1}{\sqrt{5}} \frac{1}{(z - \alpha)} - \frac{1}{\sqrt{5}} \frac{1}{(\hat{\alpha} - z)} \ .$$

Recalling that

$$\sum_{n=0}^{\infty} z^n = \frac{1}{1 - z}$$

we can write

$$\frac{1}{z - \alpha} = -\frac{1}{\alpha} \cdot \frac{1}{1 - \frac{1}{\alpha} z} = -\frac{1}{\alpha} \sum_{n=0}^{\infty} \frac{1}{\alpha^n} z^n$$

and

$$\frac{1}{\hat{\alpha}-z} = \frac{1}{\hat{\alpha}} \cdot \frac{1}{1-\frac{1}{\hat{\alpha}}z} = \frac{1}{\hat{\alpha}} \sum_{n=0}^{\infty} \frac{1}{\hat{\alpha}^n} z^n \ .$$

This yields the desired power series for g, i.e., we get

$$\begin{aligned} g(z) &= \sum_{n=0}^{\infty} a_n z^n \\ &= \frac{1}{\sqrt{5}\cdot\alpha} \sum_{n=0}^{\infty} \frac{1}{\alpha^n} z^n - \frac{1}{\sqrt{5}\cdot\hat{\alpha}} \sum_{n=0}^{\infty} \frac{1}{\hat{\alpha}^n} z^n \\ &= \frac{1}{\sqrt{5}} \sum_{n=0}^{\infty} \frac{1}{\alpha^{n+1}} z^n - \frac{1}{\sqrt{5}} \sum_{n=0}^{\infty} \frac{1}{\hat{\alpha}^{n+1}} z^n \\ &= \sum_{n=0}^{\infty} \left[\frac{1}{\sqrt{5}} \left(\frac{1}{\alpha^{n+1}} - \frac{1}{\hat{\alpha}^{n+1}} \right) \right] z^n \ . \end{aligned}$$

Thus, by Theorem 3.3 we obtain

$$a_n = \frac{1}{\sqrt{5}} \left(\frac{1}{\alpha^{n+1}} - \frac{1}{\hat{\alpha}^{n+1}} \right)$$

Finally, putting it all together, after a short calculation we arrive at

$$a_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right) \ . \tag{3.2}$$

Since $\frac{1-\sqrt{5}}{2} \approx 0.619 < 1$, we immediately see that $\lim_{n\to\infty} \frac{1-\sqrt{5}}{2} = 0$. Thus, a_n grows asymptotically as fast as $\frac{1+\sqrt{5}}{2}$; and hence $n+1 = \log_{\frac{1+\sqrt{5}}{2}} a_n$. We leave it to the reader to verify the precise estimate of the constants. This proves Claim B.

Putting Claim A and B together, directly yields Theorem 3.2. ∎

Exercise 6. *Use Euclid's Algorithm to find* $\gcd(1258, 5151)$. *Also, find an integer solution to* $5151x + 1258y = 119$.

Exercise 7. *Find the last digit of* 3^{1996}.

3.3. Algorithms for Computing in $\mathbb{Z}_m$

The following theorem completely characterizes the existence of modular inverses.

Theorem 3.4. *The congruence* $ax \equiv 1 \bmod m$ *is solvable iff* $\gcd(a, m) = 1$. *Moreover, if* $ax \equiv 1 \bmod m$ *is solvable, then the solution is uniquely determined.*

Proof. First, assume $\gcd(a, m) = 1$. We have to show that $ax \equiv 1 \bmod m$ is solvable. Since $\gcd(a, m) = 1$, there are integers x, y such that $1 = ax + my$. Hence, m divides $1 - ax$, i.e., $ax \equiv 1 \bmod m$. Thus $x \bmod m$ is the wanted solution.

Next, assume $ax \equiv 1 \bmod m$ to be solvable. Hence, there exists an x_0 such that $ax_0 \equiv 1 \bmod m$. Consequently, m divides $ax_0 - 1$, and therefore, there exists a y such that $my = ax_0 - 1$. Let d be any natural number dividing both m and a. Dividing the left side of the latter equation by d leaves the remainder 0. Hence, dividing the right side must also yield the remainder 0. Since $d|a$, we may conclude $d|1$, and thus $d = 1$.

Finally, assume $ax \equiv 1 \bmod m$ to be solvable. Suppose, there are solutions x_1, x_2. Thus, we have

$$ax_1 \equiv 1 \bmod m \tag{3.3}$$
$$ax_2 \equiv 1 \bmod m \tag{3.4}$$

By Theorem 3.1 we can subtract (3.4) from (3.3) and obtain $a(x_1 - x_2) \equiv 0 \bmod m$, i.e., m divides $a(x_1 - x_2)$. Since $\gcd(a, m) = 1$, we may conclude that m divides $x_1 - x_2$, i.e., $x_1 \equiv x_2 \bmod m$. Thus, the solution is unique modulo m. ∎

So, what can be said about the complexity of computing modular inverses? The answer is given by the following theorem.

Theorem 3.5. *Modular inverse can be computed in time* $O(\max\{\log a, \log m\}^3)$.

Proof. As the proof of Theorem 3.4 shows, all we have to do is to apply Procedure ECL presented above. Thus, the assertion follows. ∎

By Theorem 3.4 it is appropriate to consider $\mathbb{Z}_m^* = \{[a] \in \mathbb{Z}_m \mid \gcd(a, m) = 1\}$. Note that Theorem 3.4 directly implies that $(\mathbb{Z}_m^*, \cdot)$ constitutes a group. Again, we simplify notation and refer to $(\mathbb{Z}_m^*, \cdot)$ as to $\mathbb{Z}_m^*$ for short. Furthermore, we usually omit the brackets when referring to members of $\mathbb{Z}_m$ and $\mathbb{Z}_m^*$, respectively. That is, we write $a \in \mathbb{Z}_m$ and $a \in \mathbb{Z}_m^*$ instead of $[a] \in \mathbb{Z}_m$ and $[a] \in \mathbb{Z}_m^*$, respectively.

In order to get more familiarity with the congruence relation $\equiv$, let us derive a rule for deciding whether or not an integer given in decimal notation is divisible by 3. Since the divisibility by 3 is not affected by the sign, it suffices to consider $z = \sum_{i=0}^{n} z_i 10^i$, where $z_i \in \{0, 1, \ldots, 9\}$ for all $i = 0, \ldots, n$. Then, by the reflexivity of "$\equiv$" we have

$$z_i \equiv z_i \bmod 3 \tag{3.5}$$

for all $i = 0, \ldots, n$. Moreover, $10 \equiv 1 \bmod 3$ and thus by Property (4) of Theorem 3.1 we know that

$$10^n \equiv 1^n \equiv 1 \bmod 3\ . \tag{3.6}$$

Next, we apply Property (1) of Theorem 3.1 $(n + 1)$ many times to (3.5) and (3.6) and obtain

$$\sum_{i=0}^{n} z_i 10^i \equiv \sum_{i=0}^{n} z_i \bmod 3\ .$$

Consequently, we directly get the following theorem.

Theorem 3.6. *A number given in decimal notation is divisible by 3 if and only if the sum of its digits is divisible by 3.*

The proof given above directly allows a corollary concerning the divisibility by 9. By reflexivity we also have

$$z_i \equiv z_i \bmod 9 \tag{3.7}$$

and (3.6) also holds modulo 9, i.e.,

$$10^n \equiv 1^n \equiv 1 \bmod 9\ . \tag{3.8}$$

Thus, putting (3.7) and (3.8) together directly yields the following corollary.

Corollary 3.7. *A number given in decimal notation is divisible by 9 if and only if the sum of its digits is divisible by 9.*

In order to see that decimal notation is crucial here, let us consider numbers given in binary, i.e., $z = \sum_{i=0}^{n} z_i 2^i$, where $z_i \in \{0,1\}$ for all $i = 0, \ldots, n$. Again, we have

$$z_i \equiv z_i \bmod 3 \tag{3.9}$$

as before, but (3.6) translates into

$$2^n \equiv (-1)^n \bmod 3\ . \tag{3.10}$$

Thus, now we get

$$\sum_{i=0}^{n} z_i 2^i \equiv \sum_{i=0}^{n} (-1)^n z_i \bmod 3\ .$$

Consequently, *a number given in binary notation is divisible by 3 if and only if the alternating sum of its digits is divisible by 3.*

We finish this lecture by proving an important theorem that will be needed later. Before we can present it, we need the following definition.

Definition 3.1. *Integers* a *and* b *are said to be* ***relatively prime*** *if* $\gcd(a,b) = 1$.

Integers $m_1, \ldots, m_r$ *are said to be* ***pairwise relatively prime*** *if every pair* m_i, m_j, $i \neq j$ *is relatively prime.*

Theorem 3.8 (Chinese Remainder Theorem).
Let $m_1, \ldots, m_r$ *be pairwise relatively prime numbers, and let* $M = \prod_{i=1}^{r} m_i$. *Furthermore, let* $a_1, \ldots, a_r$ *be any integers. Then there is a unique* $y \in \mathbb{Z}_M$ *such that* $y \equiv a_i \bmod m_i$ *for* $i = 1, \ldots, r$. *Moreover,* y *can be computed in time polynomial in the length of the input.*

Proof. For each $i = 1, \ldots, r$, we set $n_i = M/m_i$. Then for all $i = 1, \ldots, r$, the number n_i satisfies $n_i \in \mathbb{N}$, and $\gcd(m_i, n_i) = 1$. Consequently, the modular inverses n_i^{-1} modulo m_i do exist for all $i = 1, \ldots, r$. Now, let

$$\hat{y} = \sum_{i=1}^{r} n_i \cdot n_i^{-1} \cdot a_i$$

and let y be $\hat{y}$ reduced modulo M. Taking into account that $m_i | n_j$ for all $i = 1, \ldots, r$, $j = 1, \ldots, r$ provided $j \neq i$, we conclude

$$y \equiv \hat{y} \equiv n_i n_i^{-1} a_i \equiv a_i \bmod m_i \ .$$

Thus, we have found a number y simultaneously fulfilling all the wanted congruences.

It remains to show that this y is uniquely determined modulo M. Suppose the converse, i.e., there exists an x such that $x \equiv a_i \bmod m_i$ for $i = 1, \ldots, r$ and $x \not\equiv y \bmod M$. Subtracting $y \equiv a_i \bmod m_i$ from $x \equiv a_i \bmod m_i$ for all $i = 1, \ldots, r$ yields $x - y \equiv 0 \bmod m_i$ for all $i = 1, \ldots, r$, and thus m_i divides $x - y$. However, all the m_i are pairwise relatively prime. Hence, $\prod_{i=1}^{r} m_i$ must divide $(x - y)$, too. But this means $x - y \equiv 0 \bmod M$, a contradiction. Thus, y is uniquely determined modulo M.

Finally, by Theorem 3.5 we know that the modular inverses can be each computed in time polynomial in the input. All other computations, i.e., multiplication, addition and reduction modulo M are known to be performable in polynomial time, too. ∎

Now, let us try it out. Note that Theorem 3.8 is telling us nothing in case the moduli are not pairwise relatively prime.

Exercise 8. *Use the Chinese Remainder Theorem to find a solution (or show one doesn't exist) for the following sets of equations.*

(1) $x \equiv 3 \bmod 7,\ x \equiv 4 \bmod 9,\ x \equiv 2 \bmod 5$

(2) $x \equiv 10 \bmod 21,\ x \equiv 3 \bmod 8,\ x \equiv 8 \bmod 15$

The following exercises establish some nice properties which we need later.

Exercise 9. *Let n be an integer greater or equal to 2. Prove the* zero-divisor property *for the integers* mod n*, i.e.,*

$$\forall k\, \forall \ell \ .\ \text{if } k\ell \equiv 0 \bmod n,\ \text{then either } k \equiv 0 \bmod n \text{ or } \ell \equiv 0 \bmod n.$$

holds if and only if n is prime.

Exercise 10. *Let n be an integer greater or equal to 2. Prove the* cancellation law *for the integers* mod n*, i.e.,*

$$\forall j\, \forall k\, \forall \ell \ .\ \text{if } j \not\equiv 0 \bmod n \text{ and } jk \equiv j\ell \bmod n \text{ then } k \equiv \ell \bmod n.$$

holds if and only if n is prime.

The following references are recommended for further reading.

References

[1] R.L. GRAHAM, D.E. KNUTH AND O. PATASHNIK (1989), *Concrete Mathematics* (Addison-Wesley, Reading, Massachusetts).

[2] N. KOBLITZ (1994), *A Course in Number Theory and Cryptography* (Springer, Berlin).

Lecture 4: Number Theoretic Algorithms

4.1. Solving Linear Congruences

In order to develop some more familiarity with calculations in the ring $\mathbb{Z}_m$ we continue by studying the solvability of the easiest form of congruences involving a variable, i.e., of linear congruences $ax \equiv c \bmod b$. This is an important practical problem. There may be zero, one, or more than one solution satisfying $ax \equiv c \bmod b$. The following theorem precisely characterizes the solvability of linear congruences.

Theorem 4.1. *Let $a, c \in \mathbb{Z}$ and let $b \in \mathbb{N}$, $b \geqslant 2$. Then the linear congruence $ax \equiv c \bmod b$ is solvable if and only if $\gcd(a, b)$ divides c. Moreover, if $d = \gcd(a, b)$ and $d|c$ then there are precisely d solutions in $\mathbb{Z}_b$ for $ax \equiv c \bmod b$.*

Proof. The proof is quite similar to the demonstration of Theorem 3.4. First, let $d = \gcd(a, b)$ and let us assume that d divides c. Then we consider $\tilde{a} = a/d$, $\tilde{b} = b/d$, $\tilde{c} = c/d$, and $\tilde{a}x \equiv \tilde{c} \bmod \tilde{b}$. Since $\gcd(\tilde{a}, \tilde{b}) = 1$, we can apply Theorem 3.4 and conclude that there is a number y such that

$$\tilde{a}y \equiv 1 \bmod \tilde{b} \ . \tag{4.1}$$

Consequently, multiplying (4.1) with $\tilde{c}$ yields

$$\begin{aligned} \tilde{a}y\tilde{c} &\equiv \tilde{c} \bmod \tilde{b} \\ \tilde{a}x_0 &\equiv \tilde{c} \bmod \tilde{b} \ , \end{aligned} \tag{4.2}$$

where $x_0 = y\tilde{c}$. Hence, there is a $k \in \mathbb{Z}$ such that

$$k\tilde{b} = \tilde{a}x_0 - \tilde{c} \ .$$

Multiplying both sides by d directly yields

$$\begin{aligned} k\tilde{b}d &= \tilde{a}dx_0 - \tilde{c}d \\ kb &= ax_0 - c \end{aligned}$$

but this means nothing else than $ax_0 \equiv c \bmod b$. Consequently, x_0 is also a solution of $ax \equiv c \bmod b$.

The remaining $(d-1)$ solutions of $ax \equiv c \bmod b$ are obtained by setting $x_j = x_0 + j\tilde{b}$ for $j = 1, \ldots, d-1$. Note that $x_0 < x_0 + \tilde{b} < \cdots < x_0 + (d-1)\tilde{b}$. Therefore, $x_0,\ x_0 + \tilde{b},\ \ldots,\ x_0 + (d-1)\tilde{b}$ are pairwise incongruent modulo b. On the other hand, since $j\tilde{b} \equiv 0 \bmod \tilde{b}$ for all $j \in \mathbb{Z}$, we also have

$$\tilde{a}(x_0 + j\tilde{b}) \equiv \tilde{c} \bmod \tilde{b} \ ,$$

and thus there are k_j, $j = 1, \ldots, d-1$, such that

$$k_j\tilde{b} = \tilde{a}(x_0 + j\tilde{b}) - \tilde{c} \ .$$

Multiplying both sides of the latter equality by d directly yields

$$k_j b = a(x_0 + j\tilde{b}) - c \ ,$$

which again directly implies $a(x_0+j\tilde{b}) \equiv c \bmod b$. Thus, $x_0,\ x_0+\tilde{b},\ \ldots,\ x_0+(d-1)\tilde{b}$ are all solutions of $ax \equiv c \bmod b$.

It remains to show that there are no other solutions. Suppose the converse, i.e., there is a z such that

$$\begin{aligned} az &\equiv c \bmod b && (4.3)\\ z &\not\equiv x_0 + j\tilde{b} \bmod b \text{ for all } j = 0, \ldots, d-1 \ . && (4.4) \end{aligned}$$

Now, (4.3) implies $\tilde{a}z \equiv \tilde{c} \bmod \tilde{b}$ and since $\gcd(\tilde{a}, \tilde{b}) = 1$, by (4.2) we can conclude

$$z \equiv x_0 \bmod \tilde{b} \ .$$

Therefore, $z = x_0 + \ell\tilde{b}$. Finally, since $d\tilde{b} = b$, we can conclude $\ell \in \{0, \ldots, d-1\}$, a contradiction to (4.4). Consequently, there are precisely d different solutions of $ax \equiv c \bmod b$.

Second, let us assume that $ax \equiv c \bmod b$ is solvable. We have to show that $\gcd(a, b)$ divides c. Let z be a solution of $ax \equiv c \bmod b$, i.e., we have $az \equiv c \bmod b$. Thus, there must be a $k \in \mathbb{Z}$ such that $kb = az - c$. But this means $kb - az = -c$ and consequently $\gcd(a, b)$ divides c. ∎

Exercise 11. *Determine the complexity of computing all solutions of the linear congruence* $ax \equiv c \bmod b$ *in dependence on the length of the input* $a, c \in \mathbb{Z}$ *and* $b \in \mathbb{N}$, $b \geqslant 2$.

Exercise 12. *Calculate all solutions of* $111x \equiv 75 \bmod 321$.

Moreover, we directly obtain the following corollary.

Corollary 4.2. *Let* $b \in \mathbb{N}$, $b \geqslant 2$, *and let* $a, c \in \mathbb{Z}$. *If* $\gcd(a, b) = 1$ *then the linear congruence* $ax \equiv c \bmod b$ *has a unique solution modulo* b.

Next, we should apply our knowledge about linear congruences to the problem of computing all integer solutions of *linear Diophantine equations*, i.e., equations of the form $ax + by = c$ for $a, b, c \in \mathbb{Z}$. This is left as an exercise. As we shall see, a linear Diophantine equation has either no solution or it has infinitely many. Thus, if it is solvable we cannot compute all of its solutions. However, in practice, we are often given additional constraints that the solutions must satisfy. In many cases, this yields a finite set of solutions which we can compute. Here is an example.

Exercise 13. *Solve the following problem:*

Hiroko has 10 000 Yen and has to buy eggs, chicken pieces, or turkey. An egg cost 25 Yen, a chicken piece 100 Yen and a turkey is 2500 Yen. Hiroko has to by 100 items of at least two different types and she should spend all her money. What is she buying?

Next, we look at the problem of computing large powers modulo a number m. Computing large powers quickly leads to very large numbers, and thus already outputting these numbers requires a lot of time. Moreover, in many applications is not necessary to compute really large powers but instead it suffices to compute large powers modulo a number m. This problem is usually referred to as to modular exponentiation.

4.2. Modular Exponentiation

Modular exponentiation is formally defined as follows.

Problem 4.2. Modular Exponentiation

Input: Modulus $m \in \mathbb{N}$, $m \geqslant 2$, and $a \in \mathbb{Z}_m^*$ as well as $x \in \mathbb{N}$.

Problem: Compute the $y \in \{0, 1, \ldots, m-1\}$ such that $y \equiv a^x \bmod m$.

Theorem 4.3. *Modular exponentiation can be computed in time*

$O(\max\{\log a, \log m, \log x\}^3)$.

Proof. Let $x = \sum_{i=0}^{k} x_i 2^{k-i}$ where $x_i \in \{0, 1\}$, i.e., x_i are the digits of x in binary notation. Then, the following procedure computes $a^x \bmod m$.

Procedure EXP: "Set $y_0 = 1$

For $i = 0$ to k do

If $x_i = 0$ then $y_{i+1} := y_i^2 \bmod m$;

If $x_i = 1$ then $y_{i+1} := a \cdot y_i^2 \bmod m$;

Output y_{k+1}."

Claim A. *Procedure EXP computes* y *correctly.*

We prove Claim A by induction on k. For $k = 0$ we distinguish the cases $x = 0$ and $x = 1$. Obviously, if $x = 0$, then $y_1 = 1 \equiv a^0 \bmod m$, and thus correct. If $x = 1$, then $y_1 = a \equiv a^1 \bmod m$, and hence again correct.

Now, assume the induction hypothesis for k. Let $x = x_0 \ldots x_k x_{k+1}$. Thus, we may write $x = 2(x_0 \cdots x_k) + x_{k+1}$, and obtain

$$a^x = a^{2(x_0 \cdots x_k) + x_{k+1}} = a^{2(x_0 \cdots x_k)} \cdot a^{x_{k+1}} \equiv (a^{x_0 \cdots x_k})^2 \cdot a^{x_{k+1}} \equiv y_{k+1}^2 a^{x_{k+1}} \bmod m \,.$$

Note that the latter congruence is due to the induction hypothesis. Consequently, if $x_{k+1} = 0$ then $y_{k+2} \equiv y_{k+1}^2 \bmod m$, and thus correct. Finally, if $x_{k+1} = 1$ then $a^{x_{k+1}} = a$, and hence $y_{k+2} \equiv a \cdot y_{k+1}^2 \bmod m$ which is again correct.

Finally, Procedure *EXP* computes at most $2\lceil \log x \rceil$ many products modulo m over numbers from $\mathbb{Z}_m$. Thus, the Procedure EXP takes at most time cubic in the lengths of a, m, x. ∎

The latter theorem shows that we can exponentiate efficiently modulo m, but what about the inverse operations? Finding discrete roots of numbers modulo m appears little less tractable, if m is prime or if the prime factorization of m is known. In the general case, the problem of taking discrete roots seems sufficiently intractable that is has been proposed as the basis of the RSA public key cryptosystem (cf. Lecture 11).

4.3. Towards Discrete Roots

We continue to recall basic number theory to the extend needed for designing our main algorithms. Let $m \in \mathbb{N}$; by $\varphi(m) = |\mathbb{Z}_m^*|$ we denote *Euler's totient function*. A natural number $p \geqslant 2$ is said to be *prime* if it is only divisible by itself and 1. The following exercise summarizes some well-know facts.

Exercise 14. *Prove the following:*

(1) $\varphi(mn) = \varphi(m)\varphi(n)$ *if* $\gcd(m, n) = 1$,

(2) $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ *if* p *is prime and* $\alpha \geqslant 1$,

(3) $\varphi(p) = p - 1$ *if and only if* p *is prime.*

Next, we formally define the problem of taking discrete roots.

Problem 4.3. Discrete Roots

Input: Modulus $m \in \mathbb{N}$, $a \in \mathbb{Z}_m^*$, and $r \in \mathbb{N}$.

Problem: Compute the solutions of $x^r \equiv a \bmod m$ provided they exist or output "there are no solutions."

For dealing with discrete roots as well as for dealing with primality tests, we need some more insight into the structure of the multiplicative group $\mathbb{Z}_m^*$. In order to avoid confusion, the following remark is mandatory. It is a well-known result of elementary group theory that every group of prime order is cyclic. Recall that the *order* of a finite group is the number of its elements. However, the group $\mathbb{Z}_p^*$ for p prime, we want to deal with, has the order $p - 1$ which is *not prime*. Furthermore, recall that the *order* of an element $g \in \mathbb{Z}_m^*$ is defined to be the least positive integer d for which $g^d \equiv 1 \bmod m$.

Theorem 4.4. *If* p *is prime then* $\mathbb{Z}_p^*$ *is a cyclic group of order* $p - 1$.

Proof. Let p prime. By Exercise 14 we already know that $\varphi(p) = |\mathbb{Z}_p^*| = p - 1$; thus $\mathbb{Z}_p^*$ has order $p - 1$. For seeing that $\mathbb{Z}_p^*$ is cyclic, we have to show that is has an element of order $p - 1$. This is achieved by counting elements of different order. Let d be any positive integer such that $d|(p - 1)$. Define

$$S_d = \{a \in \mathbb{Z}_p^* \mid \ a \text{ is of order } d\} \tag{4.5}$$

These sets S_d partition $\mathbb{Z}_p^*$, so we have

$$\sum_{d|(p-1)} |S_d| = |\mathbb{Z}_p^*| = p - 1 \tag{4.6}$$

Fix d such that $d|(p-1)$. We show that either $|S_d| = 0$ or $|S_d| = \varphi(d)$. Suppose $S_d \neq \emptyset$, and choose some $a \in S_d$. Then $a, a^2, \ldots, a^d$ are all distinct modulo p and each one is a solution of $x^d \equiv 1 \bmod p$. By the Lemma 4.5 below, this equation has at most d solutions modulo p, so these are all of the solutions. Consequently, $S_d \subseteq \{a^k \mid 1 \leqslant k \leqslant d\}$.

Now, fix $k \in \{1, \ldots, d\}$. If $\gcd(k, d) = \ell > 1$, then $(a^k)^{d/\ell} = (a^{k/\ell})^d \equiv 1 \bmod p$, so a^k has order less than d, thus $a^k \notin S_d$.

If $\gcd(k, d) = 1$, then there exists ℓ such that $k\ell \equiv 1 \bmod d$ (cf. Theorem 3.4). Hence, $a^{k\ell} \equiv a \bmod p$. Furthermore, for any $e \in \{1, \ldots, d-1\}$ we have

$$((a^k)^e)^\ell \equiv a^e \not\equiv 1 \bmod p,$$

so a^k is of order d, i.e., $a^k \in S_d$.

Thus, we have shown

$$S_d = \{a^k \mid 1 \leqslant k \leqslant d,\ \gcd(k, d) = 1\}$$

and consequently $|S_d| = \varphi(d)$.

Now suppose that for some d such that $d|(p-1)$, $S_d = \emptyset$. Then

$$\sum_{d|(p-1)} |S_d| < \sum_{d|(p-1)} \varphi(d)\ . \tag{4.7}$$

By Lemma 4.6 below,

$$\sum_{d|(p-1)} \varphi(d) = p - 1\ .$$

Thus, (4.7) would give a contradiction to (4.6). Hence, for each d with $d|(p-1)$ we have $|S_d| = \varphi(d)$. This proves the theorem. Additionally, note that in particular, the number of elements of order $p-1$ is $\varphi(p-1)$. ∎

It remains to provide the lemmata announced during the proof above. The first lemma is left as an exercise (hint: use induction).

Exercise 15. *Prove the following lemma:*

Lemma 4.5. *If p is prime and $f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ is such that $f(b) \not\equiv 0 \bmod p$ for some b, then $f(b) \equiv 0 \bmod p$ has at most n distinct solutions modulo p.*

Finally, we prove the remaining lemma announced above.

Lemma 4.6. *For all positive integers* n *we have* $\sum_{d|n} \varphi(d) = n$

Proof. Let n be some positive integer. For each d such that d divides n define

$$R_d = \{m \cdot i |\ m = n/d \text{ and } i \in \mathbb{Z}_d^*\}$$

Clearly, $R_d \subset \{1, \ldots, n\}$ and $|R_d| = \varphi(d)$. Consider any $x \in \{1, \ldots, n\}$. Let $m = \gcd(x, n)$, $d = n/m$, and $i = x/m$. Then, since $x = mi$ and $n = md$ we have $\gcd(i, d) = 1$; thus $x \in R_d$. If for some e such that e divides n we would have $x \in R_e$, then $x = \hat{m}\hat{\imath}$, where $n = \hat{m}e$ and $\gcd(e, \hat{\imath}) = 1$. Hence, $\hat{m} = \gcd(x, n)$ and $e = d$. Thus, for each $x \in \{1, \ldots, n\}$ the number x belongs to one and only one of the sets R_d. Consequently,

$$n = \sum_{d|n} |R_d| = \sum_{d|n} \varphi(d)$$

∎

As we have seen, if p is prime then $\mathbb{Z}_p^*$ is cyclic. Every element g of order $p - 1$ is called a *generator* of $\mathbb{Z}_p^*$. Hence, for every $a \in \mathbb{Z}_p^*$ there exists exactly one $x \in \{1, 2, \ldots, p\}$ such that $a = g^x$. We refer to x as to the *discrete logarithm* of a with respect to g, and denote it by $x = \mathrm{dlog}_g a$.

Not that the condition p being prime is sufficient but not necessary for the cyclicity of $\mathbb{Z}_p^*$, since one can prove the following.

Theorem 4.7. *$\mathbb{Z}_n^*$ is cyclic if and only if n is 1, 2, 4, p^k, or $2p^k$ for some odd prime number p and $k \in \mathbb{N}^+$.*

We omit the proof of this theorem here and refer the reader to Niven and Zuckerman [2].

So, it is appropriate to generalize the definition of discrete logarithms.

Definition 4.1 (Discrete Logarithm).
*Let $n \in \mathbb{N}$ be such that $\mathbb{Z}_n^*$ is cyclic. Furthermore, let g be a generator of $\mathbb{Z}_n^*$ and let $a \in \mathbb{Z}_n^*$. Then there exists a unique number $z \in \mathbb{N}$ such that $g^z \equiv a \bmod n$. This z is called the **discrete logarithm** of a modulo n to the base g and denoted by* $\mathrm{dlog}_g a$.

Now, let p be a prime and let g be any generator for $\mathbb{Z}_p^*$. Then we obviously have $g^{p-1} \equiv 1 \bmod p$. The latter property is, however, not restricted to generators as the following theorem shows.

Theorem 4.8 (Fermat's Little Theorem). *Let p be a prime. Then $a^{p-1} \equiv 1 \bmod p$ for all $a \in \mathbb{Z}_p^*$.*

Proof. Let $a \in \mathbb{Z}_p^*$ be arbitrarily fixed. Since $\mathbb{Z}_p^*$ is a multiplicative group, the elements a, $2a$, $3a$, $\ldots (p-1)a$ form a complete system of residues modulo p, i.e.,

$$\{a,\ 2a,\ 3a,\ \ldots (p-1)\ a\} = \{1,\ 2,\ 3, \ldots,\ p-1\}\ . \qquad (+)$$

For seeing this, suppose there are $b, c \in \{1, 2, 3, \ldots, p-1\}$ such that $ba \equiv ca \bmod p$ and $b \neq c$. By the choice of b, c we may conclude that $b \not\equiv c \bmod p$, too. On the other hand, a has a multiplicative inverse a^{-1} and therefore $baa^{-1} \equiv caa^{-1} \bmod p$. Since $aa^{-1} \equiv 1 \bmod p$, we may conclude $b \equiv c \bmod p$, a contradiction.

Thus, by (+) we may conclude $\prod_{i=1}^{p-1} ia \equiv \prod_{i=1}^{p-1} i \bmod p$. Consequently,

$$a^{p-1}(p-1)! \equiv (p-1)! \bmod p\ .$$

Finally, since p is prime we additionally know $\gcd(p, (p-1)!) = 1$. Thus, $(p-1)!$ has a multiplicative inverse (cf. Theorem 3.4). Hence, multiplying both sides of the latter congruence by it results in $a^{p-1} \equiv 1 \bmod p$. ∎

Next, we prove the following generalization of Theorem 4.8 usually referred to as ***Theorem of Euler***.

Theorem 4.9. *Let $n \in \mathbb{N}$, $n \geqslant 2$; then $a^{\varphi(n)} \equiv 1 \bmod n$ for all $a \in \mathbb{Z}_n^*$.*

Proof. Taking into account that the order of $\mathbb{Z}_n^*$ is $\varphi(n)$ the Theorem follows by recalling a little group theory.

For the sake of convenience, we also provide a direct proof of it. First we prove the Theorem of Euler for the case when m is a prime power, i.e., $n = p^x$. This can be easily done using induction on x. For $x = 1$, the induction base is precisely the Fermat's Little Theorem (cf. Theorem 4.8). Now, let $x \geqslant 2$ and assume the induction hypothesis for p^{x-1}. Taking into account that $\varphi(p^x) = p^{x-1}(p-1) = p^x - p^{x-1} = p(p^{x-1} - p^{x-2})$, we directly obtain:

$$a^{\varphi(p^x)} \equiv a^{p(p^{x-1}-p^{x-2})} \equiv (a^{p^{x-1}-p^{x-2}})^p \bmod p^x. \tag{4.8}$$

Now, observing that $\varphi(p^{x-1}) = p^{x-1} - p^{x-2}$ we may apply the induction hypothesis, and obtain $a^{p^{x-1}-p^{x-2}} \equiv 1 \bmod p^{x-1}$. Hence $a^{p^{x-1}-p^{x-2}} = 1 + p^{x-1}b$ for some integer b. Let $y = p^{x-1}b$; by the binomial theorem we have

$$(1+y)^p = \sum_{\nu=0}^{p} \binom{p}{\nu} 1^\nu y^{p-\nu}.$$

Since p divides $\binom{p}{\nu}$ for all $\nu = 1, \ldots, p-1$ we obtain $(1+y)^p \equiv 1 + y^p \bmod p$. Since $(x-1)p \geqslant x$, (4.8) delivers

$$a^{\varphi(p^x)} \equiv 1 + (p^{x-1}b)^p \equiv 1 + p^{(x-1)p}b^p \equiv 1 \bmod p^x.$$

This proves the Theorem of Euler for the case that m is a prime power. Now, let n be any number with $n \geqslant 2$. Then, let $n = p_1^{x_1} \cdots p_k^{x_k}$ be the unique prime factorization of n. Since $\gcd(p_i^{x_i}, p_j^{x_j}) = 1$, the theorem directly follows by applying the following. If $a \equiv b \bmod m_1$ and $a \equiv b \bmod m_2$, and $\gcd(m_1, m_2) = 1$, then $a \equiv b \bmod m_1 m_2$. ∎

Testing primality means that one has to decide for any given number taken as input whether or not it is prime. Though this is a very old problem, no deterministic algorithm has been known that runs in time polynomial in the length of the input until 2002. Then Agrawal, Kayal and Saxena [1] succeeded to provide an affirmative answer to this very long standing open problem. However, Solovay and Strassen [3] provided an efficient probabilistic algorithm for testing primality. The Solovay-Strassen [3] algorithm is also much faster than the newly found deterministic polynomial time primality test. We therefore present here only the Solovay-Strassen algorithm. The reader is refered to Agrawal, Kayal and Saxena [1] for the deterministic polynomial time primality test.

Clearly, one could get a deterministic polynomial time algorithm for testing primality, if the converse of Theorem 4.8 were true. Unfortunately, it is not. We continue by figuring out why the converse of Theorem 4.8 is not true.

4.4. Pseudo Primes

Definition 4.2 (Pseudo Primes). *Let* $n \in \mathbb{N}$ *be an odd composite number, and let* $b \in \mathbb{N}$ *such that* $\gcd(b, n) = 1$. *Then* n *is said to be* ***pseudo-prime to the base*** b *if* $b^{n-1} \equiv 1 \bmod n$.

For example, $n = 91$ is a pseudo-prime to the base 3, since $91 = 7 \cdot 13$ and, furthermore, $3^{90} \equiv 1 \bmod 91$ (note that $3^6 = 729 = 8 \cdot 91 + 1 \equiv 1 \bmod 91$).

But 91 is not a pseudo-prime to the base 2, because of $2^{90} \equiv 64 \bmod 91$.

The following theorem summarizes important properties of pseudo-primes.

Theorem 4.10. *Let* $n \in \mathbb{N}$ *be an odd composite number. Then we have:*

(1) n *is pseudo-prime to the base* b *with* $\gcd(b, n) = 1$ *if and only if the order* d *of* b *in* $\mathbb{Z}_n^*$ *divides* $n - 1$.

(2) *If* n *is pseudo-prime to the bases* b_1 *and* b_2 *such that* $\gcd(b_1, n) = 1$ *and* $\gcd(b_2, n) = 1$, *then* n *is also pseudo-prime to the bases* $b_1 b_2$, $b_1 b_2^{-1}$, *and* $b_1^{-1} b_2$.

(3) *If there is a* $b \in \mathbb{Z}_n^*$ *satisfying* $b^{n-1} \not\equiv 1 \bmod n$, *then*

$$|\{b \in \mathbb{Z}_n^* \mid \ b^{n-1} \not\equiv 1 \bmod n\}| \geqslant \frac{\varphi(n)}{2} \ .$$

Proof. First, we show (1). The necessity can be seen as follows. Let n be pseudo-prime to the base b with $\gcd(b, n) = 1$. Then, we have $b^{n-1} \equiv 1 \bmod n$. Let d be the smallest positive number for which $b^d \equiv 1 \bmod n$. Suppose, $n - 1 = kd + r$ with $0 < r < d$. Then we would get

$$b^{n-1} \equiv b^{kd+r} \equiv b^{kd} b^r \equiv \left(b^d\right)^k b^r \equiv b^r \not\equiv 1 \bmod n \ ,$$

a contradiction. Hence, d must divide $n - 1$.

For the sufficiency, assume d divides $n-1$. Thus, $n-1 = kd$ for some k. Hence, $b^{n-1} \equiv (b^d)^k \equiv 1^k \equiv 1 \bmod n$. Consequently, n is pseudo-prime to the base b.

Assertion (2) is left as an exercise.

Finally, we prove (3). Let $b \in \mathbb{Z}_n^*$ be such that $b^{n-1} \not\equiv 1 \bmod n$. Let $\{b_1, \ldots, b_s\}$ all the bases for which n is pseudo-prime, i.e.,

$$b_i^{n-1} \equiv 1 \bmod n \text{ for all } i = 1, \ldots, s \,. \tag{4.9}$$

Since

$$b^{n-1} \equiv c \not\equiv 1 \bmod n \tag{4.10}$$

for some $c \in \mathbb{Z}_n^*$, we obtain, by multiplying (4.9) with (4.10), where $i = 1, \ldots, s$ that

$$c \equiv b_i^{n-1} b^{n-1} \equiv (b_i b)^{n-1} \bmod n \,.$$

Hence, n is not a pseudo-prime to all the bases $\{b_1 b, \ldots, b_s b\}$. Consequently, there are at least as many bases for which n is not a pseudo-prime as there are bases for which n is pseudo-prime. ∎

Now, if we knew that for all odd composite numbers n there should exist at least one number $b \in \mathbb{Z}_n^*$ such that n is not a pseudo-prime to the base b, we could easily design a probabilistic polynomial time algorithm for testing primality. But again, unfortunately, there are odd composite numbers n such that $b^{n-1} \equiv 1 \bmod n$ for *all* $b \in \mathbb{Z}_n^*$. These numbers are called ***Carmichael numbers***.

The following theorem establishes fundamental properties of Carmichael numbers. But before providing it, we need one more exercise.

Exercise 16. *Let p be a prime number. Then $\mathbb{Z}_{p^2}^*$ is cyclic.*

Furthermore, a number n is said to be ***square-free*** if there is no square number dividing it.

Theorem 4.11. *Let $n \in \mathbb{N}$ be an odd composite number. Then we have:*

(1) *If there is a square number $q^2 > 1$ dividing n then n is not a Carmichael number.*

(2) *If n is square-free, then n is Carmichael number if and only if $(p-1)$ divides $n-1$ for every prime p dividing n.*

Proof. Assume any number $q^2 > 1$ dividing n, and let $p > 2$ be a prime factor of q. Since $q^2 | n$, we also know that p^2 is dividing n. Moreover, by Exercise 16 we know that $\mathbb{Z}_{p^2}^*$ is cyclic. Let g be a generator of $\mathbb{Z}_{p^2}^*$. Next, we construct a number $b \in \mathbb{Z}_n^*$ such that $b^{n-1} \not\equiv 1 \bmod n$. If we can do that, then n cannot be a Carmichael number.

Let $\tilde{n}$ be the product of all primes $r \neq p$ that divide n. Obviously, $\gcd(p^2, \tilde{n}) = 1$. By the Chinese Remainder Theorem there is a number b such that

$$\begin{aligned} b &\equiv g \bmod p^2 \\ b &\equiv 1 \bmod \tilde{n} \end{aligned}$$

So b is also a generator of $\mathbb{Z}^*_{p^2}$. But we still do not know whether or not $b \in \mathbb{Z}^*_n$. For seeing it is, we show that $\gcd(n, b) = 1$. Suppose the converse, i.e., $1 < d = \gcd(n, b)$.

Case 1. p divides d.

If p divides d, we also know that $p|b$ and since $p^2|(b - g)$, we additionally have $p|(b - g)$. Consequently, $p|g$, too. But this implies $g \notin \mathbb{Z}^*_{p^2}$, a contradiction. Thus, Case 1 cannot happen.

Case 2. p does not divide d.

Consider any prime r dividing n and d simultaneously. Then, $r \neq p$ by assumption. Hence, $r|b$, too, and moreover, $\tilde{n}|(b - 1)$ because of $b \equiv 1 \bmod \tilde{n}$. But $r \neq p$, so $r|\tilde{n}$, too, and thus $r|(b - 1)$. This implies $r = 1$, a contradiction.

This proves $b \in \mathbb{Z}^*_n$. Finally, we have to show that $b^{n-1} \not\equiv 1 \bmod n$. Suppose the converse, i.e., $b^{n-1} \equiv 1 \bmod n$. Since $p^2|n$, we conclude $b^{n-1} \equiv 1 \bmod p^2$, too. But b is a generator of $\mathbb{Z}^*_{p^2}$. Thus, by the Theorem of Euler we get $\varphi(p^2)|(n - 1)$, i.e., $p(p - 1)|(n - 1)$. This means in particular

$$n - 1 \equiv 0 \bmod p \ .$$

On the other hand, by construction we know that $p|n$, and hence

$$n - 1 \equiv -1 \bmod p \ ,$$

a contradiction. Therefore, we have proved $b^{n-1} \not\equiv 1 \bmod n$ and Assertion (1) is shown.

Next, we prove Assertion (2).

Sufficiency: Let $b \in \mathbb{Z}^*_n$; we have to show $b^{n-1} \equiv 1 \bmod n$. Since n is square-free, it suffices to show $p|(b^{n-1} - 1)$ provided $p|n$. Assume $p|n$ and by assumption also $k(p - 1) = n - 1$ for some k. By Theorem 4.8 we have $b^{p-1} \equiv 1 \bmod p$, and consequently

$$1 \equiv 1^k \equiv (b^{p-1})^k \equiv b^{n-1} \bmod p \ .$$

This holds for all prime divisors p of n, and thus the sufficiency follows.

Necessity: Assume $b^{n-1} \equiv 1 \bmod n$ for all $b \in \mathbb{Z}^*_n$. Now, we have to show that $(p - 1)|(n - 1)$ for all primes p with $p|n$. Suppose there is a prime p with $p|n$ such that $(p - 1)$ does not divide $(n - 1)$. Hence, there are numbers k, r such that $(n - 1) = k(p - 1) + r$ and $0 < r < p - 1$. Now, we again construct a $b \in \mathbb{Z}^*_n$ with

$b^{n-1} \not\equiv 1 \bmod n$. Let g be a generator of $\mathbb{Z}_p^*$ and let $\tilde{n} = n/p$. By the Chinese Remainder Theorem there is a number b such that

$$\begin{aligned} b &\equiv g \bmod p \\ b &\equiv 1 \bmod \tilde{n} \end{aligned}$$

Consequently, b is also a generator of $\mathbb{Z}_p^*$. On the other hand,

$$b^{n-1} \equiv b^{k(p-1)+r} \equiv 1^k b^r \equiv b^r \not\equiv 1 \bmod p \ ,$$

since b is generator. Thus, p does not divide $(b^{n-1} - 1)$, and therefore n does not divide $(b^{n-1} - 1)$, too. ∎

In order to have an example, it is now easy to see that 561 is a Carmichael number. We have just to verify that 2, 10, and 16 divide 560.

Exercise 17. *Every Carmichael number is the product of at least 3 distinct primes.*

References

[1] M. Agrawal, N. Kayal, and N. Saxena (2002), PRIMES in P, Manuscript, `http://www.cse.iitk.ac.in/users/manindra/primality.ps` or `http://www.cse.iitk.ac.in/news/primality.pdf`

[2] I. Niven and H.S. Zuckerman (1991), *An Introduction to the Theory of Numbers.* John Wiley and Sons, New York.

[3] R. Solovay and V. Strassen (1977), A fast Monte-Carlo test for primality, *SIAM Journal on Computing* **6**, 84 – 85.

Lecture 5: Testing Primality and Taking Discrete Roots

It advantageous to introduce the following notions.

Definition 5.1 (Legendre symbol).
Let p *be an odd prime, and let* $a \in \mathbb{Z}_p^*$. a *is said to be a* ***quadratic residue*** *modulo* p *if* $x^2 \equiv a \bmod p$ *is solvable in* $\mathbb{Z}_p^*$. *We define the Legendre symbol* $\left(\frac{a}{p}\right)$ *as follows:*

$$\left(\frac{a}{p}\right) = \begin{cases} 1\ , & \text{if } a \text{ is quadratic residue modulo } p \\ -1\ , & \text{otherwise.} \end{cases}$$

If a *is not a quadratic residue modulo* p*, then we call* a *a* ***quadratic nonresidue****.*

The following theorem is needed below.

Theorem 5.1. *Let* p *be an odd prime and let* $g \in \mathbb{Z}_p^*$ *be a generator for* $\mathbb{Z}_p^*$. *Then for all* $a \in \mathbb{Z}_p^*$ *we have:* a *is quadratic residue modulo* p *if and only if* $\operatorname{dlog}_g a$ *is even.*

Proof. Necessity. Let $a \equiv g^{2m} \bmod p$ for some $m > 0$. Then, $b \equiv g^m$ is obviously a solution of $x^2 \equiv a \bmod p$. Thus a is quadratic residue modulo p.

Sufficiency. Let b be a solution of $x^2 \equiv a \bmod p$, and let $m = \operatorname{dlog}_g b$, i.e., $b \equiv g^m \bmod p$. Consequently, $a \equiv g^{2m} \bmod p$. Thus, by Theorem 4.8 we have $\operatorname{dlog}_g a \equiv 2m \bmod (p-1)$. Since $2|(p-1)$, we can conclude $2|\operatorname{dlog}_g a$, too. ∎

The latter theorem directly implies the following corollary.

Corollary 5.2. *Let* p *be an odd prime. Then there are precisely* $(p-1)/2$ *many quadratic residues and* $(p-1)/2$ *many quadratic nonresidues in* $\mathbb{Z}_p^*$.

The following theorem is needed below.

Theorem 5.3. *Let* p *be an odd prime and let* g *be a generator of* $\mathbb{Z}_p^*$. *Then we have* $g^{(p-1)/2} \equiv -1 \bmod p$.

Proof. By Theorem 4.8 we have $\left(g^{(p-1)/2}\right)^2 \equiv g^{p-1} \equiv 1 \bmod p$. Thus, $g^{(p-1)/2}$ is a solution of $x^2 \equiv 1 \bmod p$.

Now, it suffices to show that $x^2 \equiv 1 \bmod p$ possesses precisely two solutions. Clearly, there are at least two solutions, i.e., 1 and -1. Suppose there is a third solution $r \in \{2, \ldots, p-2\}$. Let $\ell = \operatorname{dlog}_g r$. Then $r^2 \equiv g^{2\ell} \equiv 1 \bmod p$. By Theorem 4.8 we thus know that $2\ell \equiv 0 \bmod (p-1)$. Furthermore, $0 < \ell < p-1$, since otherwise $r \equiv 1 \bmod p$. Therefore, if $\ell = k(p-1)/2$, then $k = 1$ and thus $\ell = (p-1)/2$. Consequently, $r \equiv g^{(p-1)/2} \bmod p$. That is, except 1 there is only the solution $g^{(p-1)/2} \bmod p$ of $x^2 \equiv 1 \bmod p$. Thus, $g^{(p-1)/2} \equiv -1 \bmod p$ must hold. ∎

The next theorem provides one way to compute the Legendre symbol.

Theorem 5.4. *Let* p *be an odd prime and let* $a \in \mathbb{Z}_p^*$, *then*

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \bmod p\ .$$

Proof. We distinguish the following cases.

Case 1. $\left(\frac{a}{p}\right) = 1$

Consequently, there exists a $b \in \mathbb{Z}_p^*$ such that $b^2 \equiv a \bmod p$. Thus

$$a^{(p-1)/2} \equiv b^{p-1} \equiv 1 \bmod p \ .$$

Case 2. $\left(\frac{a}{p}\right) = -1$

Let g be a generator of $\mathbb{Z}_p^*$. Then $a \equiv g^{2m+1} \bmod p$ for some $m \in \mathbb{N}$, since a is a quadratic residue modulo p if and only if the discrete logarithm of a (with respect to g) is even (cf. Theorem 5.1). Hence, using Theorem 5.3 we get

$$a^{(p-1)/2} \equiv g^{(2m+1)(p-1)/2} \equiv g^{m(p-1)} g^{(p-1)/2} \equiv 1 \cdot (-1) = -1 \bmod p \ . \qquad \blacksquare$$

Thus, we can calculate the Legendre symbol by using the procedure *EXP* given in Lecture 4.

The following definition generalizes in some sense the Legendre symbol, but not with respect to the existence of discrete square roots. But it is still providing enough information to design an efficient probabilistic test for primality.

Definition 5.2 (Jacobi symbol).
*Let $Q > 1$ be an odd number, and let $Q = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, where p_i prime for all $i = 1, \ldots, k$ (but not necessarily $p_i \neq p_j$ for $i \neq j$). Let $a \in \mathbb{Z}_Q^*$. The **Jacobi symbol** $\left(\frac{a}{Q}\right)$ is defined as follows:*

$$\left(\frac{a}{Q}\right) = \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \cdot \ldots \cdot \left(\frac{a}{p_k}\right)$$

For having an example, $\left(\frac{2}{9}\right) = \left(\frac{2}{3}\right) \cdot \left(\frac{2}{3}\right) = 1$ but $x^2 \equiv 2 \bmod 9$ is *not* solvable in $\mathbb{Z}_9^*$.

5.1. Solovay and Strassen's Primality Test

Now, we turn our attention to a probabilistic algorithm for testing primality. We shall arrive at a ***Monte Carlo*** algorithm, i.e., a randomized procedure that may produce incorrect results but with bounded error probability. A formal definition of the relevant complexity class will be provided in Lecture 9. As a matter of fact, our algorithm will make only one type of error (see below for details).

The following result is due to Solovay and Strassen [3], the proof given here, however, is not, since I could not completely verify their proof.

Theorem 5.5. *Testing primality can be done in one-sided error probabilistic polynomial time.*

Proof. Let $n \in \mathbb{N}$ be any given number. Clearly, if n is even, this can be trivially recognized. Thus, it suffices to show how to recognize odd primes. Consider the following algorithm:

Algorithm $\mathcal{PT}$

Input: An odd number $n \in \mathbb{N}$.

Method: (1) Choose at random a number $a \in \{1, \ldots, n-1\}$.

(2) Compute $d := \gcd(a, n)$. If $d > 1$ then output ***composite***, and stop. Otherwise, goto (3)

(3) Compute the following quantities:
$\delta = a^{(n-1)/2} \bmod n$
$\varepsilon = \left(\frac{a}{n}\right)$ (the Jacobi symbol)

Output: If $\delta \not\equiv \varepsilon \bmod n$ then output ***composite***, and stop.
If $\delta \equiv \varepsilon \bmod n$ then output ***possibly prime***, and stop.

Next, we prove a series of lemmata which will yield the statement of the theorem.

Lemma 1. *If n is prime, then $\mathcal{PT}$ must output **possibly prime**.*

If n is prime then $\gcd(a, n) = 1$ for all $a \in \{1, \ldots, n-1\}$, and by Theorem 5.4,

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \bmod n \; .$$

Thus, the algorithm $\mathcal{PT}$ necessarily outputs "possibly prime."

Lemma 2. *If n is composite, then $\mathcal{PT}$ outputs **composite** with probability at least* $1/2$.

The main ingredient for proving this lemma is the following claim.

Claim 1. *Let $n \in \mathbb{N}$ be an odd composite number. Then we have for*

$$S = \left\{ a \in \mathbb{Z}_n^* \mid \; a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \bmod n \right\} \quad \textit{that} \;\; |S| \leqslant |\mathbb{Z}_n^*|/2 \; .$$

It is easy to see that S is a subgroup of $\mathbb{Z}_n^*$. Thus, $|S|$ must divide $|\mathbb{Z}_n^*|$, and hence either $|S| = |\mathbb{Z}_n^*|$ or $|S| \leqslant |\mathbb{Z}_n^*|/2$. So it suffices to show that $|S| \neq |\mathbb{Z}_n^*|$. Suppose that $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \bmod n$ for all $a \in \mathbb{Z}_n^*$. Since $\left(\frac{a}{n}\right) = \pm 1$, we conclude $a^{n-1} \equiv 1 \bmod n$ for all $a \in \mathbb{Z}_n^*$, thus n must be a Carmichael number. By Theorem 4.11, n must be square-free and by Exercise 17, n must be the product of at least three different primes. Therefore,

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \cdot \ldots \cdot \left(\frac{a}{p_k}\right) \; ,$$

where $p_1, \ldots, p_k$ are prime numbers and $k \geqslant 3$. Let g be a generator for $\mathbb{Z}_n^*$, and let $\tilde{n} = n/p_1$. By the Chinese Remainder Theorem there exists an $a \in \mathbb{Z}_n^*$ such that

$$a \equiv g \bmod p_1 \tag{5.1}$$
$$a \equiv 1 \bmod \tilde{n} \tag{5.2}$$

In particular, we therefore have $a \equiv 1 \bmod p_j$ for all $j \geqslant 2$, and hence a is quadratic residue modulo p_j for all $j \geqslant 2$. Thus, $\left(\frac{a}{p_j}\right) = 1$ for all $j \geqslant 2$. Moreover, by Theorems 5.3 and 5.4 we have

$$a^{(p_1-1)/2} \equiv g^{(p_1-1)/2} \equiv -1 \equiv \left(\frac{a}{p_1}\right) \bmod p_1 \ .$$

Consequently, $\left(\frac{a}{n}\right) = -1$, too, and therefore $a^{(n-1)/2} \equiv -1 \bmod n$. This implies $a^{(n-1)/2} \equiv -1 \bmod \tilde{n}$. By (5.2) we have $a \equiv 1 \bmod \tilde{n}$, and hence $a^{(n-1)/2} \equiv 1 \bmod \tilde{n}$. This contradiction shows that $S = \mathbb{Z}_n^*$ is impossible. Thus Claim 1 is shown.

Now, if n is composite, then with probability $1/2$ the algorithm $\mathcal{PT}$ chooses an $a \in \{1, \ldots, n-1\}$ such that $\delta \not\equiv \varepsilon$, and therefore, with probability at least $1/2$ the output is ***composite***.

This proves the correctness of the algorithm $\mathcal{PT}$. It remains to evaluate the running time of $\mathcal{PT}$. Everything is clear except the calculation of the Jacobi symbol. If the Jacobi symbol can be computed in polynomial time (as shown below), we are done. ∎

So, it remains to provide an effective method for computing the Jacobi symbol. Note that we cannot reduce the computation of the Jacobi symbol to its definition, since this would require that we know the prime factorization of n. But there is a very nice method which is based on the following theorem and its supplement.

Theorem 5.6 (*Law of Quadratic Reciprocity*). *For all odd numbers* $P, Q \in \mathbb{N}$ *with* $\gcd(Q, P) = 1$ *we have*

$$\left(\frac{Q}{P}\right) = (-1)^{(P-1)(Q-1)/4} \left(\frac{P}{Q}\right) \ .$$

Because of the lack of time, we do not prove this theorem here. There are numerous proofs in print, and we refer the reader to e.g., Niven and Zuckerman [2].

In order to apply Theorem 5.6 successfully, we need the following supplements.

Theorem 5.7. *For all* $a, b \in \mathbb{N}$ *and all odd* $Q \in \mathbb{N}$ *we have*

(1) *If* $a \equiv b \bmod Q$, *then* $\left(\frac{a}{Q}\right) = \left(\frac{b}{Q}\right)$.

(2) $\left(\frac{1}{Q}\right) = 1$

(3) $\left(\frac{-1}{Q}\right) = (-1)^{(Q-1)/2}$

(4) $\left(\frac{ab}{Q}\right) = \left(\frac{a}{Q}\right) \cdot \left(\frac{b}{Q}\right)$

(5) $\left(\frac{2}{Q}\right) = (-1)^{(Q^2-1)/8}$

Again, we refer to Niven and Zuckerman [2] for a proof.

So, the complexity of computing the Jacobi symbol is of the same order as the complexity of the extended Euclidean algorithm (cf. Theorem 3.2).

Next, we provide a method for improving the error probability of the Solovay-Strassen algorithm exponentially while maintaining its polynomial running time.

Corollary 5.8. *If we run the algorithm* $\mathcal{PT}$ k*-times then*

$$\Pr\{k \text{ successive runs output "possibly prime"}\} \leqslant \frac{1}{2^k}$$

provided n *is composite.*

Proof. As we have seen, a composite number may lead to the wrong output ***possibly prime*** with probability $\leqslant 1/2$. Thus, if we run the algorithm $\mathcal{PT}$ k-times we have k independent Bernoulli trials with failure probability $1/2$. Hence, the wanted probability is

$$\Pr\{k \text{ successive runs output "possibly prime"}\} \leqslant \frac{1}{2^k} \ ,$$

since it equals the probability of k successive failures. ∎

Exercise 18. *Apply Theorem 5.7 to compute* $\left(\frac{119}{291}\right)$.

Exercise 19. *Prove the following: Let* $p \in \mathbb{N}$ *be a prime and let* $a \in \mathbb{Z}_p^*$.

(1) $x^2 \equiv -1 \bmod p$ *is solvable if and only if* $p \equiv 1 \bmod 4$.

(2) *If* $p \equiv 3 \bmod 4$ *then either* a *or* $-a$ *is a quadratic residue.*

(3) *If* $p \equiv 1 \bmod 4$ *then either both* a *and* $-a$ *are quadratic residues or both are quadratic non-residues.*

5.2. Taking Discrete Roots

This is a good place to return to the problem of computing discrete roots. The following theorem refers to Berlekamp's algorithm for computing discrete square roots modulo a *prime number*, and may give us a flavor about the techniques used. In general, however, the problem of finding discrete square roots must be considered to be difficult. As a matter of fact, one can prove that finding the *least* solution of $x^2 \equiv a \bmod n$ in positive integers, where $n \in \mathbb{N}$ and $a \in \mathbb{Z}_n^*$, is an $\mathcal{NP}$-hard problem.

5.3. Berlekamp's Procedure for Taking Discrete Square Roots

Before providing the theorem already mentioned, we shortly explain what is meant by ***Las Vegas*** algorithm. A randomized procedure is called Las Vegas algorithm,

if the procedure always correctly computes the desired result (that is, independently from the random choices made). The run time of the procedure, however, does depend on the random choices made. Then, the time complexity of a Las Vegas algorithm on input X is defined to be the expected value with respect to all possible random choices. Note that the following algorithm is due to Berlekamp [1].

Theorem 5.9. *Let $p \in \mathbb{N}$ be a prime and let $a \in \mathbb{Z}_p^*$. Then there is a randomized polynomial time procedure (Las Vegas) to find all solutions of $x^2 \equiv a \bmod p$.*

Proof. Consider the following algorithm:

Input: *An odd prime p and an integer $a \in \mathbb{Z}$ such that $\gcd(a, p) = 1$.*

Output: no solutions *provided a is a quadratic non-residue modulo p;*
all solutions of $x^2 \equiv a \bmod p$, if a is a quadratic residue modulo p.

Method: (1) Compute $\left(\frac{a}{p}\right)$; if $\left(\frac{a}{p}\right) = 1$ then goto (2).
Otherwise, output **no solutions**, and stop.

(2) Choose randomly a $\gamma \in \mathbb{Z}_p^*$ until a number γ has been found such that $\left(\frac{\gamma^2 - a}{p}\right) = -1$.
Compute $\left(x^{\frac{p-1}{2}} - 1\right) \bmod ((x-\gamma)^2 - a)$, and let $\delta(x - \rho)$ be the result of your computation.
Output $(\rho - \gamma)$ and $-(\rho - \gamma)$, and stop.

Before proving the correctness of this procedure, let us consider the following example where the input is $p = 17$ and $a = 8$. Since

$$\left(\frac{8}{17}\right) \equiv 8^8 \equiv 4^4 \equiv (-1)^2 \equiv 1 \bmod 17 ,$$

we see that $x^2 \equiv 8 \bmod 17$ is solvable.

Now, we choose $\gamma = 6$ and easily verify

$$\begin{aligned}\left(\frac{\gamma^2 - a}{p}\right) &= \left(\frac{36-8}{17}\right) = \left(\frac{28}{17}\right) = \left(\frac{11}{17}\right) \\ &\equiv 11^8 \equiv 121^4 \equiv 2^4 \equiv -1 \bmod 17\end{aligned}$$

Next, we have to compute $(x^8 - 1) \bmod ((x-6)^2 - 8)$. As an easy but somehow tedious computation shows, the result is $6521856x - 20674305 \equiv 10x - 10 \equiv 10(x-1) \bmod 17$. Therefore, $\delta = 10$ and $\rho = 1$. Consequently, we output -5 and 5.

Note that, in general, one has to do a bit more for getting δ and ρ. To see this, let us have a look at another computation arising by choosing $\gamma = 8$ instead of 6.

$$\left(\frac{\gamma^2-a}{p}\right) = \left(\frac{64-8}{17}\right) = \left(\frac{56}{17}\right) = \left(\frac{5}{17}\right)$$
$$\equiv 5^8 \equiv 390625 \equiv -1 \bmod 17$$

Now, $(x^8-1) \bmod ((x-8)^2-8) = 33325056x - 171831277 \equiv 7x-6 \bmod 17$. Thus, for finding δ and ρ, first we have to compute the *modular inverse* (this is the new part) of 7 modulo 17, which is 5. Finally, we get:

$$7x-6 \equiv 7x-6\cdot \underbrace{7\cdot 5}_{\equiv 1 \bmod 17} \equiv 7(x-6\cdot 5) \equiv 7(x-13) \bmod 17.$$

Hence, $\delta = 7$ and $\rho = 13$. We again output $\rho-\gamma = 13-8 = 5$, and -5.

It can be easily checked that $25 \equiv 8 \bmod 17$.

Proof. First, we prove the correctness of the procedure given above.

Obviously, if a is a quadratic non-residue modulo p than the Legendre symbol evaluates to -1, and hence the algorithm is correct.

Next, we assume a to be a quadratic residue modulo p. Hence, the Legendre symbol evaluates to 1, and Instruction (2) is executed. Suppose, we have found a number γ such that $\left(\frac{\gamma^2-a}{p}\right) = -1$. Taking into account that $x^2 \equiv a \bmod p$ is solvable, we may conclude that

$$(x-\gamma)^2 - a \equiv 0 \bmod p \tag{5.3}$$

is solvable, too. This is obvious, if you look at $x-\gamma$ as at a new variable. In particular, this statement does *not* depend on the choice of γ. The choice of γ, however, is important for deriving useful information as we shall see in Claim 1 below.

Let ρ and σ be the solution of $(x-\gamma)^2 \equiv a \bmod p$, i.e., we have

$$(\rho-\gamma)^2 - a \equiv 0 \bmod p$$
$$(\sigma-\gamma)^2 - a \equiv 0 \bmod p$$

Next, we prove a very helpful claim.

Claim 1. $\rho\cdot\sigma \equiv \gamma^2 - a \bmod p$

We have the congruence $z^2 - a \equiv 0 \bmod p$, where $z = (x-\gamma)$. By (5.3) we know that this congruence has precisely two solution, say z_1, z_2. Moreover, taking into account that $z_1 \equiv -z_2 \bmod p$ we may conclude

$$z_1\cdot z_2 \equiv -z_1\cdot z_1 \equiv -z_1^2 \equiv -a \bmod p$$

Thus, $z_1\cdot z_2 \equiv -a \bmod p$. In particular, $z_1 = (\rho-\gamma)$ and $z_2 = (\sigma-\gamma)$. Consequently,

$$(\rho-\gamma)(\sigma-\gamma) \equiv -a \bmod p.$$

Therefore, we obtain

$$\rho\sigma - \gamma\sigma - \gamma\rho + \gamma^2 \equiv -a \bmod p \ . \tag{5.4}$$

Furthermore, $\rho - \gamma \equiv -\sigma + \gamma \bmod p$, and thus $-\sigma \equiv \rho - 2\gamma \bmod p$. Consequently, we obtain from (5.4):

$$\begin{aligned} \rho\sigma + \gamma(\rho - 2\gamma) - \gamma\rho + \gamma^2 &\equiv -a \bmod p \\ \rho\sigma + \gamma\rho - 2\gamma^2 - \gamma\rho + \gamma^2 &\equiv -a \bmod p \\ \rho\sigma &\equiv \gamma^2 - a \bmod p \end{aligned}$$

This proves Claim 1.

Taking into account that $\left(\frac{\rho\sigma}{p}\right) = \left(\frac{\rho}{p}\right)\left(\frac{\sigma}{p}\right)$, and $\left(\frac{\gamma^2 - a}{p}\right) = -1$, we conclude that $\left(\frac{\rho}{p}\right) = -\left(\frac{\sigma}{p}\right)$. Without loss of generality, let $\left(\frac{\rho}{p}\right) = 1$. Then, $(x - \rho)$ is a factor of $x^{(p-1)/2} - 1$ modulo p while $(x - \sigma)$ is not. This follows directly from the Euler criterion, since $\rho^{(p-1)/2} \equiv 1 \bmod p$, and thus ρ is a root of the polynomial $x^{(p-1)/2} - 1$ over $\mathbb{Z}_p$. Consequently,

$$\gcd\big((x-\gamma)^2 - a,\, x^{(p-1)/2} - 1\big) = (x - \rho),$$

since ρ and σ are the only solutions of $(x-\gamma)^2 - a \equiv 0 \bmod p$. Hence,

$$\big(x^{(p-1)/2} - 1\big) \bmod (x-\gamma)^2 - a$$

is a polynomial of degree 1 which can be written as $\delta(x-\rho)$. Finally, as we have seen, $(\rho-\gamma)$ is a discrete root of a modulo p. Since there are precisely two roots, $-(\rho-\gamma)$ is the only other solution. This proves the correctness.

Finally, we have to deal with the question of finding γ such that $\left(\frac{\gamma^2-a}{p}\right) = -1$. Note that if $p \equiv 3 \bmod 4$ then $\left(\frac{-a}{p}\right) = -\left(\frac{a}{p}\right) = -1$. Thus, in this case the choice $\gamma = 0$ will always succeed and no randomization is needed.

The remaining case is handled by the following lemma.

Lemma 5.10. *Let $p \in \mathbb{N}$ be prime with $p \equiv 1 \bmod 4$ and let $a \in \mathbb{Z}_p^*$ be such that $\left(\frac{a}{p}\right) = 1$. Then at most half of the elements of $\gamma \in \mathbb{Z}_p^*$ satisfy $\left(\frac{\gamma^2-a}{p}\right) = 1$.*

We need the following claim.

Claim 2. *Let p be a prime number such that $p \equiv 1 \bmod 4$ and let g be a generator for $\mathbb{Z}_p^*$. Furthermore, for $i, j \in \{0,1\}$ let*

$$S_{ij} = \{(x,y) \mid \ x, y \in \mathbb{Z}_{p-1} \text{ and } x \equiv i \bmod 2,\ y \equiv j \bmod 2 \text{ and } g^x + 1 \equiv g^y \bmod p\} \ .$$

Then, $|S_{00}| = \dfrac{p-1}{4} - 1$.

Proof. First, note that the sets S_{00}, S_{01}, S_{10}, S_{11} are pairwise disjoint. Moreover, for each $x \in \mathbb{Z}_{p-1}$ with $x \neq (p-1)/2$ we have $g^x + 1 \neq 0 \bmod p$. Thus, there exists a unique $y \in \mathbb{Z}_{p-1}$ such that $g^x + 1 \equiv g^y \bmod p$. Consequently, we obtain

$$|S_{00}| + |S_{01}| + |S_{10}| + |S_{11}| = p - 2 \; . \tag{5.5}$$

Furthermore, we have

$$|S_{11}| = |S_{10}| \; . \tag{5.6}$$

Condition (5.6) is true, since the mapping

$$(x, y) \mapsto (-x, y - x)$$

between S_{11} and S_{10} is a bijection. For seeing this, note that $g^{2m+1}+1 \equiv g^{2n+1} \bmod p$ implies $g^{2m+1} \cdot g^{-(2m+1)} \equiv 1 \bmod p$. Because of $g^{2m+1} \equiv g^{2n+1} - 1 \bmod p$, we get

$$\begin{aligned} (g^{2n+1} - 1) \cdot g^{-(2m+1)} &\equiv 1 \bmod p \\ g^{2n+1} \cdot g^{-(2m+1)} - g^{-(2m+1)} &\equiv 1 \bmod p \\ g^{2(n-m)} &\equiv g^{-(2m+1)} + 1 \bmod p \; . \end{aligned}$$

Hence, the mapping defined above is bijective.

Next, we show that

$$|S_{10}| = |S_{01}| \; . \tag{5.7}$$

For seeing this, note that $g^{2m+1}+1 \equiv g^{2n} \bmod p$ implies $-g^{2n}+1 \equiv -g^{2m+1} \bmod p$. The latter congruence in turn implies that

$$g^{2n+\frac{p-1}{2}} + 1 \equiv g^{2m+1+\frac{p-1}{2}} \; .$$

Therefore, by taking into account that $(p-1)/2$ is even, we see that the mapping

$$(x, y) \mapsto \left(y + \frac{p-1}{2}, x + \frac{p-1}{2} \right)$$

is a bijection between S_{10} and S_{01}.

Moreover, we can also calculate the following.

$$|S_{11}| + |S_{10}| = \frac{p-1}{2} \; . \tag{5.8}$$

Since $S_{11} \cap S_{10} = \emptyset$, we know that $|S_{11}| + |S_{10}| = |S_{11} \cup S_{10}|$. But

$$S_{11} \cup S_{10} = \{(x, y) \mid \; x, y \in \mathbb{Z}_{p-1} \text{ and } x \equiv 1 \bmod 2 \text{ and } g^x + 1 \equiv g^y \bmod p\} \; ,$$

and therefore,

$$|S_{11} \cup S_{10}| = \frac{p-1}{2} \; .$$

Finally, putting (5.6), (5.7) and (5.8) together yields

$$|S_{11}| = |S_{10}| = |S_{01}| = \frac{p-1}{4} \ .$$

Thus, by (5.5) we can conclude $|S_{00}| = \frac{p-1}{4} - 1$. This proves Claim 2.

Now, we are ready to show the lemma. Let g be any generator for $\mathbb{Z}_p^*$ and let S_{00} be defined with respect to g as in Claim 2. Furthermore, we define

$$R = \left\{ \gamma \in \mathbb{Z}_p^* \mid \left(\frac{\gamma^2 - a}{p}\right) = 1 \right\} \quad \text{and}$$

$$S = \left\{ b \in \mathbb{Z}_p^* \mid \left(\frac{b-a}{p}\right) = 1 \text{ and } \left(\frac{b}{p}\right) = 1 \right\} \ .$$

Claim 3. $|R| = 2|S|$.

Let $b \in S$, then $\left(\frac{b}{p}\right) = 1$. Hence, b is a quadratic residue modulo p. Consequently, $x^2 \equiv b \bmod p$ is solvable and there are two different solutions γ_1 and γ_2, i.e.,

$$\gamma_1^2 \equiv b \bmod p \quad \text{and} \quad \gamma_2^2 \equiv b \bmod p \ .$$

Therefore, from $\left(\frac{b-a}{p}\right) = 1$ we can immediately conclude that $\left(\frac{\gamma_i^2 - a}{p}\right) = 1$ for $i = 1, 2$. But this means that every element from S gives rise to two elements of R. Hence, Claim 3 is shown.

Moreover, since $\left(\frac{a}{p}\right) = 1$ and $p \equiv 1 \bmod 4$ by assumption, we know $(p-1)/2$ is even, and we get $\left(\frac{-a}{p}\right) = 1$, too (cf. the case $p \equiv 3 \bmod 4$). By Theorem 5.1 we have $\operatorname{dlog}_g(-a)$ is even, say $2m = \operatorname{dlog}_g(-a)$. Hence, we arrive at $-a \equiv g^{2m} \bmod p$.

Now, for every $b \in S$ we obtain *mutatis mutandis* that there is an n such that $2n = \operatorname{dlog}_g b$ and an r with $2r = \operatorname{dlog}_g(b-a)$. Therefore, it holds

$$\begin{aligned} b - a &\equiv g^{2n} + g^{2m} \equiv g^{2r} \bmod p \\ g^{2(n-m)} + 1 &\equiv g^{2(r-m)} \bmod p \end{aligned}$$

Next, let $\nu = 2(n-m) \bmod (p-1)$ and $\omega = 2(r-m) \bmod (p-1)$. Then we obviously have $\nu \equiv 0 \bmod 2$, $\omega \equiv 0 \bmod 2$ and $g^\nu + 1 \equiv g^\omega \bmod p$, thus $(\nu, \omega) \in S_{00}$.

Clearly, $b \mapsto (\nu, \omega)$ is an injection from S into S_{00}. Hence, $|S| \leqslant |S_{00}|$ and therefore by Claim 2, $|S| \leqslant (p-1)/4 - 1$.

Finally, using Claim 3 yields $|R| = 2|S| \leqslant (p-1)/2 - 2$. This proves the lemma. ∎

Thus, in case of $p \equiv 1 \bmod 4$ the expected number of random choices required in (2) is bounded by 2. Obviously, all computations in (1) can be done in time polynomial in the lengths of p and a and so can the computation of $\left(\frac{\gamma^2 - a}{p}\right)$ in (2) until an appropriate γ is found. Finally, the computation of $\left(x^{(p-1)/2} - 1\right) \bmod \left((x-\gamma)^2 - a\right)$ can be done by successively squaring x and reducing it modulo $\left((x-\gamma)^2 - a\right)$ as in the computation of $a^m \bmod n$ outlined in procedure ***EXP***. ∎

References

[1] E.R. Berlekamp (1970), Factoring polynomials over large finite fields, *Mathematics of Computation* **24**, 713–745.

[2] I. Niven and H.S. Zuckerman (1991), *An Introduction to the Theory of Numbers.* John Wiley and Sons, New York.

[3] R. Solovay and V. Strassen (1977), A fast Monte-Carlo test for primality, *SIAM Journal on Computing* **6**, 84 – 85.

Lecture 6: Complexity Classes

Next, we turn our attention to complexity classes. In order to do so, we first recall the definition of Turing machines. For the sake of presentation, we start with one-tape Turing machines and time complexity.

6.1. Deterministic One-tape Turing Machines and Time Complexity

For the sake of presentation, first we recall the definition of one-tape deterministic Turing machines.

A one-tape Turing machine consists of an infinite tape which is divided into cells. Each cell can contain exactly one of the tape-symbols. Initially, we assume that all cells of the tape contain the symbol $*$ except those in which the actual input has been written. Moreover, we enumerate the tape cells as shown in Figure 6.1.

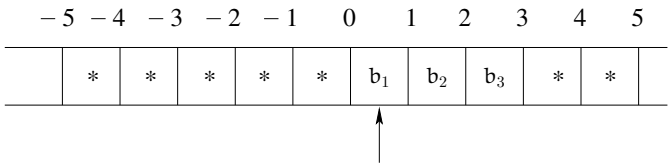

Figure 6.1: The tape of a Turing machine with input $b_1b_2b_3$.

Furthermore, the Turing machine possesses a read-write head. This head can observe one cell at a time. Additionally, the machine has a finite number of states it can be in and a set of instructions it can execute. Initially, it is always in the start state z_s. Then, the machine works as follows. When in state z and reading tape symbol b it writes tape symbol b' into the observed cell, changes its state to z' and moves the head either to the left (denoted by L) or to the right (denoted by R) or does not move the head (denoted by N) provided (z, b, b', m, z') is in the instruction set of the Turing machine, where $m \in \{L, N, R\}$. The execution of one instruction is called ***step***. When the machine reaches a distinguished state z_f (the final state), it stops. Thus, formally, we can define a Turing machine as follows. In the following, for any set S, we write $|S|$ to denote its cardinality.

Definition 6.1. $M = [B, Z, A]$ *is called* ***deterministic one-tape Turing machine*** *if* B, Z, A *are non-empty finite sets such that* $B \cap Z = \emptyset$ *and*

(1) $|B| \geqslant 2$ ($B = \{*, |, \ldots\}$) (*tape-symbols*),

(2) $|Z| \geqslant 2$ ($Z = \{z_s, z_f, \ldots\}$) (*set of states*),

(3) $A \subseteq Z\backslash\{z_f\} \times B \times B \times \{L, N, R\} \times Z$ (*instruction set*), *where for every* $z \in Z\backslash\{z_f\}$ *and every* $b \in B$ *there is precisely one 5-tuple* $(z, b, \cdot, \cdot, \cdot)$.

	$*$	$\mid$	b_2	$\dots$	b_n
z_s	$b'Nz_3$				
z_1	$\cdot$				
$\cdot$	$\cdot$				
$\cdot$	$\cdot$				
$\cdot$	$\cdot$				
z_n	$\cdot$				

Figure 6.2: A Turing table

Often, we represent the instruction set A in a table (see Figure 6.2). Let Σ denote any finite alphabet or synonymously, a set of symbols. Then we use Σ^* to denote the free monoid over Σ. In the following we shall use λ to denote the empty string. We set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. Note that $\lambda \neq *$. Any set $L \subseteq \Sigma^*$ is called a ***language***.

Now, we define what does it mean that a Turing machine is accepting a language L.

Definition 6.2. *A language $L \subseteq \Sigma^*$ is **accepted** by Turing machine M if for every string $w \in \Sigma^*$ the following conditions are satisfied.*

If w is written on the empty tape of M (beginning in cell 0) and the Turing machine M is started on the leftmost symbol of w in state z_s then M stops after having executed finitely many steps in state z_f. Moreover,

(1) *if $w \in L$ then the cell observed by M in state z_f contains a $\mid$.*
In this case we also write $M(w) = \mid$.

(2) *If $w \notin L$ then the cell observed by M in state z_f contains a $*$.*
In this case we also write $M(w) = *$.

Of course, in order to accept a language $L \subseteq \Sigma^*$ by a Turing machine $M = [B, Z, A]$ we always have to assume that $\Sigma \subseteq B$.

Moreover, for every Turing machine M we define

$$L(M) = \{w \mid w \in \Sigma^* \ \wedge \ M(w) = \mid\} \ ,$$

and we refer to $L(M)$ as to the ***language accepted by*** M.

Example 1. Let $\Sigma = \{a\}$ and $L = \Sigma^+$.

We set $B = \{*, a, \mid\}$, $Z = \{z_s, z_f\}$ and define A as follows.

$$\begin{aligned} z_s * &\longrightarrow \mid N z_f \\ z_s a &\longrightarrow \mid N z_f \\ z_s \mid &\longrightarrow \mid N z_s \end{aligned}$$

where $zb \longrightarrow b'mz'$ if and only if $(z, b, b', m, z') \in A$. Note that we have included the instruction $z_s \mid \longrightarrow \mid N z_s$ only for the sake of completeness, since this is required

by Definition 6.1. In the following we shall often omit instructions that cannot be executed.

Next, we formally define time complexity for Turing machines. In order to do so, we need the following notations. For any string w, we use $|w|$ to denote the length of w. Furthermore, for every alphabet Σ and $n \in \mathbb{N}$ we set

$$\Sigma^n = \{w \mid w \in \Sigma^* \ \wedge \ |w| = n\} .$$

Definition 6.3. *Let* $M = [B, Z, A]$ *be a any Turing machine, and* $w \in B^*$. *We set*

$$T_M(w) \ = \ \text{number of steps performed by } M \text{ on input } w \text{ when started on the leftmost symbol of } w \text{ in state } z_s \text{ until reaching state } z_f \ .$$

Furthermore, we define $T_M(n) = \max\{T_M(w) \mid w \in \Sigma^n\}$.

Next, let $T\colon \mathbb{N} \to \mathbb{N}$ be any function. Then we define the ***time complexity class*** generated by T as follows.

$$\mathit{Time}(T(n)) \ = \ \{L \mid L \subseteq \Sigma^* \text{ and there is a Turing machine } M \text{ accepting } L \text{ such that } T_M(n) \leqslant T(n) \text{ for all but finitely many } n\} .$$

This is a good place to recall our deterministic Turing Machine accepting the language of all palindromes over the two letter alphabet $\Sigma = \{a, b\}$ from our course *Theory of Computation*. That is, $L_{pal} = \{w \mid w \in \Sigma^*, \ w = \ w^T\}$ and $M = [B, Z, A]$, where $B = \{*, |, \ a, \ b\}$, $Z = \{z_s, z_1, z_2, z_3, z_4, z_5, z_6, z_f\}$ and A is given by the following table.

	a	b	$*$	$\|$
z_s	$*Rz_1$	$*Rz_2$	$\|Nz_f$	$\|Nz_f$
z_1	aRz_1	bRz_1	$*Lz_3$	$\|Nz_f$
z_2	aRz_2	bRz_2	$*Lz_4$	$\|Nz_f$
z_3	$*Lz_5$	$*Nz_f$	$\|Nz_f$	$\|Nz_f$
z_4	$*Nz_f$	$*Lz_5$	$\|Nz_f$	$\|Nz_f$
z_5	aLz_6	bLz_6	$\|Nz_f$	$\|Nz_f$
z_6	aLz_6	bLz_6	$*Rz_s$	$\|Nz_f$

Figure 6.3: Instruction set of a Turing machine accepting L_{pal}

So, this machine remembers the actual leftmost and rightmost symbol, respectively. Then it is checking whether or not it is identical to the rightmost and leftmost symbol, respectively. Thus, one easily sees that the time complexity of this machine is $O(n^2)$.

Of course, we could have designed a machine that works *mutatis mutandis* as the machine above, but which memorizes two symbols each time, or more generally, k symbols. Nevertheless, the resulting time complexity is still $O(n^2)$. Thus, it is only natural to ask whether or not we can do any better, i.e., finding a Turing machine that

accepts L_{pal} but which has time complexity $o(n^2)$. The negative answer is provided by the following theorem which has been found by Jānis Bārzdiņš.

Theorem 6.1. *For every deterministic one-tape Turing machine* M *accepting* L_{pal} *there is a constant* $c_M > 0$ *such that* $T_M(n) \geqslant c_M n^2$ *for all but finitely many* $n \in \mathbb{N}$.

The proof is given in the appendix (cf. Subsection 15.1).

As we already know, every regular language can be accepted by a finite automaton. Thus, for every $L \in \mathcal{REG}$ there is also a Turing machine M accepting it such that $T_M(n) = n$ for all $n \in \mathbb{N}$. Thus, Theorem 6.1 directly implies $L_{pal} \notin \mathcal{REG}$.

So, we already know that time n^2 is enough to accept non-regular languages by using deterministic one-tape Turing machines. But what happens if we add less resources? The following gap-theorem answers this question.

Theorem 6.2. *Let* M *be a deterministic one-tape Turing machine and assume* $T_M(n) = o(n \log n)$. *Then* $L(M) \in \mathcal{REG}$.

We refer the interested reader to the appendix for a proof (cf. Subsection 15.2).

Note that the bound proved in Theorem 6.2 cannot be improved. For seeing this, we recommend solving the following exercise.

Exercise 20. *Consider the language* $L = \{0^n 1^n \mid n \in \mathbb{N}\} \notin \mathcal{REG}$. *Prove that there is a deterministic one-tape Turing machine* M *accepting* L *in time* $n \log n$.

On the one hand, the Theorems 6.1 and 6.2 are very strong. On the other hand, they also show that just having one tape is causing a lot of work which could be avoided if the Turing machine would have more than one tape. The need for more than one tape is also evident if we wish to study space complexity. If we would consider again deterministic one-tape Turing machines, then the problem is ill posed, since we need already n cells to write the input on the tape and thus could not study sublinear space complexity classes. The idea is to consider Turing machines having at least an input-tape with a head that is only allowed to read, and a work-tape with read-write head. But our goal is a bit more far reaching, thus we are going to consider the more general case of having k tapes, where $k \geqslant 2$.

6.2. Space and Time Complexity of Deterministic k-tape Turing Machines

Definition 6.4. *A Turing machine* $M = [B, Z, A]$ *is called* ***deterministic k-tape Turing machine****,* $k \geqslant 2$, *provided* M *has an input-tape with read-only head and* $k-1$ *many work-tapes each of which possesses exactly one read-write head.*

M *works as the previously defined one-taped Turing machine except that now in every step* k *heads are moved and* k *cells (one on each tape) are observed. Thus formally, now we have*

$$A \subseteq Z \setminus \{z_f\} \times B^k \times (B \times \{L, N, R\})^k \times Z \ ,$$

with the restriction that M *is not allowed to write on its input-tape.*

Initially, M is in state z_s (the start state) and all heads are observing the first cell located right to position 0. Thus, we also have to modify the definition of accepting a language. This is done as follows.

Definition 6.5. *A language $L \subseteq \Sigma^*$ is **accepted** by a deterministic k-tape Turing machine M if for every string $w \in \Sigma^*$ the following conditions are satisfied.*

If w is written on the empty input-tape of M beginning in cell 0 and the Turing machine M is started such that the read-only head on the input-tape is put on the leftmost symbol of w and all other heads are put on the first cell located right to position 0 in state z_s then M stops after having executed finitely many steps in state z_f. Moreover,

(1) *if $w \in L$, the cell observed by M on its first work-tape in state z_f contains a $|$. In this case we also write $M(w) = |$.*

(2) *if $w \notin L$, the cell observed by M on its first work-tape in state z_f contains a $*$. In this case we also write $M(w) = *$.*

Again, we use $L(M)$ to denote the language accepted by Turing machine M. Next, we define what is meant by space complexity.

Definition 6.6. *Let M be a deterministic k-tape Turing machine, $k \geqslant 2$, let $w \in \Sigma^*$ be M's input. Then we define the **space complexity** of M on input w to be the number of cells visited by M on its work-tapes when started in its initial configuration as described above until it stops.*

We denote the space complexity of M on input w by $S_M(w)$. Furthermore, for every $n \in \mathbb{N}$ we set

$$S_M(n) = \max\{S_M(w) \mid w \in \Sigma^n\} .$$

The time complexity is defined as before. Finally, we define the following complexity classes. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. We shall refer to f as *bounding function*. Furthermore, we set

$$\begin{aligned}
\mathit{Time}_k(f(n)) &= \{L(M) \mid M \text{ is det. } k\text{-tape Turing machine and } T_M(n) \leqslant f(n)\} \\
\mathit{Space}_k(f(n)) &= \{L(M) \mid M \text{ is det. } k\text{-tape Turing machine and } S_M(n) \leqslant f(n)\} \\
\mathit{TIME}(f(n)) &= \{L(M) \mid M \text{ is det. Turing machine and } T_M(n) \leqslant f(n)\} \\
\mathit{SPACE}(f(n)) &= \{L(M) \mid M \text{ is det. Turing machine and } S_M(n) \leqslant f(n)\}
\end{aligned}$$

Next, we prove a linear speed-up theorem.

Theorem 6.3. *For every constant $c > 0$, $c \in \mathbb{N}$, and every function $f\colon \mathbb{N} \to \mathbb{N}$, such that $c \cdot f(n) \geqslant n$ for all n we have $\mathit{TIME}(f(n)) = \mathit{TIME}(c \cdot f(n))$.*

Proof. We only sketch the proof here. The remaining details are left as an exercise. Given a Turing machine $M = [B, Z, A]$ one can construct a Turing machine M' working c-times faster than M. Turing machine M' simulates in each step c steps of M by using the tape alphabet B^c and the appropriately defined state set. ∎

Thus, in the following it is always sufficient to show that $T_M(n) = O(f(n))$ instead of proving $T_M(n) \leqslant f(n)$.

Next, we define the important notions of (weak) space (abbr. S) and time (abbr. T) constructibility for functions.

Definition 6.7. *Let a function* $f\colon \mathbb{N} \to \mathbb{N}$ *be given.*

(1) *Function* f *is said to be* ***weakly S-constructible****, if there is a deterministic Turing machine* M *such that* $S_M(n) = f(n)$ *and for every* $n \in \mathbb{N}$ *there exists a string* $w \in \Sigma^*$ *with* $|w| = n$ *and* $S_M(w) = f(n)$.

(2) *Function* f *is said to be* ***S-constructible****, if there is a deterministic Turing machine* M *such that for all* $w \in \Sigma^*$ *the equality* $S_M(w) = f(|w|)$ *is fulfilled.* M *is then called a* ***marker*** *for space* $f(n)$.

(3) *Function* f *is said to be* ***weakly T-constructible****, if there is a deterministic Turing machine* M *such that* $T_M(n) = f(n)$ *and for every* $n \in \mathbb{N}$ *there exists a string* $w \in \Sigma^*$ *with* $|w| = n$ *and* $T_M(w) = f(n)$.

(4) *Function* f *is said to be* ***T-constructible****, if there is a deterministic Turing machine* M *such that for all* $w \in \Sigma^*$ *the equality* $T_M(w) = f(|w|)$ *is fulfilled.* M *is then called a* ***clock*** *for time* $f(n)$.

We continue with some examples. The proof of the assertions made is left as exercise. The functions $f(n) = n$ and $f(n) = 2^n$ are both S-constructible and T-constructible.

The sum and product of S-constructible and T-constructible functions are again S-constructible and T-constructible, respectively. Consequently, all polynomials are both S-constructible and T-constructible.

However, the function $f(n) = n \log\log n$ is *not* T-constructible.

Next, we ask how the number of tapes does influence the complexity of Turing machine computations. As we shall see, the answer depends on both the complexity measure considered and the type of the Turing machine (deterministic or nondeterministic).

6.3. Reducing the Number of Tapes

Theorem 6.4. *Let* $f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ *be any bounding function. Then we have*

$$\mathit{TIME}(f(n)) = \mathit{Time}_2((f(n))^2) \ .$$

Proof. Let M be any deterministic k-tape Turing machine, $k > 1$, such that $T_M(n) \leqslant f(n)$ for all $n \in \mathbb{N}$. For showing the theorem, it suffices to construct a deterministic 2-tape Turing machine M' satisfying $L(M') = L(M)$ and $T_{M'}(n) \leqslant c(T_M(n))^2$ for some constant $c > 0$. This is done as follows.

Let Υ be the alphabet used by M on all its $k-1$ work tapes and assume without loss of generality that $\# \notin \Upsilon$. Then M' will use the alphabet $(\Upsilon \times \{0,1\})^{k-1} \cup \{\#\}$ on its work tape. The ith component of the form $\Upsilon \times \{0,1\}$ at position z contains the content of the ith work tape of M at position z, where "1" is used to mark the actual position of the read-write head of M on this tape. The symbol # is used to mark the left and right end of what was written on the work tape of M'. An illustration of this construction is provided in Figure 6.4.

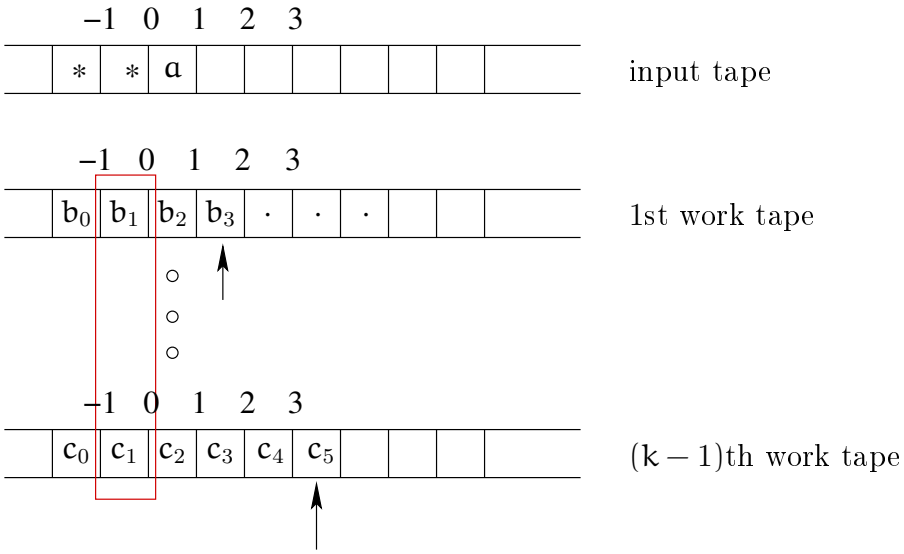

Figure 6.4: Illustration for the use of the new letters

What is inside the red box is becoming the new letter $((b_1,0),\ldots,(c_1,0))$ which is then written in the cell between -1 and 0 of the single work tape of M'.

The simulation of one step of computation performed by M is done by M' as follows. Let w be the input to both M and M'. On the input tape, both machines behave identically. On its work tape M' behaves as follows.

(1) The read-write head on the work tape of M' is reading the whole inscription between the two end markers # from left to right. While doing this, M' memorizes in states the $k-1$ tape contents marked with 1, i.e., the actual head positions on the $k-1$ work tapes of M.

(2) By using the Turing table of M the machine M' is computing the changes to be made, and

(3) then actually performing these changes by moving its read-write head on the work tape from right to left. By doing this, it possibly has also to move its end markers #.

Consequently, the length of the tape inscription on the work tape of M' cannot exceed $2 \cdot T_M(w) + 2$. Hence, one step of M's computation can be simulated in $O(T_M(w))$

steps by M'. At all, $T_M(w)$ many steps of M's computation have to be simulated. Thus, M' performs at most $O((T_M(w))^2)$ many steps. By Theorem 6.3 the assertion of the theorem follows. ∎

Note that the same construction also works if we wish to simulate a deterministic k-tape Turing machine by a deterministic one-tape Turing machine. Thus, we even have the following result.

Corollary 6.5. *Let $f: \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be any bounding function. Then we have $TIME(f(n)) = Time((f(n))^2)$.*

Furthermore, the proof of the latter theorem directly allows the following corollary.

Corollary 6.6. *Let $f: \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be any bounding function. Then we have $SPACE(f(n)) = Space_2(f(n))$.*

Proof. The proof is identical to the proof of Theorem 6.4, since the simulation given there is not increasing the amount of space needed, i.e., $S_{M'}(w) \leqslant S_M(w)$. ∎

Consequently, when studying the amount of space needed to accept non-regular languages, it suffices to deal with 2-tape Turing machines. Recalling a bit automata theory, we directly get the following lemma.

Lemma 6.7. *For every regular language L there is a deterministic 2-tape Turing machine M such that $L = L(M)$ and $S_M(n) = 0$ for all $n \in \mathbb{N}$.*

Again, it is only natural to ask how much space is needed for accepting non-regular languages. Answering this question reveals a further surprise. For showing the theorem below and several other results, we need the following definition.

Definition 6.8. *Let M be a Turing machine. A **macro state** of M is a tuple containing the following*

(1) *the head position on the input tape of M,*

(2) *the actual state of M,*

(3) *for every work tape of M the inscription of all cells visited so far and the actual position of the read-write head.*

*If (1) is omitted, then we refer to the resulting tuple as **configuration** of M.* Note that the definition of configuration, when applied to one-tape Turing machines means that we just record the actual state of the machine.

Theorem 6.8. *Let M be any deterministic 2-tape Turing machine such that $S_M(n) = o(\log\log n)$. Then $L(M)$ is regular.*

For a proof of the latter theorem, we refer the interested reader to the appendix (cf. Subsection 15.3).

Next, we turn our attention to complexity hierarchies for deterministic Turing machines.

6.4. Deterministic Complexity Hierarchies

Before dealing with complexity hierarchies it is meaningful to ask whether or not we can improve Theorem 6.4. This is indeed the case as the following theorem shows.

Theorem 6.9. *Let* $f: \mathbb{N} \to \mathbb{N}$ *be any bounding function. Then we have*

$$TIME(f(n)) = Time_3((f(n)) \cdot \log(f(n))) \ .$$

We omit the proof of Theorem 6.9 here, since there are several more important theorems we want to deal with in this course. We have to economize our time, too.

Next, we have to establish the following fundamental theorem.

Theorem 6.10. *Assuming an appropriate enumeration of all deterministic Turing machines the following holds:*

(1) *There is a universal deterministic Turing machine* U *for all deterministic Turing machines* $(M_i)_{i\in\mathbb{N}}$ *such that*

$$L(U) = \{bin(i) * w \mid w \in L(M_i),\ i \in \mathbb{N}\} \ ,$$

and for every $i \in \mathbb{N}$ *there is a constant* c_i *such that for all* $w \in \Sigma^*$ *the condition* $T_U(|bin(i) * w|) < c_i \cdot T_{M_i}(|w|) \cdot \log T_{M_i}(|w|)$ *is fulfilled.*

(2) *There is a universal deterministic Turing machine* U *for all deterministic Turing machines* $(M_i)_{i\in\mathbb{N}}$ *such that*

$$L(U) = \{bin(i) * w \mid w \in L(M_i),\ i \in \mathbb{N}\} \ ,$$

and for every $i \in \mathbb{N}$ *there is a constant* c_i *such that for all* $w \in \Sigma^*$ *the condition* $S_U(|bin(i) * w|) < c_i \cdot S_{M_i}(|w|)$ *is fulfilled.*

Proof. The existence of universal Turing machines is a fundamental result that has been proved in our course *Theory of Computation*. Now, for showing the theorem, the enumeration is chosen in a way such that the code of M_i at the universal machine is just $bin(i)$. Moreover, the universal deterministic Turing machine U possesses an input tape and at least two work tapes. Any fixed number $k \geqslant 2$ of work tapes will do.

On input $bin(i) * w$, the universal machine U simulates the deterministic Truing machine M_i on input w step by step. Assuming the coding is appropriately chosen, for doing this, it suffices that U reads in each simulation step the "programming code" $bin(i)$, memorizes the actions to be performed and performs the step to be simulated.

Hence, the constant c_i is obtained by counting the number of steps the universal machine needs for simulating one step of M_i. Furthermore, the simulation has to incorporate the reduction of work tapes to the previously fixed number k. By Theorem 6.9, this reduction requires that in the deterministic case $T_{M_i}(w) \cdot \log(T_{M_i}(w))$ steps have to be simulated. Thus, Part (1) of the theorem follows.

For showing Part (2), the only difference is the application of Corollary 6.6 for the tape reduction in the deterministic case. Note that this corollary also causes the improvement in Part (2) concerning the space complexity needed by the universal deterministic acceptor U. ∎

Theorem 6.11. *Let* $f, g: \mathbb{N} \to \mathbb{N}$ *be any bounding functions such that* g *is* T*-constructible and* $f(n) \log f(n) = o(g(n))$*. Then we have*

$$\mathit{TIME}(f(n)) \subset \mathit{TIME}(g(n)) \ .$$

Proof. Since the assumption implies $f(n) < f(n) \log f(n) < g(n)$ for all but finitely n, the inclusion $\mathit{TIME}(f(n)) \subseteq \mathit{TIME}(g(n))$ is obvious. We have to show that there is a language $L_d \in \mathit{TIME}(g(n)) \setminus \mathit{TIME}(f(n))$. This is done by *diagonalization*.

We construct the wanted language L_d by using the universal acceptor U from Theorem 6.10 and by defining a deterministic Turing machine M as follows. Then, L_d contains all strings w, $w = 0^k \mathit{bin}(i)$ that are accepted by M and nothing else.

On input w, $w = 0^k \mathit{bin}(i)$, the machine M works as U on input $\mathit{bin}(i) * w$. Moreover, M uses a clock for $g(n)$ and rejects its input if U does not stop within $g(w)$ many steps. If U does stop within $g(w)$ many steps, then M rejects the input if U accepts and accepts if U rejects.

Thus, by construction we already know $L_d \in \mathit{TIME}(g(n))$. Next, suppose there is deterministic Turing machine M_i such that

$$L_d = L(M_i) \quad \text{and} \quad T_{M_i}(n) \leqslant f(n) \ . \tag{6.1}$$

By Theorem 6.10 we may conclude that for input $0^k \mathit{bin}(i)$ the following holds

$$\begin{aligned} T_U(\mathit{bin}(i) * 0^k \mathit{bin}(i)) &\leqslant c_i \cdot T_{M_i}(0^k \mathit{bin}(i)) \cdot \log\left(T_{M_i}(0^k \mathit{bin}(i))\right) \\ &\leqslant c_i \cdot f(0^k \mathit{bin}(i)) \cdot \log\left(f(0^k \mathit{bin}(i))\right) \ . \end{aligned}$$

Moreover, if k is sufficiently large, we also have

$$T_U(\mathit{bin}(i) * 0^k \mathit{bin}(i)) \leqslant g(0^k \mathit{bin}(i)) \ .$$

Hence, Theorem 6.10 and the construction of M imply that

$$\begin{aligned} 0^k \mathit{bin}(i) \in L(M_i) \quad &\text{iff} \quad 0^k \mathit{bin}(i) \in L(M) \quad (* \text{ since } L(M_i) = L_d = L(M) *) \\ &\text{iff} \quad \mathit{bin}(i) * 0^k \mathit{bin}(i) \notin L(U) \quad (* \text{ by construction of } M *) \\ &\text{iff} \quad 0^k \mathit{bin}(i) \notin L(M_i) \quad (* \text{ by definition of } L(U) *) \ . \end{aligned}$$

Thus, we have obtained the contradiction $0^k \mathit{bin}(i) \in L(M_i)$ iff $0^k \mathit{bin}(i) \notin L(M_i)$. This contradiction is caused by our supposition (6.1). Consequently, our supposition is false. Since there is obviously a deterministic Turing machine M_i with $L_d = L(M_i)$, we must conclude that every machine M_i with $L_d = L(M_i)$ has to satisfy $T_{M_i}(n) > f(n)$. Therefore, we directly arrive at $L_d \notin \mathit{TIME}(f(n))$. ∎

Applying the same ideas as in the proof of Theorem 6.11 we directly get the following hierarchy theorem for deterministic space complexity.

Theorem 6.12. *Let* g *be any* S*-constructible function and let* $f(n)$ *be any space bound such that* $f(n) = o(g(n))$*. Then we have*

$$SPACE(f(n)) \subset SPACE(g(n)) \ .$$

The hierarchy theorems already proved do yield only infinite deterministic complexity time and space hierarchies provided there are arbitrarily complex T-constructible and S-constructible functions. Therefore, we should prove the following exercise.

Exercise 21. *For every general recursive function* $f\colon \mathbb{N} \to \mathbb{N}$ *there exist general recursive functions* g, $g'\colon \mathbb{N} \to \mathbb{N}$ *such that*

(1) $f(n) < g(n)$ *and* $f(n) < g'(n)$ *for all* $n \in \mathbb{N}$*, and*

(2) g *is* T*-constructible, and* g' *is* S*-constructible.*

After having studied deterministic Turing machines in some more detail, it is time to turn our attention to nondeterministic Turing machines formally defined below.

6.5. Nondeterministic k-Tape Turing Machines

Definition 6.9. *A Turing machine* $M = [B, Z, A]$ *is called* ***nondeterministic k-tape Turing machine****,* $k \geqslant 1$*, provided* $B \cup Z = \emptyset$*, and*

(1) $B = \{*, |, \ldots\}$ *is a finite set such that* $|B| \geqslant 2$*,*

(2) $Z = \{z_s, z_f, \ldots\}$ *is a finite set such that* $|Z| \geqslant 2$*,*

(3) $A\colon Z \setminus \{z_f\} \times B^k \to \wp\left((B \times \{L, N, R\})^k \times Z\right) \setminus \{\emptyset\}$

and M *has an input-tape with read-only head (read-write head iff* $k = 1$*) and* $k - 1$ *work-tapes (none iff* $k = 1$ *) each of which possesses exactly one read-write head.*

Again, in every step M *moves* k *heads and observes* k *cells (one on each tape). Also, we make the restriction that* M *is not allowed to write on its input-tape iff* $k > 1$*. Initially,* M *is in state* z_s *(the start state) and all heads are observing the first cell located right to position* 0*. What* M *is writing into the cells it is observing and which move the heads make is decided nondeterministically by choosing one possibility from the set of allowed possibilities (cf. Condition (3)).*

Looking at Definitions 6.4 and 6.9, we see that the main difference to a deterministic Turing machine is Condition (3). For a deterministic Turing machine we had

$$\begin{aligned} A \ &\subseteq Z \setminus \{z_f\} \times B^k \times (B \times \{L, N, R\})^k \times Z \ , \text{ while now we have} \\ A \ &: Z \setminus \{z_f\} \times B^k \to \wp\left((B \times \{L, N, R\})^k \times Z\right) \setminus \{\emptyset\} \end{aligned}$$

That is, a nondeterministic Turing machine possesses in each step a set of possible continuations. From this set, one possibility is chosen nondeterministically.

Initially, M is in state z_s (the start state) and all heads are observing the first cell located right to position 0. But now, on one and the same input there are many possible computations that can be performed by M. Thus, we also have to modify the definition of accepting a language. As before, we assume $\Sigma \subseteq B$ to be the input alphabet, where in particular $* \notin \Sigma$. This is done as follows.

Definition 6.10. *A language $L \subseteq \Sigma^*$ is **accepted** by a nondeterministic k-tape Turing machine M if for every string $w \in \Sigma^*$ the following conditions are satisfied.*

*If w is written on the empty input-tape of M (beginning in cell 0) and M is started such that the read-only head on the input-tape is put on the leftmost symbol of w and all other heads are put on the first cell located right to position 0 in state z_s then, if $w \in L$, **there is a possible computation path** such that M stops after having executed finitely many steps in state z_f and the cell observed by M on its first work-tape in state z_f contains a $|$. In this case we also write $M(w) = |$.*

In contrast to a deterministic Turing machine, if $w \notin L$ then a nondeterministic Turing machine M is *not* supposed to deliver any information. We do not even require M to stop on inputs $w \notin L$. By $L(M)$ we denote the language accepted by M.

Next, we define time and space complexity for nondeterministic Turing machines.

Definition 6.11. *Let M be a nondeterministic Turing machine and $w \in \Sigma^*$. We define $T_M(w)$ to be the **minimum number of steps** executed by M on input w among all its accepting computations, if $w \in L(M)$, and the **minimum number of steps** executed by M among all its computation executed on input w if $w \notin L(M)$.*

*We define $S_M(w)$ to be the **minimum number of all cells** on M's work tapes visited by the read-write heads of M on input w among all its accepting computations if $w \in L(M)$ and **minimum number of all cells** on M's work tapes visited by the read-write heads of M on input w among all its computations if $w \notin L(M)$. Both functions T and S remain undefined if there is no computation of M on input w that stops. Furthermore, we set*

$$\begin{aligned} T_M(n) &= \max\{T_M(w) \mid |w| = n\} \\ S_M(n) &= \max\{S_M(w) \mid |w| = n\} \end{aligned}$$

Finally, we define the resulting complexity classes as follows.

$$\begin{aligned} \mathit{NTime}_k(f(n)) &= \{L(M) \mid M \text{ is nondet. } k\text{-tape Turing mach. and } T_M(n) \leqslant f(n)\} \\ \mathit{NSpace}_k(f(n)) &= \{L(M) \mid M \text{ is nondet. } k\text{-tape Turing mach. and } S_M(n) \leqslant f(n)\} \\ \mathit{NTIME}(f(n)) &= \{L(M) \mid M \text{ is nondet. Turing machine and } T_M(n) \leqslant f(n)\} \\ \mathit{NSPACE}(f(n)) &= \{L(M) \mid M \text{ is nondet. Turing machine and } S_M(n) \leqslant f(n)\} \end{aligned}$$

Note that for nondeterministic Turing machines with a constructible time or space bound, one can always require that they stop on every input. For realizing this, one has simply to combine them with a clock and marker, respectively. Nevertheless, one usually does not get information concerning both $w \in L$ and $w \notin L$, respectively.

Lecture 7: More about Complexity Classes

Within this lecture, we aim to show hierarchy results for nondeterministic complexity classes. This is much more complicated than for the deterministic case. First, we have to look again at tape reductions.

7.1. More about Tape Reductions

We ask whether or not we can improve Theorem 6.4 or Theorem 6.9 for nondeterministic Turing machines. Surprisingly, allowing two work tapes for nondeterministic Turing machines is already sufficient to simulate any nondeterministic Turing machine without increasing the time complexity.

Theorem 7.1. *Let* $f: \mathbb{N} \to \mathbb{N}$ *be any bounding function. Then we have*

$$NTIME(f(n)) = NTime_3((f(n))) \ .$$

Proof. Let any nondeterministic k-tape Turing machine $M = [B, Z, A]$ be given such that $k > 3$. If $k \leqslant 3$, the theorem is obvious. Now, we construct a nondeterministic 3-tape Turing machine M^* with $L(M^*) = L(M)$ and $T_{M^*}(w) = O(T_M(w))$ for all w.

On its first work tape M^* is using the tape alphabet B and on its second work tape the tape alphabet $Z \times B^k$. On input w, the machine M^* works as follows.

(1) M^* writes nondeterministically an arbitrary finite number of symbols on its second work tape. Note that these symbols are from $Z \times B^k$, i.e., $(k+1)$-tuples.

(2) The ith symbol $(z^i, b^1, \ldots, b^k)$ written on the second work tape is interpreted as the actual situation machine M is in its ith step, i.e., it is assumed to be in state z^i and observing the symbols $b^1, \ldots, b^k$ on its input tape and its $k-1$ work tapes.

(3) Using the Turing table of M, the machine M^* checks deterministically whether or not the sequence of symbols on its second work tape is an accepting computation of M on input w. If it is, M^* accepts the input w, otherwise the input w is rejected.

The check is done in $k-1$ stages. In stage i, $1 \leqslant i \leqslant k-1$, the ith work tape of M is completely documented on the first work tape of M^*.

Hence, M^* can find an accepting computation path in time $k \cdot T_M(w) = O(T_M(w))$ if and only if $w \in L(M)$. Finally, the theorem follows by Theorem 6.3. ∎

Dealing with nondeterministic space complexity does not require any new idea. Looking at the proof of Theorem 6.4, we immediately realize that the construction performed there can be also applied in the nondeterministic case. Since this tape reduction does not increase the amount of space needed, we get the following corollary.

Corollary 7.2. *Let* $f\colon \mathbb{N} \to \mathbb{N}$ *be any bounding function. Then we have*

$$NSPACE(f(n)) = NSpace_2(f(n)) \ .$$

Next, we reformulate Theorem 6.10 for nondeterministic Turing machines.

Theorem 7.3. *Assuming an appropriate enumeration of all nondeterministic Turing machines the following holds:*

(1) *There is a universal nondeterministic Turing machine* V *for all nondeterministic Turing machines* $(M_i)_{i\in\mathbb{N}}$ *such that*

$$L(V) = \{bin(i) * w \mid w \in L(M_i),\ i \in \mathbb{N}\} \ ,$$

and for every $i \in \mathbb{N}$ *there is a constant* c_i *such that for all* $w \in \Sigma^*$ *the condition* $T_V(|bin(i) * w|) < c_i \cdot T_{M_i}(|w|)$ *is fulfilled.*

(2) *There is a universal nondeterministic Turing machine* V *for all nondeterministic Turing machines* $(M_i)_{i\in\mathbb{N}}$ *such that*

$$L(V) = \{bin(i) * w \mid w \in L(M_i),\ i \in \mathbb{N}\} \ ,$$

and for every $i \in \mathbb{N}$ *there is a constant* c_i *such that for all* $w \in \Sigma^*$ *the condition* $S_V(|bin(i) * w|) < c_i \cdot S_{M_i}(|w|)$ *is fulfilled.*

Proof. The proof is done analogously to the demonstartion of Theorem 6.10. The only difference is the reduction of tapes. By Theorem 7.1, now the reduction does not require additional steps, and hence Part (1) follows.

Part (2) is shown analogously. The only difference is the application of Corollary 7.2 instead of Theorem 7.1. ∎

The proof methods developed so far are not directly applicable to the nondeterministic case. Looking at the proof of Theorem 6.11, we see that we have taken advantage of the fact that deterministic complexity classes are closed under complement. More precisely, if $\mathcal{C}$ is a complexity class, then we set

$$\text{co-}\mathcal{C} = \{L \mid L \subseteq \Sigma^*,\ \Sigma^* \setminus L \in \mathcal{C}\} \ .$$

We say that a complexity class is ***closed under complement*** if co-$\mathcal{C} \subseteq \mathcal{C}$.

While it was fairly easy to complement deterministic complexity classes by just returning 'no' for all accepting computations, and 'yes' to all rejecting computations, this method does not work for nondeterministic complexity classes. The fact that a particular computation did not succeed, is not guarantee that others do not, so the strategy above could put some strings both in the language and in its complement. Thus, additional work is needed here. For space complexity there is the famous Immerman-Szelepcsényi Theorem stating that $NSPACE(f(n)) = \text{co-}NSPACE(f(n))$ for all bounding functions f satisfying $f(n) \geqslant \log n$ for all $n \in \mathbb{N}$ (see below).

But for nondeterministic time complexity classes no such theorem is known. The best one has is the almost trivial result given by the following exercise.

Exercise 22. *Let* $f: \mathbb{N} \longrightarrow \mathbb{N}$ *be any bounding function. Then we have*

$$\text{co-}NTIME(f(n)) \subseteq TIME\left(\bigcup_{c>0} 2^{cf(n)}\right) \subseteq NTIME\left(\bigcup_{c>0} 2^{cf(n)}\right) .$$

However, we can show a hierarchy theorem for nondeterministic time complexity classes.

7.2. A Complexity Hierarchy for Nonderterministic Time

We continue with the following lemma that will help us to circumvent the problem whether or not nondeterministic time complexity classes are closed under complement.

Lemma 7.4. *For every nondeterministic Turing machine* M_i *there is a nondeterministic Turing machine* M_j *such that for all* $w \in \Sigma^*$

$$\begin{aligned} L(M_j) &= \{w |\ bin(j) * w \in L(M_i)\} \text{ and} \\ T_{M_j}(w) &\leqslant c_i \cdot T_{M_i}(bin(j) * w) . \end{aligned}$$

Proof. By Theorem 7.1, it suffices to consider an enumeration $(M_i)_{i\in\mathbb{N}}$ of all 3-tape nondeterministic Turing machines. For each given $i \in \mathbb{N}$, we construct the wanted j as follows.

(1) First, we consider the general recursive function $r: \mathbb{N} \to \mathbb{N}$ induced by the given enumeration $(M_i)_{i\in\mathbb{N}}$ having the property that

$M_{r(k)}$, on input any $w \in \Sigma^*$, first writes $bin(k)$ on its first work tape, and then works as M_k does.

(2) Next, for each given i, we construct a special machine M_I. The machine M_I, on input any $w \in \Sigma^*$ on its input tape and $bin(k)$ on its first work tape, works as M_i on input $bin(r(k)) * w$.

We define $j = r(I)$. By construction, M_j works as follows. Since $M_j = M_{r(I)}$, by (1) we get that, on input w, M_j first writes $bin(I)$ on its first work tape and then works as M_I does.

By (2) we can conclude that M_j works as M_i does on input $bin(r(I)) * w$. But since $j = r(I)$, this means that M_j works as M_i does on input $bin(j) * w$. Hence,

$$L(M_j) = \{w |\ bin(j) * w \in L(M_i)\} ,$$

giving the first part of the lemma.

Furthermore, we have

$$T_{M_j}(w) \leqslant c_i \cdot T_{M_i}(bin(j) * w) \ ,$$

since M_j has to execute $T_{M_i}(bin(j) * w)$ many steps plus a constant number. The constant results from the number of steps needed to write $bin(I)$ on its first work tape and then returning the head into its starting position. This number is for all w the same, thus we can compensate this additional constant by the factor c_i. This proves the second part of the lemma. ∎

Using the latter lemma, we can prove the following hierarchy theorem for nondeterministic time complexity classes. Before presenting it, we also need to prove that there are arbitrarily complex languages acceptable by nondeterministic Turing machines. This is done via the following exercise.

Exercise 23. *Let g be any T-constructible bounding function and let $M = [B, Z, A]$ be any nondeterministic Turing machine such that $T_M(w) \leqslant g(|w|)$. Then there is a deterministic Turing machine M' such that $L(M) = L(M')$ and $T_{M'}(w) \leqslant g(|w|) \cdot k^{g(|w|)}$, where $k = 2 \cdot |B|$.*

Having the result of Exercise 23 on hand, Theorem 6.11 directly implies via Exercise 21 that there are arbitrarily complex languages for nondeterministic time and nondeterministic space.

Theorem 7.5. *Let $g: \mathbb{N} \to \mathbb{N}$ be any T-constructible bounding function and let $f: \mathbb{N} \to \mathbb{N}$ be any bounding function such that $f(n) = o(g(n))$ and $f(n+1) = O(g(n))$. Then we have*

$$NTIME(f(n)) \subset NTIME(g(n)) \ .$$

Proof. The simple inclusion $NTIME(f(n)) \subseteq NTIME(g(n))$ is obvious by the asymptotic relation $f(n) < g(n)$ for all but finitely many $n \in \mathbb{N}$. Thus, it remains to show that the inclusion is proper. For seeing this, we construct a language L_{nd} such that $L_{nd} \in NTIME(g(n)) \setminus NTIME(f(n))$. This is done by using the universal acceptor V from Theorem 7.3 and by defining a nondeterministic Turing machine M as follows. For all $i \in \mathbb{N}$ and $w \in \Sigma^*$ we define:

On input $bin(i) * w$ machine M alternatively simulates one step a clock for g and one step of the universal acceptor V both on input $bin(i) * w$.

If V on input $bin(i) * w$ does not stop within $g(|bin(i) * w|)$ many steps, the input is rejected.

If V on input $bin(i) * w$ does stop within at most $g(|bin(i) * w|)$ many steps, then M accepts its input iff V accepts $bin(i) * w$.

By construction, $L(M) \in NTIME(g(n))$. Next, suppose there is a nondeterministic Turing machine M_i such that

$$\begin{aligned} L_{nd} &= L(M_i) \text{ and} \\ T_{M_i}(n) &\leqslant f(n) \ . \end{aligned} \tag{7.1}$$

Now, the contradiction is obtained via a machine M_j that respects a universal time bound but can accept arbitrary complex languages.

Let any nondeterministic machine $\tilde{M}$ be given. We first construct a machine M' and then the wanted machine M_j.

The nondeterministic machine M' works as follows.

(1) On input any y, the machine M' first checks whether or not $y = bin(j) * w * 0^k$ for some $j, k \in \mathbb{N}$ and some $w \in \Sigma^*$. If not, then y is rejected. If y does, Instruction (2) is executed.

(2) M' simulates k steps of $\tilde{M}$'s work on input w. If $\tilde{M}$ accepts w within this simulation, then M' accepts its input y.

If $\tilde{M}$ does not accept w within this simulation, then M' simulates M_i on input $bin(j) * w * 0^{k+1}$. Recall that M_i is the machine supposed to accept L_{nd}.

Now, by construction and (7.1) we have

$$T_{M'}(y) = 2|y| + f(|y| + 1) = O(g(|y|)) .$$

Next, we apply Lemma 7.4 to the machine M'. Thus, there exist a Turing machine M_j such that

$$\begin{aligned} L(M_j) &= \{w * 0^k \mid bin(j) * w * 0^k \in L(M')\} \text{ and} \\ T_{M_j}(w * 0^k) &\leqslant c \cdot T_{M'}(bin(j) * w * 0^k) = O(g(|bin(j) * w * 0^k|)) . \end{aligned} \tag{7.2}$$

We continue by showing $w * 0^k \in L(M_j)$ iff $w \in L(\tilde{M})$.

(1) For $k \geqslant T_{\tilde{M}}(w)$ we have by construction

$$w * 0^k \in L(M_j) \text{ iff } bin(j) * w * 0^k \in L(M') \text{ iff } w \in L(\tilde{M}) .$$

For the latter "iff" it should be noted that either M' accepts w while simulating $\tilde{M}$ for k steps in Instruction (2) or it cannot accept w at all, since later it simulates M_i on input $bin(j) * w * 0^{k+1}$.

(2) Assume for some $k \leqslant T_{\tilde{M}}(w)$ that $w * 0^k \in L(M_j)$ iff $w \in L(\tilde{M})$. Then, for $k - 1$ we obtain

$$\begin{aligned} w * 0^{k-1} \in L(M_j) \quad &\text{iff} \quad bin(j) * w * 0^{k-1} \in L(M') \\ &\text{iff} \quad bin(j) * w * 0^k \in L(M_i) \\ &\text{iff} \quad bin(j) * w * 0^k \in L(V) \\ &\text{iff} \quad w * 0^k \in L(M_j) \quad (* \text{ by Theorem 7.3 } *) \\ &\text{iff} \quad w \in L(\tilde{M}) \quad (* \text{ by the induction assumption } *) \end{aligned}$$

Thus, we have shown $w * 0^k \in L(M_j)$ iff $w \in L(\tilde{M})$. But now we get a contradiction to (7.2), since M_j obeys the time bound given in (7.2) while $\tilde{M}$ can be chosen arbitrarily. Hence $L(\tilde{M})$ can be arbitrarily complex by Exercise 23. ∎

7.3. The Immerman-Szelepcsényi Theorem

We are going to prove a nondeterministic space hierarchy theorem. Fortunately enough, after three decades of failure, in 1988 it could be proved that nondeterministic space is closed under complement. Even more interestingly, in 1988 two proofs have been published independently of one another by Immerman [1] and Szelepcsényi [2], respectively. So, we continue with the Immerman-Szelepcsényi Theorem.

Theorem 7.6. co-$NSPACE(f(n)) = NSPACE(f(n))$ *for every* S*-constructible bounding function* f *satisfying* $f(n) \geqslant \log n$ *for all* $n \in \mathbb{N}$.

Proof. Let $L \in NSPACE(f(n))$ be witnessed, without loss of generality, by a nondeterministic 2-tape Turing machine $M = [B, Z, A]$ with $S_M(n) \leqslant f(n)$ for all $n \in \mathbb{N}$.

Let C_x denote the set of $f(|x|)$-space bounded configurations of M on input x. Note that

$$|C_x| \leqslant |Z| \cdot \frac{|B|^{f(|x|)+1} - 1}{|B| - 1} \cdot f(|x|) \ .$$

Furthermore, we write $C_x(\tau)$ to denote the set of configurations that can be reached by M on input x within precisely τ steps of computation. For the ease of notation, we write $c_x(\tau)$ to denote the cardinality of $C_x(\tau)$, i.e., $c_x(\tau) = |C_x(\tau)|$. Without loss of generality, we can also assume that there is a uniquely determined accepting configuration C_a and that M works in each of its computations on input x exactly t steps, where $t \leqslant |C_x|$. We leave the proof of this statement as an exercise.

Then we have

$$\begin{aligned} M \text{ does not accept } x \quad &\text{iff} \quad C_a \notin C_x(t) \\ &\text{iff} \quad C_x(t) \text{ contains } c_x(t) \text{ many rejecting final configurations.} \end{aligned}$$

The latter observation can be generalized to

$$C_a \notin C_x(\tau) \qquad \text{iff} \quad C_x(\tau) \text{ contains } c_x(\tau) \text{ many configurations } C' \neq C_a \ . \tag{7.3}$$

Note that the time needed to check Condition (7.3) is at least $c_x(\tau)$ which is clearly exponential in τ.

We have to construct a nondeterministic Turing machine M' that accepts $\overline{L}$ $(= \Sigma^* \setminus L)$ in space at most $f(n)$. The idea for the construction of M' is given by the equivalence (7.3) displayed above. Having the additional information $c_x(\tau)$ on hand, one can decide nondeterministically in space $f(|x|)$ whether or not $C_a \in C_x(\tau)$.

Let $C_0(x)$ denote the initial configuration of M on input x. If $x \notin \overline{L}$, the positive answer is easily obtained by guessing a sequence of configurations $C_1, \ldots, C_\tau$ such that $C_1 = C_0(x)$ and C_{i+1} is reachable by M in one step as well as $C_\tau = C_a$. Of course, we have to do this guessing iteratively. That is, we always have at most two configurations on the work tape. This can be easily achieved by deleting C_{i-1} as soon as C_i is generated and verified. Again, for the sake of notation, we write $C_i \vdash C_{i+1}$ if C_{i+1} can be reached by M in one step.

For the negative answer, we successively check for $C' \in C_x$ with $C' \neq C_a$ whether $C' \in C_x(\tau)$. The number of positive answers is counted (that is the reason, why we need $f(n) \geqslant \log n$). If we find $c_x(\tau)$ many such configurations C', we must conclude $C_a \notin C_x(\tau)$.

All what is left is to formalize this idea appropriately. For doing this, we first define a Procedure `REACH`. On input $\tau, c, C_1, \ldots, C_\tau$, `REACH` tries to decide whether at least one of the configurations $C_1, \ldots, C_\tau$ is in $C_x(\tau)$. If `REACH` does not succeed, it returns "?". The parameter c is used to estimate the cardinality of $C_x(\tau)$.

Procedure `REACH`$(\tau, c, C_1, \ldots, C_\tau)$

Input: τ, $c \in \mathbb{N}$, $\{C_1, \ldots, C_\tau\} \subseteq C_x$

Output: `true, false` or `?`

Method: *number* $:= 0$;

`for` $C \in C_x$ in lexicographical order `do begin`

guess nondeterministically a computation

$C_0(x) \vdash D_1 \vdash \cdots \vdash D_\tau$ of length τ;

`if` $D_\tau = C$ `then` *number* $:=$ *number* $+ 1$;

`if` $D_\tau \in \{C_1, \ldots, C_\tau\}$ `then return` `REACH`$(\tau, c, C_1, \ldots, C_\tau) =$ `true`;

`end`;

`if` *number* $= c$ `then return` `REACH`$(\tau, c, C_1, \ldots, C_\tau) =$ `false`;

`if` *number* $< c$ `then return` `REACH`$(\tau, c, C_1, \ldots, C_\tau) =$?;

`end REACH`

Now, we can show the following lemma.

Lemma 7.7. *If procedure* `REACH` *is called with* $c = c_x(\tau)$ *and* `REACH` *returns* `true` *or* `false` *then this answer is correct.*

Proof. First, assume `true` is returned. This can happen if and only if `REACH` has found an initial segment $C_0(x) \vdash D_1 \vdash \cdots \vdash D_\tau$ of a computation such that $D_\tau \in \{C_1, \ldots, C_\tau\}$. So, the answer is correct.

Next, assume `false` is returned. Then we have *number* $= c = c_x(\tau)$, i.e., `REACH` has found c many different configurations none of which is equal to one of the C_is. By definition of $c_x(\tau)$ this implies that all configurations in $C_x(\tau)$ have been tested, thus $C_x(\tau) \cap \{C_1, \ldots, C_\tau\} = \emptyset$. Hence, the answer is correct. ∎

If `REACH` returns ? for $c = c_x(\tau)$, then not all configurations $C \in C_x(\tau)$ have been found in the `for`-loop. Therefore, one can neither conclude $C_i \notin C_x(\tau)$ nor $C_i \in C_x(\tau)$.

Since *number* $\leqslant |C_x|$, one can represent *number* in space $O(f(|x|))$. Moreover, every configuration $C \in C_x(\tau)$ needs space at most $f(|x|)$, and thus `REACH` can be realized by a nondeterministic Turing machine that obeys the space bound $f(n)$.

Next, we have to deal with the question how the numbers $c_x(\tau)$ are determined. This is done iteratively by the following procedure COUNT.

Procedure COUNT(τ, c)

Input: τ, $c \in \mathbb{N}$

Output: $d \in \mathbb{N}$ or ?

Method: $d := 0$;

for $C \in C_x$ in lexicographical order do begin

compute the direct predecessors $C_1, \ldots, C_\tau$ of C;

$z :=$ REACH$(\tau - 1, c, C_1, \ldots, C_\tau)$;

if $z =$ true then $d := d + 1$;

if $z =$? then return COUNT$(\tau, c) =$?;

end;

return COUNT$(\tau, c) := d$;

end COUNT.

Lemma 7.8. *If* COUNT *is called with* $c = c_x(\tau - 1)$ *and returns a natural number* d*, then* $d = c_x(\tau)$.

Proof. If COUNT is not stopping with output "?" received from REACH, then, by Lemma 7.7, REACH correctly answers each question whether one of the predecessors C_i of C belongs to $C_x(\tau - 1)$. Thus, after completion of the for-loop, the value of d coincides with $|C_x(\tau)|$, since every $C \in C_x(\tau)$ must have a predecessor in $C_x(\tau - 1)$. ∎

Moreover, it is easy to verify that COUNT can be executed in space $O(f(|x|))$. Now, we can put it all together to decide nondeterministically whether or not M, on input any x does *not* possess an accepting computation.

On input x do the following:

$c_x(0) := 1$;

for $\tau = 1, \ldots, t$ do

if $c_x(\tau - 1) \neq$? then $c_x(\tau) :=$ COUNT$(\tau, c_x(\tau - 1))$;

else $c_x(\tau) :=$? ;

if $c_x(\tau) \neq$? then $z :=$ REACH$(t, c_x(t), C_a)$;

if $z =$ false then accept x.

This program part can be executed by a nondeterministic Turing machine M' in space $O(f(|x|))$, too. Furthermore, if M' is accepting an input x, then there must be a computation path of M' such that neither REACH nor COUNT do return "?". Hence, COUNT correctly computes the values $c_x(\tau)$ for $\tau = 1, \ldots, t$ and REACH verifies that $C_a \notin C_x(t)$. Consequently, $x \notin L$, thus $x \in \overline{L}$ and we are done.

Finally, if $x \notin L$, then for every configuration $C \in C_x(\tau)$ there is partial computation $C_0(x) \vdash D_1 \vdash \cdots \vdash D_\tau = C$ that can be guessed while processing

$$\texttt{REACH}(\tau, c_x(\tau), C_1, \ldots, C_\tau) \ .$$

Consequently, `COUNT` is correctly computing the values $c_x(1), \ldots, c_x(t)$, too. But then $\texttt{REACH}(t, c_x(t), C_a)$ must return `false`. Therefore, M' is accepting x.

Thus, we have shown that $\overline{L} \in \text{co-}NSPACE(f(n))$ implies $\overline{L} \in NSPACE(f(n))$, i.e.,

$$\text{co-}NSPACE(f(n)) \subseteq NSPACE(f(n))$$

for all S-constructible bounding functions f satisfying $f(n) \geqslant \log n$ for all $n \in \mathbb{N}$. Hence, we can conclude

$$\text{co-}NSPACE(f(n)) = NSPACE(f(n))$$

by Exercise 24 below.

Note that the S-constructibility of f ensures that $s = f(|x|)$ can be determined by M' and thus, also t is known. ∎

A closer look at the proof of Theorem 7.6 shows that an easy modification at the end of the proof even allows to prove the following more general corollary.

Corollary 7.9. co-$NSPACE(f(n)) \subseteq NSPACE(f(n))$ *for all bounding functions* f *satisfying* $f(n) \geqslant \log n$ *for all* $n \in \mathbb{N}$.

We leave it as an exercise to show Corollary 7.9. Furthermore as already mentioned in the proof of Theorem 7.6, the following result always holds.

Exercise 24. *For every complexity class* $\mathcal{C}$ *we have, if* co-$\mathcal{C} \subseteq \mathcal{C}$ *then* co-$\mathcal{C} = \mathcal{C}$.

Now, it is easy to prove the following hierarchy result for nondeterministic space.

Theorem 7.10. *Let* $f, g: \mathbb{N} \longrightarrow \mathbb{N}$ *be any two bounding functions such that* $f(n) = o(g(n))$, $g(n) \geqslant \log n$ *for all* $n \in \mathbb{N}$ *and* g *is* S*-constructible. Then*

$$NSPACE(f(n)) \subset NSPACE(g(n)) \ .$$

The proof is left as an exercise.

One final remark is in order here. In all our hierarchy theorems, we have always required the bounding function of the larger complexity class to be constructible. This condition cannot be dropped. For non-constructible functions one can prove nice gap theorems. For giving you a flavor, we finally include the following exercise here.

Exercise 25. *There are recursive functions* $f, g: \mathbb{N} \longrightarrow \mathbb{N}$ *such that*

(1) $TIME(f(n)) = TIME(2^{f(n)})$

(2) $SPACE(g(n)) = SPACE(2^{2^{g(n)}})$.

After having seen that nondeterministic space is closed under complement provided that the bounding function f satisfies $f(n) \geqslant \log n$ for all $n \in \mathbb{N}$, we can answer the question whether or not the context-sensitive languages (abbr. $\mathcal{CS}$) are closed under complement. This is done by characterizing $\mathcal{CS}$ in terms of complexity theory.

7.4. $\mathcal{CS}$ and Linear Bounded Automata

First, we define linear bounded automata. There are several possibilities to define linear bounded automata from which we have chosen the one that should make our proofs as easy as possible.

Definition 7.1. *A **linear bounded automaton** is a nondeterministic one-tape Turing machine such that*

(1) *its input alphabet contains two special symbols ¶ and \$ which are used to mark the leftmost and rightmost position of the tape, respectively, that can be reached by the head.*

(2) *The head can neither replace ¶ nor \$.*

So, a linear bounded automaton can only use the space between the two markers ¶ and \$. The input is written between these markers. Please note that the end-markers are written on the tape together with the input but are themselves not considered to belong to the input. Now, taking Theorem 10.11 from [3] into account, we can easily show the following.

Theorem 7.11. *For every context-sensitive language L there is a linear bounded automaton M such that $L = L(M)$.*

Proof. First, we divide the tape into two traces. The upper trace contains the input w which will not be changed during the computation. The lower trace is actually used to simulate a possible derivation of w provided it exists. Both traces are uniformly marked by ¶ and \$.

Let $L = L(G)$, where $G = [T, N, \sigma, P]$ is without loss of generality length increasing. (cf. [3], Theorem 10.11). If the input is λ, M just stops in its accepting state. If the input is not λ, M starts its computation by writing σ on the leftmost place in the lower trace. Then M guesses nondeterministically a derivation. If the derivation yields the input w, the input string w is accepted. Otherwise, M stops without accepting the input.

To formalize this idea, we have to think about a way how this nondeterministic guessing is performed. This is done by guessing a production and a position on the tape. From the production it can be derived how many cells are needed to replace the nonterminal at the guessed position. If the guessed position does not contain a nonterminal in the lower trace, nothing is done and a new guess is made. If there is a nonterminal at the guessed position, the substring to the right of the position is moved by the number of cells needed to replace the nonterminal. If there is enough

space to perform this shift, then the replacement is done. If there is not enough space, then M stops without accepting.

Finally, if a string has been derived that is precisely as long as the input, it is compared with the input in the upper trace. If both strings are identical, M accepts. Otherwise it stops without accepting. Since $\mathcal{G}$ is length increasing, there cannot be a derivation for w that exceeds the space between the markers ¶ and \$. Therefore, M accepts a string w if and only if $w \in \mathsf{L}(\mathcal{G})$. ∎

Interestingly enough, the converse direction is also true. That is, every language accepted by a linear bounded automaton is context-sensitive. Thus, we have the following result.

Theorem 7.12. *If* $\mathsf{L} = \mathsf{L}(\mathsf{M})$ *for a linear bounded automaton, then* $\mathsf{L} \in \mathcal{CS}$.

The proof is not too hard and left as an exercise.

Thus we could characterize the context-sensitive languages as the set of all those languages that are accepted by a linear bounded automaton. Since a linear bounded automaton uses space $|w| + 2$ on its tape, we can apply the Immerman-Szelepcsényi Theorem and obtain the affirmative answer to the problem whether or not $\mathcal{CS}$ is closed under complement.

Corollary 7.13. $\mathcal{CS}$ *is closed under complement.*

7.5. Important Complexity Classes

Now, we are ready to introduce the following important complexity classes.

$$\begin{aligned}
\mathcal{L} &= SPACE(\log n) \\
\mathcal{NL} &= NSPACE(\log n) \\
\mathcal{P} &= TIME(n^{O(1)}) = \bigcup_{c \in \mathbb{N}} TIME(n^c) \\
\mathcal{NP} &= NTIME(n^{O(1)}) = \bigcup_{c \in \mathbb{N}} NTIME(n^c) \\
\mathcal{PSPACE} &= SPACE(n^{O(1)}) = \bigcup_{c \in \mathbb{N}} SPACE(n^c) \\
\mathcal{NPSPACE} &= NSPACE(n^{O(1)}) = \bigcup_{c \in \mathbb{N}} NSPACE(n^c)
\end{aligned}$$

One remark is mandatory here. Quite often one uses $\mathcal{P}$ to denote the set of all partial recursive functions over $\mathbb{N}$, while we use it here to denote the set of all languages acceptable by a deterministic Turing machine within polynomial time. Though this is for sure an overloading of this particular notation, it is quite common in the literature and it should always be clear from context what is meant.

Then, we can immediately make the following observations.

Proposition 7.1.

(1) $\mathcal{L} \subseteq \mathcal{NL}$,

(2) $\mathcal{P} \subseteq \mathcal{NP}$,

(3) $\mathcal{PSPACE} \subseteq \mathcal{NPSPACE}$,

(4) $\mathcal{P} \subseteq \mathcal{PSPACE}$,

(5) $\mathcal{NP} \subseteq \mathcal{NPSPACE}$.

Moreover, we already know the following proper inclusion.

Theorem 7.14. $\mathcal{L} \subset \mathcal{PSPACE}$

Proof. The inclusion is obvious. The proper inclusion is a direct consequence of Theorem 6.12. ∎

Moreover, one is often interested in *simultaneously* bounding the time and space resources needed to accept a language. Thus, we shall also study the following complexity classes.

$$\begin{aligned} \mathit{DTISP}(f(n), g(n)) &= \{L(M) \mid M \text{ is DTM and } T_M(n) \leqslant f(n) \wedge S_M(n) \leqslant g(n)\} \\ \mathit{NDTISP}(f(n), g(n)) &= \{L(M) \mid M \text{ is NDTM and } T_M(n) \leqslant f(n) \wedge S_M(n) \leqslant g(n)\} \\ \mathcal{PLOPS} &= \mathit{DTISP}\left(n^{O(1)}, (\log n)^{O(1)}\right) \end{aligned}$$

Algorithms belonging to the complexity class $\mathcal{PLOPS}$ are considered to be practically realizable using current computer technology. Nevertheless, even this statement has to be read with care, since in practice one has also to ensure that the exponents and constants involved are moderate.

References

[1] N. IMMERMAN (1988), Nondeterministic space is closed under complementation. *SIAM J. Computing* **17**, 935–938.

[2] R. SZELEPCSÉNYI (1988), The method of forced enumeration for nondeterministic automata. *Acta Inf.* **26**, 279–284.

[3] T. ZEUGMANN (2007), Course notes on theory of computation. Technical Report TCS-TR-B-07-2, Division of Computer Science, Hokkaido University.

Lecture 8: More about Important Complexity Classes

We finished the last lecture by introducing several important complexity classes. Within this lecture, we study them in some more detail.

8.1. Fundamental Inclusions

First we show that a logarithmic space bound can always be combined with a polynomial time bound.

Theorem 8.1.

(1) $\mathcal{L} \subseteq \mathit{DTISP}(n^{O(1)}, \log n)$,

(2) $\mathcal{NL} \subseteq \mathit{NDTISP}(n^{O(1)}, \log n)$.

Proof. Let M be a k-tape Turing machine such that $S_M(n) = O(\log n)$. That is, there exists a constant $c > 0$ such that $S_M(n) \leqslant c \cdot \log n$. Recalling that a macro state consists of the head position on the input tape, the actual state of M, and for every work tape the content of all cells visited as well as the actual head position, we can bound the total number of *macro states* by

$$n \cdot |Z| \left(|B|^{c \cdot \log n} c \cdot \log n\right)^{k-1} \leqslant n^{O(1)} \ . \tag{8.1}$$

Here n is the number of possibilities for the head position on the input tape. Moreover, the machine M can be in at most $|Z|$ many different states. On each work tape, the head can have visited at most $c \cdot \log n$ many positions, and thus it can write only strings of the same length on each of its work tapes. Since we have $|B|$ many symbols, and $k-1$ many work tapes, the formula displayed above follows. Recall that

$$\log_{|B|} n \ = \ \frac{\ln n}{\ln |B|} \ \text{ and } \ \log n = \frac{\ln n}{\ln 2} \ ,$$

and hence

$$c \cdot \log n \ = \ c \cdot \frac{\ln n}{\ln 2} = c \cdot \frac{\ln |B|}{\ln 2} \cdot \log_{|B|} n = \hat{c} \cdot \log_{|B|} n$$

Consequently, $|B|^{c \cdot \log n} = |B|^{\hat{c} \cdot \log_{|B|} n} = n^{\hat{c}}$. Furthermore, $\log n \leqslant n$ for $n \geqslant 1$, and therefore

$$\left(|B|^{c \cdot \log n} c \cdot \log n\right)^{k-1} \ \leqslant \ \left(c \cdot n^{\hat{c}+1}\right)^{k-1} = c^{k-1} n^{\tilde{c}}$$

for $\tilde{c} = (\hat{c}+1)(k-1)$. Additionally, for $n \geqslant 2$ there exists an m such that $n^m \geqslant |Z|$. Thus, the estimate (8.1) is proved and the theorem follows for the deterministic case.

For the nondeterministic case, we have additionally to take into consideration that every polynomial is T-constructible. Hence, the nondeterministic Turing machine M can be combined with a clock for the particular polynomial time arising without changing the language accepted. We leave it as an exercise to show that the amount of space needed to implement the clock can be logarithmically bounded. Thus, (2) follows. ∎

The polynomial time bound just proved is essential to show the following inclusion.

Theorem 8.2. $\mathcal{NL} \subseteq \mathcal{P}$.

Proof. Let M be a nondeterministic Turing machine such that $S_M(n) \leqslant c \cdot \log n$ for a suitably chosen constant $c > 0$. We have to construct a deterministic Turing machine $\tilde{M}$ that accepts the same language as M and that uses at most polynomial time, i.e., $T_{\tilde{M}}(n) \leqslant n^{O(1)}$.

The machine $\tilde{M}$ works as follows.

(1) $\tilde{M}$ uses the same input w as M does. First, it writes all possible macro states of M on its first work tape. By Theorem 8.1 we already know that there are only polynomially many macro states.

(2) Next, $\tilde{M}$ marks the one macro state of all the macro states written on its first work tape in which M starts its computation.

(3) Then, $\tilde{M}$ marks all macro states that can be reached in one step by M from one of those already marked. If this increases the number of marked macro states, $\tilde{M}$ repeats Stage (3).

    Otherwise, $\tilde{M}$ checks whether or not there is marked macro state in which M accepts the current input. If this is the case, $\tilde{M}$ accepts the current input. Otherwise, the input is rejected.

By construction, we directly obtain $L(M) = L(\tilde{M})$. Finally, there are only polynomially many macro states to be read one time in each execution of Stage (3). Hence, $\tilde{M}$ executes at most $n^{O(1)}$ many steps. ∎

Note that the deterministic Turing machine provided in the proof above also uses $n^{O(1)}$ many tape cells on its work tape.

Next, we aim to show that $\mathcal{NPSPACE} \subseteq \mathcal{PSPACE}$. This will be done by proving the following more general theorem.

Theorem 8.3. *Let $f(n)$ be an S-constructible function satisfying $S(n) \geqslant \log n$. Then, we have $NSPACE(f(n)) \subseteq SPACE((f(n))^2)$.*

We shall prove this theorem in the appendix in a more general context.

So far, we have obtained the following insight:

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{NPSPACE} = \mathcal{PSPACE} \ .$$

Moreover, by Theorem 7.14 we also know that at least one of the inclusions must be *proper*. It is conjectured that *all* inclusions are proper. However, despite many efforts, so far none of the inclusions could be proved to be proper nor could any equality be shown.

Right now, we turn our attention to the problem whether or not there are problems which can be considered to be the most difficult ones within their corresponding

complexity class. As we shall see, such problems do exist. We shall refer to these problems as to *complete problems*.

The importance of complete problems is easily explained. The efficient solution of a *complete* problem for $\mathcal{NL}$ or $\mathcal{NP}$ could be used to efficiently solve *all* of the problems in $\mathcal{NL}$ or $\mathcal{NP}$, respectively. Thus, establishing the existence of complete problems allows one to focus all efforts to solve any such complete problem in a way such that the resulting algorithm obeys the constraints of the complexity class in question. For example, showing for one $\mathcal{NL}$-complete problem to be in $\mathcal{L}$ then establishes the equality $\mathcal{L} = \mathcal{NL}$.

8.2. Hardness and Completeness

For defining what is meant by complete and hard problem, respectively, we introduce the notion of *reduction*. For that purpose, we have to modify the deterministic Turing machine model in a way such that strings can be computed as output. We use Σ to denote any fixed finite alphabet and Σ^* for denoting the free monoid over Σ.

Definition 8.1. *A function* $f\colon \Sigma^* \to \Sigma^*$ *is said to be* ***log-space computable*** *if there exists a deterministic Turing machine* M_f *satisfying the following properties:*

(1) M_f *possesses an input tape with a two-way read-only head, an output tape with a one-way write-only head and finitely many work tapes each of which has a two-way read-write head.*

(2) *On input* w *the machine* M_f *computes* $f(w)$ *and writes it on its output tape. While performing this computation,* M_f *uses on each of its work tapes at most* $O(\log n)$ *many tape cells.*

Log-space computable functions have an interesting property which is stated as an exercise.

Exercise 26. *Show that for each log-space computable function the condition* $|f(w)| \leqslant |w|^{O(1)}$ *is satisfied.*

Next, we define reductions.

Definition 8.2. *Let* $A, B \subseteq \Sigma^*$ *be any two decidable languages.* A *is said to be* ***log-space reducible*** *to* B *(written* $A \leqslant_{\log} B$*) if there exists a log-space computable function* f *such that for all* $w \in \Sigma^*$ *the condition* $w \in A$ *if and only if* $f(w) \in B$ *is satisfied.*

Exercise 27. *Let* $L_1, L_2, L_3 \subseteq \Sigma^*$ *be any decidable languages. Then we have: If* $L_1 \leqslant_{\log} L_2$ *and* $L_2 \leqslant_{\log} L_3$ *then* $L_1 \leqslant_{\log} L_3$*, i.e., log-space reducibility is transitive.*

Now, we are ready to define the notion of hardness and completeness.

Definition 8.3. *Let* $\mathcal{S}$ *be a family of decidable languages over* Σ^* *and let* $L_0 \subseteq \Sigma^*$*.* L_0 *is said to be* ***log-space hard*** *for* $\mathcal{S}$ *if* $L \leqslant_{\log} L_0$ *for every* $L \in \mathcal{S}$*.*

If additionally $L_0 \in \mathcal{S}$ *is satisfied then* L_0 *is said to be* ***log-space complete*** *for* $\mathcal{S}$.

Next, we ask in which sense a language $A \subseteq \Sigma^*$ is easier than a language $B \subseteq \Sigma^*$ provided $A \leqslant_{\log} B$. This is done via the following lemma.

Lemma 8.4. *Let* M *be a Turing machine such that* $S_M(n) \notin o(\log n)$. *If a language* $L \subseteq \Sigma^*$ *is log-space reducible to* $L(M)$ *then there exists a Turing machine* $\tilde{M}$ *such that* $L = L(\tilde{M})$ *and* $S_{\tilde{M}} \in O(S_M(n))$.

Proof. The proof idea is to combine the acceptor Turing machine M with a Turing machine M_f that realizes the log-space translation of L into $L(M)$. But there is a problem. The space bound of M does not allow, in general, to write the result $f(w)$ of the translation of w via M_f on M's work tape(s). Hence, we have to modify M_f appropriately. We define a deterministic Turing machine M'_f as follows.

On input w and input $\mathtt{bin}(k)$ on an auxiliary work tape M'_f works as M_f but writes only the kth symbol of $f(w)$ on its output tape. Since $|f(w)| \leqslant |w|^{O(1)}$ the space bound $O(\log n)$ for M'_f is ensured. M'_f can count all attempts of M_f to write a symbol on its output tape until the kth one is reached which is then executed.

Finally, M is modified in way such that each change of the head position on the input tape of M is accompanied by setting the binary counter to the actual input head position and by computing the symbol to be read by executing M'_f on input w and $\mathtt{bin}(k)$ as described above. ∎

Please note that the condition $S_M(n) \notin o(\log n)$ was essential for proving the latter lemma, since otherwise we could not have used the binary counter.

Lemma 8.4 allows the following corollary.

Corollary 8.5. *Let* $L, L' \subseteq \Sigma^*$ *be any languages.*

(1) *If* $L \in \mathcal{L}$ *and* $L' \leqslant_{\log} L$ *then* $L' \in \mathcal{L}$.

(2) *If* $L \in \mathcal{L}$ *and* $\emptyset \neq L' \neq \Sigma^*$ *then* $L \leqslant_{\log} L'$.

Proof. We leave it as an exercise to prove this corollary. ∎

Consequently, $\mathcal{L}$ constitutes the lowest level of log-space reducibility. It should also be noted that there are a couple of reducibility notion around which have been intensively studied in the literature. We mention here only ***polynomial-time*** reducibility which is defined analogously as log-space reducibility. The only difference to Definition 8.2 is that the function f is now only required to be computable by a deterministic Turing machine obeying a polynomial time bound for its computation time instead of the log-space bound required in Definition 8.1. If a language L_1 is polynomial-time reducible to a language L_2 then we write $L_1 \leqslant_{\text{poly}} L_2$. For getting a better understanding of polynomial-time reducibility, we recommend to solve the following exercise.

Exercise 28. *Let* L_1, L_2 *be any two languages. If* $L_1 \leqslant_{\log} L_2$ *then* $L_1 \leqslant_{\text{poly}} L_2$.

The notion of reducibility also allows one to define an equivalence relation.

Definition 8.4. *Let* $L_1, L_2 \subseteq \Sigma^*$ *be any two decidable languages.* L_1 *and* L_2 *are said to be* ***equivalent*** *with respect to log-space (polynomial-time) reducibility if* $L_1 \leqslant_{\log} L_2$ *and* $L_2 \leqslant_{\log} L_1$ ($L_1 \leqslant_{\text{poly}} L_2$ *and* $L_2 \leqslant_{\text{poly}} L_1$).

If L_1 *and* L_2 *are equivalent with respect to log-space and polynomial-time reducibility then we write* $L_1 \equiv_{\log} L_2$ *and* $L_1 \equiv_{\text{poly}} L_2$*, respectively.*

We leave it as an exercise to show that "$\equiv_{\log}$" and "$\equiv_{\text{poly}}$" are indeed equivalence relations.

However, we shall mainly deal with log-space reducibility within this course. Our next goal is to establish the existence of complete problems for the complexity class $\mathcal{NL}$ defined in the last lecture.

For that purpose, we first define the ***graph accessibility problem*** (abbr. GAP).

GAP:

Input: A directed graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_m\}$ and a distinguished start node v_s and a distinguished end node v_e.

Problem: Does there exist a path between v_s and v_e?

If the graph G is given by its adjacency-list, then the input length n of GAP can be bounded by $O(m^2 \log m)$. Moreover, we can safely assume $n \geqslant m$.

8.3. Properties of the GAP problem

Next, we show GAP to be $\mathcal{NL}$-complete. This is done in two steps, i.e., by first showing GAP to be $\mathcal{NL}$-hard and then GAP to be in $\mathcal{NL}$.

Lemma 8.6. GAP *is* $\mathcal{NL}$*-hard.*

Proof. Let M be a Turing machine such that $S_M(n) \in O(\log n)$ and let w be an input to M. We define a graph $G_w = (V, E)$ as follows.

The nodes of G_w are *all* the macro states of M that can occur under the space bound $S_M(|w|)$. Let v and v' be any two macro states of M (that is, any two nodes of G_w). Then we define $(v, v') \in E$ if and only M can reach macro state v' from macro state v in one step. Without loss of generality, we can assume that the macro state at the beginning of M's computation on input w is uniquely determined. Moreover, again without loss of generality, we can additionally assume that, if M accepts w the the accepting macro state of M is uniquely determined, too.

Now, it is easy to see that the graph G_w is log-space computable from input w, since the number of nodes is uniformly polynomially bounded in $|w|$. Finally, if the nodes of G_w are appropriately numbered, then our construction directly implies

$$w \in L(M) \iff G_w \in \text{GAP} \ .$$

Hence, every language from $\mathcal{NL}$ is log-space reducible to GAP. ∎

Next, we show GAP to be acceptable by a nondeterministic Turing machine.

Lemma 8.7. GAP $\in \mathcal{NL}$.

Proof. Let any graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_m\}$ and a distinguished start node v_s and a distinguished end node v_e be given as input. Let n be the length of the input. As shown above, n can be bounded by $O(m^2 \log m)$. Thus, the space bound $\log n$ is sufficient to store any node number in binary.

The nondeterministic Turing machine M works as follows. First, it stores the number s of the start node v_s. Then, non-deterministically any successor of v_s, say v_i, is chosen (that is, $(v_s, v_i) \in E$), s is erased, and the number i is stored as actual node number.

Next, the process is iterated. That is, assuming j to be the actual node number, any successor of v_j, say v_k, is chosen and its number k is stored as actual node number and j is erased. The storing and erasing is done in a way such that the total amount of space used by M is $O(\log n)$.

The graph G is accepted, if e is reached as actual node number. Otherwise, G is not accepted.

Clearly, if there is a path from v_s to v_e in G, then there is an accepting computation of M on input G. Otherwise, no computation can accept G. ∎

By Definition 8.3, the Lemmata 8.6 and 8.7 directly imply the following corollary.

Corollary 8.8. GAP *is* $\mathcal{NL}$*-complete.*

Moreover, we immediately obtain the following corollary.

Corollary 8.9. *Let* $f(n) \neq o(\log n)$ *be a space bounding function. Then we have:* GAP $\in$ *SPACE*$(f(n))$ *if and only if* $\mathcal{NL} \subseteq$ *SPACE*$(f(n))$.

Further $\mathcal{NL}$-complete problems are studied in the appendix (see Subsection 15.5).

8.4. $\mathcal{NP}$-complete Problems

Now, we turn our attention to the class $\mathcal{NP}$ which contains many important problems. We start with a list of examples for decision problems that turn out to be all in $\mathcal{NP}$. We define these problems here as languages and assume any reasonable encoding of the input.

Let $G = (V, E)$ be an undirected graph. A complete subgraph of size k of G is said to be a ***k-Clique***. Here a graph is said to be complete if every vertex is connected to any other vertex. We set

$$\text{CLIQUE} = \{(G, k) \mid G \text{ possesses a } k\text{-Clique}\} .$$

Let $G = (V, E)$ be an undirected graph. A set $U \subseteq V$ is said to be ***independent*** if $(u, v) \notin E$ for all $u, v \in U$, $u \neq v$. We set

$$\text{INDSET} = \{(G, k) \mid G \text{ possesses an independent set of size } k\} .$$

Now, let $G = (V, E)$ be a directed graph. A ***Hamiltonian path*** is a path visiting all vertices of G exactly ones. We set

$$\text{dHAMILTON} = \{(G, k) |\ G \text{ possesses a Hamiltonian path}\} .$$

A ***vertex cover*** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$. The size of a vertex cover V' is the cardinality of V'. We set

$$\text{VCOVER} = \{(G, k) |\ G \text{ possesses a vertex cover of size } k\} .$$

Subset Sum Problem

Input: a number M and a vector $(a_0, \ldots, a_{n-1}) \in \mathbb{N}^n$.

Problem: Decide whether there exists a vector $(b_0, \ldots, b_{n-1}) \in \{0, 1\}^n$ such that $M = \sum_{j=0}^{n-1} a_j b_j$.

As with any arithmetic problem, it is important to recall that the input integers are coded in binary. Then we define SUBSUM to be the language of all subset sum problems $((a_0, \ldots, a_{n-1}), M)$ for which there is a vector $(b_0, \ldots, b_{n-1}) \in \{0, 1\}^n$ such that $M = \sum_{j=0}^{n-1} a_j b_j$.

Finally, we define the famous satisfiability problem.

Definition 8.5. *Let $F = f(x_1, \ldots, x_n)$ be a Boolean formula consisting of the variables $x_1, \ldots, x_n$ and the Boolean operators $\vee, \wedge, \neg$. F is said to be in ℓ-CNF form if F is in conjunctive normal form and each clause contains precisely ℓ literals. F is said to be **satisfiable** if the exists an assignment $(a_1, \ldots, a_n) \in \{0, 1\}^n$ to the variables $x_1, \ldots, x_n$ such that $F(a_1, \ldots, a_n) = 1$.*

The satisfiability problem is then the language

$$\text{SAT} = \{F |\ F \text{ is a satisfiable formula}\}.$$

Let us ask what all the languages defined in this subsection do have in common. The general pattern is that is presumably very hard to *find* a witness that any of its instances belongs to them. For example, in order to find a satisfying assignment one may have to try all possible assignments, i.e., all 2^n many Boolean vectors $a_1, \ldots, a_n \in \{0, 1\}^n$. The same clearly applies for SUBSUM.

As for dHAMILTON, one may be forced to try all $n!$ permutations of the vertices of $G = (V, E)$, where $|V| = n$ in order to find a Hamiltonian path.

On the other hand, it is for all the languages given above easy to *check* whether or not a witness is given. For instance, for any given assignment one can quickly check whether or not it is satisfying a given Boolean formula by a deterministic Turing

machine. Informally, this property may serve as a rule of thumb for deciding whether or not any given language belongs to $\mathcal{NP}$.

Next, we ask whether or not the class $\mathcal{NP}$ contains an $\mathcal{NP}$-complete language. The affirmative answer has been given by Cook [1], who could show the following important theorem.

Theorem 8.10. SAT *is* $\mathcal{NP}$*-complete.*

We are not going to prove this theorem here, since there are many proofs in the literature. Furthermore, you will not need to prove any problem to be $\mathcal{NP}$-complete by using Cook's [1] original proof technique. Instead, to show the $\mathcal{NP}$-completeness of any other language L it suffices to reduce SAT or any other language known to be $\mathcal{NP}$-complete to L. This approach has been used to show a large number of languages to be $\mathcal{NP}$-complete (cf. Garey and Johnson [2] and numerous resources on the web). Therefore, we are going to exemplify it here, too. Let ℓ-SAT be the language of all Boolean formulae in ℓ-CNF form that are satisfiable. Then, we can show the following.

Theorem 8.11. ℓ-SAT *is* $\mathcal{NP}$*-complete for all* $\ell \geqslant 3$.

Proof. Since SAT is in $\mathcal{NP}$ we obviously have ℓ-SAT $\in \mathcal{NP}$, too. Thus, it suffices to log-space reduce SAT to ℓ-SAT. First, we show that any Boolean formula F can be transformed into a sat-equivalent formula F' in CNF. Here by sat-equivalent we mean that F is satisfiable if and only if F' is satisfiable.

Note that we cannot just transform F into its CNF, since the length of the CNF may be exponential in the length of F, thus violating our requirement to log-space reduce SAT to ℓ-SAT.

For obtaining the desired transformation of F into a sat-equivalent formula F' in CNF, in general we need new auxiliary variables. Here by new we mean that these variables do not occur in F. We proceed as follows. In our first step we transform F into a logical equivalent formula F' by using de Morgan's rules as well as $\neg\neg x \equiv x$. Note the we use both $\neg x$ and $\overline{x}$ to denote negation.

***Step* 1.** Using de Morgan's rules, we transform F into F' such that all negations in F' appear at the variables.

Example: Let $F = \neg(\neg(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_4 \vee (x_3 \wedge \overline{x}_5)))$. Then, in Step 1 we successively obtain:

$$\begin{aligned}
\neg(\neg(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_4 \vee (x_3 \wedge \overline{x}_5))) &\equiv \neg\neg(x_1 \vee x_2 \vee \overline{x}_3) \vee \neg(x_4 \vee (x_3 \wedge \overline{x}_5)) \\
&\equiv (x_1 \vee x_2 \vee \overline{x}_3) \vee \neg(x_4 \vee (x_3 \wedge \overline{x}_5)) \\
&\equiv (x_1 \vee x_2 \vee \overline{x}_3) \vee (\overline{x}_4 \wedge \neg(x_3 \wedge \overline{x}_5)) \\
&\equiv (x_1 \vee x_2 \vee \overline{x}_3) \vee (\overline{x}_4 \wedge (\overline{x}_3 \vee x_5)) \ .
\end{aligned}$$

After a bit of reflection it is easy to see that Step 1 can be realized in log-space.

Let F' be the formula obtained so far. Next, we transform F' into a sat-equivalent CNF. This transformation is based on the following observation. If $F' = F_1 \vee F_2$ and F_1, F_2 are already in CNF, then we can replace F' by

$$(F_1 \vee y) \wedge (F_2 \vee \overline{y}) \ ,$$

where y is a new variable. Clearly, the new formula is sat-equivalent to F'. We refer to this rule as to Rule 1. Furthermore, we need Rule 2 displayed below to transform $F_1 \vee y$ and $F_2 \vee \overline{y}$ into a CNF. This is done as follows. Let $F_i = G_1 \wedge G_2 \wedge \ldots \wedge G_k$. Then $F_i \vee y^\alpha$ is equivalent to

$$(G_1 \vee y^\alpha) \wedge (G_2 \vee y^\alpha) \wedge \ldots \wedge (G_k \vee y^\alpha) \ ,$$

and we have again a conjunction of clauses.

***Step* 2.** Apply Rule 1 and Rule 2 recursively until a CNF is obtained.

Step 2 introduces for every $\vee$ a new variable. Let the result be F''. So, if the original formula F has length m then F'' has length at most $O(m^2)$ and the transformation takes at most $O(m^2)$ steps. Again, it is not too hard to show that Step 2 can be realized in log-space, too. We leave it as an exercise to prove this formally.

Continuing our example, we thus obtain (where $\sim$ denotes sat-equivalence)

$$\begin{aligned} & (x_1 \vee x_2 \vee \overline{x}_3) \vee (\overline{x}_4 \wedge (\overline{x}_3 \vee x_5)) \\ \sim \ & (x_1 \vee x_2 \vee \overline{x}_3 \vee y_1) \wedge (\overline{x}_4 \vee \overline{y}_1) \wedge (\overline{x}_3 \vee x_5 \vee \overline{y}_1) \ . \end{aligned} \tag{8.2}$$

Next, we have to show that any formula in CNF can be transformed into a sat-equivalent formula in ℓ-CNF form. For the sake of presentation we handle here the case $\ell = 3$, only. Consider any clause $C = (z_1 \vee \cdots \vee z_k)$. In dependence on k we replace C by the following formula by using new variables y_i:

$$\begin{aligned} k = 1 \ &: \ (z_1 \vee y_1 \vee y_2) \wedge (z_1 \vee \overline{y}_1 \vee y_2) \wedge (z_1 \vee y_1 \vee \overline{y}_2) \wedge (z_1 \vee \overline{y}_1 \vee \overline{y}_2) \\ k = 2 \ &: \ (z_1 \vee z_2 \vee y_1) \wedge (z_1 \vee z_2 \vee \overline{y}_1) \\ k = 3 \ &: \ (z_1 \vee z_2 \vee z_3) \quad \text{i.e., we do not change } C \\ k > 3 \ &: \ (z_1 \vee z_2 \vee y_1) \wedge (\overline{y}_1 \vee z_3 \vee y_2) \wedge (\overline{y}_2 \vee z_4 \vee y_3) \wedge \\ & \quad \ldots \wedge (\overline{y}_{k-4} \vee z_{k-2} \vee y_{k-3}) \wedge (\overline{y}_{k-3} \vee z_{k-1} \vee z_k) =: \tilde{C} \ . \end{aligned}$$

Clearly, these formulae can be computed in log-space.

The sat-equivalence of the formulae obtained can be seen as follows. In case $k = 1$, the four clauses can be simultaneously satisfied if and only if z_1 is assigned the value 1, since independently of the assignments for y_1, y_2, in one of the four clauses the resulting evaluation is 0. Analogously, one directly sees that in case $k = 2$ the two clauses can be simultaneously satisfied if and only if z_1 or z_2 is assigned the value 1. For $k = 3$ nothing has to be shown.

Finally, for $k > 3$ it remains to show that C is satisfiable if and only if $\tilde{C}$ is satisfiable. Assume $(z_1 \vee \cdots \vee z_k)$ is satisfied by $z_i = 1$. If $i = 1$ or $i = 2$, then we set $y_j = 0$ for all $j = 1, \ldots, k-3$. So, the first clause in $\tilde{C}$ is satisfied by z_1 or z_2 and all remaining clauses in $\tilde{C}$ are satisfied by $\overline{y}_j$, $j = 1, \ldots, k-3$.

If $i \geqslant 3$, then we set $y_1 = y_2 = \cdots = y_{i-2} = 1$, $y_{i-1} = y_i = \cdots y_{k-3} = 0$. Now, by construction, in $\tilde{C}$ the first $i-2$ clauses are satisfied by by the y_i, the $(i-1)$st clause (containing z_i) is clearly satisfied by z_1, and the remaining $k-(i-2)-3$ clauses in $\tilde{C}$ are satisfied by $\overline{y}_i$.

Next, assume $\tilde{C}$ to be satisfied. We distinguish the following 3 cases. If all $y_j = 0$, then $z_{k-1} \vee z_k$ must evaluate to 1, thus also C is satisfied. Analogously, if all $y_j = 1$, then $z_1 \vee z_2$ must evaluate to 1, and hence C is satisfied, too.

It remains to consider the case that up to some i, $1 \leqslant i < k-3$ we have $y_1 = \cdots = y_i = 1$ and $y_{i+1} = 0$. Now, the $(i+1)$st clause of $\tilde{C}$ can evaluate to 1 if and only if $z_{i+2} = 1$, that is, C is again satisfied. ∎

For the sake of completeness, we finish our example here for the case of 3-CNF. It remains to transform (8.2) into a 3-CNF. So, we have to apply the rules for $k > 3$ and $k = 2$. Applying the rule for $k > 3$ requires the introduction of a new variable y_2 and applying the rule for $k = 2$ requires the introduction of a new variable y_3. Thus, we finally obtain.

$$\begin{aligned} & (x_1 \vee x_2 \vee \overline{x}_3 \vee y_1) \wedge (\overline{x}_4 \vee \overline{y}_1) \wedge (\overline{x}_3 \vee x_5 \vee \overline{y}_1) \\ \sim\ & (x_1 \vee x_2 \vee y_2) \wedge (\overline{y}_2 \vee \overline{x}_3 \vee y_1) \wedge (\overline{x}_4 \vee \overline{y}_1 \vee y_3) \wedge (\overline{x}_4 \vee \overline{y}_1 \vee \overline{y}_3) \wedge (\overline{x}_3 \vee x_5 \vee \overline{y}_1)\ . \end{aligned}$$

The importance of 3-SAT is its simple combinatorial structure which allows to apply it to prove the $\mathcal{NP}$-completeness of many other problems as shown below.

Exercise 29. *Prove or disprove* 2-SAT $\in \mathcal{P}$.

Theorem 8.12. CLIQUE *is* $\mathcal{NP}$*-complete.*

Proof. CLIQUE is in $\mathcal{NP}$, since on input (G, k) a nondeterministic Turing machine M may guess an appropriate set of k vertices and then verify that it is indeed forming a k-Clique provided $(G, k) \in$ CLIQUE. On the other hand, if $(G, k) \notin$ CLIQUE, then no witness does exist and thus M will not find any accepting computation.

So, it remains to show that CLIQUE is $\mathcal{NP}$-hard. This is done by reducing 3-SAT to CLIQUE. Let $F = F_1 \wedge \cdots \wedge F_k$ be a Boolean formula in 3-CNF, i.e., $F_i = z_{i,1} \vee z_{i,2} \vee z_{i,3}$, where $z_{i,j} \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$ for $i = 1, \ldots, k$ and $j = 1, 2, 3$. For F we define a graph $G_F = (V_F, E_F)$ having $3k$ many vertices $v_{i,j}$, $i = 1, \ldots, k$ and $j = 1, 2, 3$. Intuitively, vertex $v_{i,j}$ corresponds to literal $z_{i,j}$. A pair $(v_{i,j}, v_{i',j'})$ of vertices is connected by an edge if and only if $i \neq i'$ and $z_{i,j} \neq \overline{z}_{i',j'}$.

Claim 1. G_F *possesses a* k*-Clique if and only if* F *is satisfiable.*

First, we observe that G_F cannot possess a complete subgraph having more than k vertices, since any subgraph with more than k vertices must contain at least two vertices having the same i, i.e., $v_{i,j}$ and $v_{i,j'}$. But for all such vertices we have by definition $(v_{i,j}, v_{i,j'}) \notin E_F$.

Now, assume F to be satisfiable, i.e., there is an assignment a such that $F(a) = 1$. Then each clause F_i must contain at least one literal $z_{i,j}$ evaluating to 1 under the assignment a. Such a literal cannot be the negation of any other literal $z_{\ell,j'}$ satisfying clause F_ℓ. Consequently, for all pairs i, ℓ such that $i \neq \ell$ there must be an edge $(v_{i,j_i}, v_{\ell,j_\ell}) \in E_F$. Therefore G_F has a k-Clique.

Next, assume that G_F has a k-Clique. Then the complete subgraph forming the k-Clique must contain for every $i = 1, \ldots, k$ precisely one vertex v_{i,j_i}. Now, we can easily define an assignment a for F by assigning the value 1 to the literals z_{i,j_i}, where v_{i,j_i} belongs to the k-Clique. To the remaining literals we assign the value 0. Since no such literal can be the negation of any other such literal corresponding to a vertex in the k-Clique, no conflict can occur. Thus, a satisfies F, and Claim 1 is shown.

Therefore, we have reduced 3-SAT to CLIQUE. Moreover, it is easy to see that the reduction defined above is log-space computable. By the transitivity of $\leqslant_{\log}$ it follows that CLIQUE is $\mathcal{NP}$-hard. As shown above, CLIQUE $\in \mathcal{NP}$, we conclude that CLIQUE is $\mathcal{NP}$-complete. ∎

The following figure shows an example for the reduction given in the proof of Theorem 8.12 for

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_3 \vee \overline{x}_4) \ ,$$

that is $k = 3$. Thus the formula above is mapped to the graph G partially displayed below, where the 3-Clique corresponding to the satisfying assignment $\overline{x}_1 = 1$, $x_2 = 1$ is shown in red. Furthermore, the the 3-Clique corresponding to the satisfying assignment $x_3 = 1$ and $x_2 = 1$ is shown in blue. The green 3-Clique corresponds to the assignment $\overline{x}_1 = 1$, $x_2 = 1$ and $\overline{x}_3 = 1$.

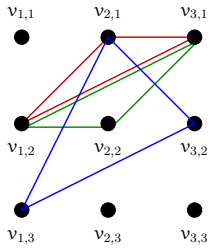

Figure 8.1: Mapping $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_3 \vee \overline{x}_4)$ to $(G, 3)$.

Having shown CLIQUE to be $\mathcal{NP}$-complete directly allows to prove VCOVER to be $\mathcal{NP}$-complete, too.

Theorem 8.13. VCOVER *is* $\mathcal{NP}$*-complete.*

Proof. It is easy to see that VCOVER $\in \mathcal{NP}$. Next, we reduce CLIQUE to VCOVER. The reduction is almost trivial. Let $G = (V, E)$ and k be given. We map G to its complement graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(u, v) \mid u, v \in V,\ u \neq v,\ (u, v) \notin E\}$. Furthermore, k is mapped to $|V| - k$. We omit the details. ∎

Exercise 30. *Show* SUBSUM *to be* $\mathcal{NP}$*-complete.*

8.5. Remarks Concerning $\mathcal{P}$ versus $\mathcal{NP}$

As already mentioned, so far we do not know whether or not $\mathcal{P} = \mathcal{NP}$. Resolving this problem remains a huge challenge.

So, let us shortly discuss consequences of the two possible answers. If $\mathcal{P} \neq \mathcal{NP}$, then not much will change, since this conjecture is favored by many scientists. The main change, of course, is then the switch from conjecture to theorem, and all the theorems having a "...if $\mathcal{P} \neq \mathcal{NP}$" in their statement would be unconditionally true.

Furthermore, there are some philosophical implications. The perhaps most important one is that testing or verifying an answer is indeed much easier than finding it. For example, verifying that a given proof is correct is easier than finding one. For the typical $\mathcal{NP}$-complete problem, finding a solution seems to involve *exhaustive search* over a set that has size exponential in the length of the input. If $\mathcal{P} \neq \mathcal{NP}$, then we know that exhaustive search cannot be avoided in general.

What are the consequences if we could prove that $\mathcal{P} = \mathcal{NP}$? Clearly this result would be also of fundamental epistemological importance. But the practical consequences may vary. If, for some important $\mathcal{NP}$-complete problem like 3-SAT someone finds a very efficient algorithm, say having running time $O(n^2)$, then the practical consequences would be heaven and hell at the same time. Heaven for those who need to find quickly solutions for $\mathcal{NP}$-complete problems, e.g., for many AI applications, for VLSI designers, for engineers.

On the other hand, all tools currently in use for privacy protection, e.g., SSL, RSA, or PGP will become useless over night. Also, much of what mathematician are doing could then be done by a machine performing efficient theorem proving.

But it is also possible that the best polynomial time algorithm for any $\mathcal{NP}$-complete problem has a running time of order $O(n^c)$, where c is a six digit number, or even a 1000000 digit number. Of course, in this case the practical consequences would be much less dramatic, since the $\mathcal{NP}$-complete problems remain hard to *solve* for larger inputs. If the latter would be true, this would also explain why we have not found any such algorithm yet.

Last but not least, if $\mathcal{P} = \mathcal{NP}$ then randomization would not provide any principal gain.

References

[1] S.A. COOK (1971), The complexity of theorem-proving procedures, *in*, Proceedings of the third annual ACM symposium on Theory of computing, Shaker Heights, Ohio, United States, pp. 151-158, ACM Press.

[2] M.R. GAREY AND D.S. JOHNSON (1979), *Computers and Intractability: A Guide to the Theory of* $\mathcal{NP}$*-completeness*, W.H. Freeman and Co., San Francisco.

Lecture 9: Probabilistic Complexity Classes

After having studied two different probabilistic algorithms in Lecture 5, and important complexity classes in Lectures 6 through 8 we are going to finish our short introduction to complexity by taking a look at probabilistic complexity classes.

9.1. Probabilistic Turing Machines

Perhaps the first papers mentioning probabilistic Turing machines are von Neumann [5] and de Leeuw *et al.* [4]. The complexity classes studied within this lecture have been introduced by Gill [3].

So, first we have to define probabilistic Turing machines. This is done informally.

Probabilistic Turing machines (abbr. PTM) are defined as deterministic Turing machines except that they have an additional tape equipped with a one-way read only head. On this auxiliary tape, an infinite sequence of zeros and one is written. These zeros and ones are realizations of a sequence of coin flips. It is assumed that zero and one each have probability $1/2$. Moreover, the coin flips are independent of one another. Furthermore, it assumed that each realization, i.e., each infinite sequence of zeros and ones written on the auxiliary tape is independent of all other such sequences.

On input a string w, the probabilistic Turing machine now works as a deterministic Turing machine. In each step of its computation it may read one symbol from the auxiliary tape. If it does, the one-way read only head of the auxiliary tape moves one position to the right. Consequently, a probabilistic Turing machine may obtain different results on the same input, but the result is determined for every random sequence written on the auxiliary tape.

So, it remains to redefine the notion of acceptance. We present here the usual definition found in the literature.

Definition 9.1. *Let* p *be a constant such that* $1/2 < p \leqslant 1$*, let* P *be a probabilistic Turing machine and let* $w \in \Sigma^*$*.* P *is said to* ***accept*** w *provided the probability of the following event* E *is greater than or equal to* p*.*

Event E*.* P *stops in an accepting configuration.*

Moreover, we write $L_p(P)$ *to denote the set of all strings* $w \in \Sigma^*$ *that are accepted by* P *with probability greater than or equal to* p*.*

Furthermore, the space and time complexity of a probabilistic Turing machine P are defined by requesting P to obey a space or time bound $f(n)$ in the event E defined above. That is, on all accepting computations P uses at most $f(n)$ cells on all its work tapes (for space complexity) and/or works at most $f(n)$ steps (for time complexity).

We are not going to provide a formal proof for the correctness of our definition, since this is beyond the scope of this course. Before discussing further issues of

probabilistic Turing machines, we are going to exemplify their power by looking again at the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$. This is done by the following theorem.

Theorem 9.1 (Freivalds [2]) *For every* p *with* $1/2 < p < 1$ *there exist a constant* $\kappa > 0$ *and a probabilistic Turing machine* P *accepting* L *with probability* p *which uses only* κ *many cells on all its work tapes.*

Proof. Let p with $1/2 < p < 1$ be given. First, we choose two numbers $c,\ d \in \mathbb{N}$, $c,\ d > 1$, depending on p such that

$$1 - 2 \cdot \left(\frac{1}{2}\right)^d \geqslant p \quad \text{and} \tag{9.1}$$

$$\left(\frac{2^c}{2^c + 1}\right)^d > 1 - p \tag{9.2}$$

are satisfied. The desired machine P (again depending on p) is defined as follows. On input $w \in \{0,1\}^*$ the machine P does the following.

(1) P checks whether or not the input w has the form $0^m 1^n$ for some $m, n \in \mathbb{N}$. If this is not the case, P rejects w. Otherwise, P continues by executing (2).

(2) P checks whether or not $(m - n) \bmod c = 0$. If this is not the case, P rejects w. Otherwise, P continues by executing (3).

(3) Now, P has to figure out whether or not $m = n$. This is done by performing the following *probabilistic experiment*. P starts on the leftmost symbol of the input and reads the whole input from left to right, thereby also reading in each step a symbol from its auxiliary tape. This is done until the last input symbol has been read. We refer to this procedure as to a *run* of the experiment. Furthermore, let us refer to the string 0^m as to the left block of the input and to 1^n as to the right block of the input.

A *run* of the experiment is *successful for the left block* of the input provided P reads only zeros on its auxiliary tape while reading zeros on its input tape. A run of the experiment is *successful for the right block* of the input provided P reads only ones on its auxiliary tape while reading the ones on its input tape.

A *set* of the experiment consists of successive runs. A set is finished provided either the run for the left block of the input was successful or the run for the right block of the input was successful. The set is then said to be won for the respective block.

The experiment consists of d sets. P accepts the input w if and only if each block could win at least one set.

Note that P, while executing (1) and (2), does not read any symbol on its auxiliary tape. Clearly, (1) can be performed without using any cells on the work tapes. It

suffices to scan the input ones, and then to return the head of the input tape to the first position of the input.

Also, for the execution of (2), P does not use any cell of its work tapes. It suffices first to count the number of zeros modulo c (by using an appropriate number of states), and then the number of ones, again modulo c, where P can memorize the outcome of counting the zeros modulo c in an appropriate state s_r. Here r stands for the remainder, i.e., for $\mathsf{m} \bmod \mathsf{c}$. From this state, P can switch to c different states (which all memorize s_r). If $\mathsf{n} \bmod \mathsf{c} = \mathsf{r}$, then P switches to the particular state in which it can start executing (3). Otherwise, the input is already rejected, and P stops.

When starting (3), there are only two cases left, i.e., $\mathsf{m} = \mathsf{n}$ or $|\mathsf{m} - \mathsf{n}| \geqslant \mathsf{c}$. It is clear that for the execution of the experiment only a constant number κ of tape cells is sufficient. Also, it is clear that the probability to arrive at a run successful for either the left of right block, say ε, satisfies $\varepsilon > 0$. Hence, $1 - \varepsilon < 1$ is the probability that a run was neither successful for the left block nor for the right block or it was successful for both blocks. Since $\lim_{\mathsf{n}\to\infty}(1-\varepsilon)^\mathsf{n} = 0$, we directly see that the probability to finish a set tends to 1 if the number of steps performed by P tends to infinity.

Consequently, it remains to calculate the probability of acceptance for the input for both cases.

Case 1. $\mathsf{n} = \mathsf{m}$.

In this case, the probability to win is for each block the same, i.e., 1/2. Therefore, the probability for the left block to win all d sets is $(1/2)^\mathsf{d}$ and so is the probability for the right block to win all d sets. Thus, the probability that one of the blocks wins all d sets is $2 \cdot (1/2)^\mathsf{d}$. By (9.1) we can conclude

$$1 - 2 \cdot \left(\frac{1}{2}\right)^\mathsf{d} \geqslant \mathsf{p} \ ,$$

and thus, with probability greater than or equal to p the input is accepted.

Case 2. $|\mathsf{m} - \mathsf{n}| \geqslant \mathsf{c}$.

Without loss of generality we can assume $\mathsf{m} < \mathsf{n}$, and therefore, $\mathsf{m} + \mathsf{c} \leqslant \mathsf{n}$.

Now, the probability for the left block to win is

$$\frac{2^{-\mathsf{m}}}{2^{-\mathsf{n}} + 2^{-\mathsf{m}} - 2^{-(\mathsf{n}+\mathsf{m})}} \ .$$

Thus, we can estimate

$$\begin{aligned} \frac{2^{-\mathsf{m}}}{2^{-\mathsf{n}} + 2^{-\mathsf{m}} - 2^{-(\mathsf{n}+\mathsf{m})}} &> \frac{2^{-\mathsf{m}}}{2^{-\mathsf{n}} + 2^{-\mathsf{m}}} \\ = \frac{2^\mathsf{c}}{2^\mathsf{c} + 2^{-\mathsf{n}+\mathsf{m}+\mathsf{c}}} &\geqslant \frac{2^\mathsf{c}}{2^\mathsf{c} + 1} \ . \end{aligned}$$

Consequently, the probability for the left block to win all d sets can be estimated by

$$\left(\frac{2^\mathsf{c}}{2^\mathsf{c} + 1}\right)^\mathsf{d} \ .$$

Finally, by (9.2) we can conclude

$$\left(\frac{2^c}{2^c+1}\right)^d > 1-p \quad \text{and, therefore}$$

$$\begin{aligned} 1-\left(\frac{2^c}{2^c+1}\right)^d &< 1-(1-p) \\ &= p\ . \end{aligned}$$

That is, the probability to accept a string of the form 0^m1^n with $m < n$, i.e., not belonging to L, is less than p. ∎

So, we have just seen what amazing computational power probabilism can provide if the computational resources are severely restricted. But some remarks are in order here. Though the algorithm presented in the proof above is very space efficient, it is not nearly as time efficient. Performing the probabilistic experiment described takes a huge amount of time.

In the following we are mainly interested in algorithms that have an efficient run-time. Thus, we shall restrict ourselves to consider probabilistic Turing machines obeying an (expected) polynomial bound for their run time.

9.2. The Probabilistic Complexity Classes $\mathcal{PP}$, $\mathcal{RP}$, $\mathcal{ZPP}$, and $\mathcal{BPP}$

We define the following complexity classes $\mathcal{PP}$, $\mathcal{RP}$, $\mathcal{ZPP}$, and $\mathcal{BPP}$ as follows. First, we recall what is meant by saying that a language is acceptable in probabilistic polynomial-time.

Definition 9.2. *The **probabilistic polynomial** ($\mathcal{PP}$) class is the set of languages L for which there is a PTM P running in polynomial-time such that for all strings x we have*

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Longrightarrow \Pr(x \text{ is rejected}) > 1/2$.

It can be said that $\mathcal{PP}$ is the weakest class of problems that can be approximately *solved* in the very intuitive sense of the word. Moreover, it seems that any problem not in $\mathcal{PP}$ will take more than polynomial-time to be solved. However, no proof is known.

Requiring the majority of answers to be correct is a natural idea. But if the correct solution is given with a probability close to 1/2 it is hard to differentiate the incorrect solution from the correct one. Moreover, as the input size grows, we may be faced with the problem that the probability of the correct solution tends more and more to 1/2. Thus, such a machine will be hard to distinguish from a machine that is simply guessing without performing any computation. This gives way to a more restrictive

definition which we present next. Now, we are going to bound away from 1/2 the margin of the solution.

Definition 9.3. *The* ***bounded probabilistic polynomial*** ($\mathcal{BPP}$) *class is the set of languages* L *for which there are a PTM* P *running in polynomial-time and some constant* $\varepsilon > 0$ *such that for all strings* x *we have*

(a) $\mathsf{x} \in \mathsf{L} \Longrightarrow \Pr(\mathsf{x} \text{ is accepted}) > 1/2 + \varepsilon$,

(b) $\mathsf{x} \notin \mathsf{L} \Longrightarrow \Pr(\mathsf{x} \text{ is rejected}) > 1/2 + \varepsilon$.

Clearly, we directly get $\mathcal{BPP} \subseteq \mathcal{PP}$, since $1/2 + \varepsilon > 1/2$ for all $\varepsilon > 0$. Note that Definition 9.3 can be modified and still gives the same complexity class. In particular, for any constant $\varepsilon \in (0, 1/2)$ we again arrive at $\mathcal{BPP}$. The class $\mathcal{BPP}$ contains all languages that can be accepted by efficient Monte-Carlo algorithms. Thus, one can use the same idea as in the proof of Corollary 5.8 to verify the invariance of $\mathcal{BPP}$ on the particular choice of $\varepsilon \in (0, 1/2)$.

Moreover, we can further sharpen the definition of $\mathcal{BPP}$ by requiring that the PTM is making only one type of error. More precisely, we require that for all inputs $\mathsf{x} \notin \mathsf{L}$ the PTM for accepting L makes *no* error. Furthermore, every string from L must be accepted with probability at least 1/2. Again, we could replace 1/2 by any constant greater than 1/2 and less than 1. The resulting complexity class is denoted by $\mathcal{RP}$. Intuitively, $\mathcal{RP}$ stands for *random polynomial time* though this term may be a bit misleading.

More formally, we arrive at the following definition.

Definition 9.4. *The* ***one-sided error probabilistic polynomial*** ($\mathcal{RP}$) *class is the set of languages* L *for which there are a PTM* P *running in polynomial-time such that for all strings* x *we have*

(a) $\mathsf{x} \in \mathsf{L} \Longrightarrow \Pr(\mathsf{x} \text{ is accepted}) > 1/2$,

(b) $\mathsf{x} \notin \mathsf{L} \Longrightarrow \mathsf{x}$ is rejected.

As a matter of fact, we already know an important problem belonging to $\mathcal{RP}$. Recalling our results obtained in Lecture 5, we have seen that $\overline{PRIM} \in \mathcal{RP}$, where $PRIM$ denotes the set of all binary representations of prime numbers.

The last probabilistic complexity class we are going to define represents the efficient Las Vegas algorithms. Here, *no error* whatsoever is allowed and the expected run time must be uniformly bounded by a polynomial in the length of all inputs over the underlying alphabet Σ. This class is denoted by $\mathcal{ZPP}$. So, $\mathcal{ZPP}$ stands for "zero-error probabilistic polynomial time."

The following theorem establishes the more obvious relations with respect to set inclusion between the probabilistic complexity complexity classes and some other previously defined complexity classes.

For this purpose, let us recall the notion of a balanced Turing machine.

Definition 9.5. *A nondeterministic Turing machine is* ***balanced*** *if all computation paths over a string* x *are of the same length and, moreover, each state is a guess state.*

Now, it is easy to see that every $\mathcal{NP}$ machine M can be replaced by a balanced NDTM M' such that $L(M) = L(M')$.

Furthermore, the notion of a balanced nondeterministic Turing machine can be easily adapted to a probabilistic Turing machine. The only modification to be made is to replace "guess state" by "coin-tossing" state.

Exercise 31. *Prove that for every probabilistic Turing machine* P *accepting a language* L *in the sense of* $\mathcal{PP}$ *there is balanced probabilistic Turing machine* P' *such that* $L(P) = L(P') = L$.

Theorem 9.2.

(1) $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{PP} \subseteq \mathcal{PSPACE}$

(2) $\mathcal{RP} \subseteq \mathcal{NP} \subseteq \mathcal{PP}$

(3) $\mathcal{ZPP}$, $\mathcal{RP}$ *and* $\mathcal{BPP}$ *are closed under union and intersection.*

Proof. First, we prove Assertion (1). Clearly, by definition we have $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP}$ and $\mathcal{BPP} \subseteq \mathcal{PP}$ (as already mentioned above).

For seeing that $\mathcal{PP} \subseteq \mathcal{PSPACE}$ it suffices to notice that a PTM P can be simulated by a deterministic Turing M machine which performs all possible computations of P. Additionally, M counts the number of accepting computations and makes at the end a majority vote. We omit the details.

Furthermore, $\mathcal{RP} \subseteq \mathcal{BPP}$ is obtained as follows. Let L be accepted in the sense of $\mathcal{RP}$ by a PTM P. Thus, if $x \notin L$ then x is always rejected. If $x \in L$, then $\Pr(x \text{ is accepted}) > 1/2$. Thus, we can construct a PTM P' which behaves as follows. On every input x it runs the PTM P exactly twice. P' accepts x, if x has been accepted by P at least once. Hence, if $x \notin L$, then P' will never accept x, too. On the other hand, if $x \in L$, then P' makes an error if and only if P has made an error on x twice. The probability that P is not accepting x twice is, however, less than $1/4$. Hence P' accepts L in the sense of $\mathcal{BPP}$. This proves Assertion (1).

For showing Assertion (2), first assume a language L that is accepted in the sense of $\mathcal{RP}$ by a PTM P. Again, if $x \notin L$, then x is always rejected. Thus, we can simply remove the coin-flips made by P and replace them by a nondeterministic choice. Hence, the resulting machine M nondeterministically accepts L. This proves $\mathcal{RP} \subseteq \mathcal{NP}$.

Next, we have to show that $\mathcal{NP} \subseteq \mathcal{PP}$. Let $L \in \mathcal{NP}$ be witnessed by the nondeterministic Turing machine M. First recall that for all $x \notin L$ there is no accepting computation of M for x. But if $x \in L$, then at least one accepting computation of M

for x must exist. However, the desired PTM P has to accept/reject a string x if the majority of the computations performed is accepting/rejecting x. For reaching this goal, we proceed as follows.

Let M be a balanced nondeterministic Turing machine and let $L(M)$ be the language accepted by M. Moreover, there exists a polynomial p such that M takes time $p(|x|)$ on all inputs x. Without loss generality we can also assume that $\Sigma = \{0,1\}$. We construct a balanced probabilistic Turing machine P from the machine M as follows.

First, each guessing state is *replaced* by a coin-tossing state. The outcome of a coin-toss then corresponds to a guess. Each terminal node of the computation tree of M over a string x with $n = |x|$ is then reached with probability $1/2^{p(n)}$. Now, a further computation is carried out at each such leaf. That is, the terminal leafs of M's computation tree on x are no longer terminal leafs for the computation tree of P over x.

Let q be a polynomial such that $p(n) \leqslant q(n)$ for all $n \in \mathbb{N}$. If the terminal leaf of M's computation tree on x is rejecting, then P will toss its coin $q(n)$ times. If not all outcomes of these coin tosses are head, then P tosses the coin again. It accepts if the last coin toss is head and rejects x otherwise. If all outcomes of these $q(n)$ coin tosses are head, then P tosses the coin again (for being balanced) but rejects the input x regardless of the outcome of the last coin toss.

On the other hand, if the reached terminal leaf in M's computation tree on x is accepting, then machine P tosses its coin again $q(n)+1$ times and accepts regardless of the outcome (again this done only to make P balanced).

It remains to analyze the probability of acceptance/rejection of x by P. Since we have designed P in a way such that it balanced, it suffices to count accepting leaves.

Case 1. $x \notin L(M)$.

Then, the definition of P directly implies that

$$\begin{aligned} \Pr(x \text{ is rejected} | x \notin L) &= \frac{2^{p(n)}\left(2^{q(n)}+1\right)}{2^{p(n)+q(n)+1}} \\ &= \frac{1}{2} + \frac{1}{2^{q(n)+1}} \end{aligned}$$

Case 2. $x \in L(M)$.

$$\begin{aligned} \Pr(x \text{ is accepted} | x \in L) &\geqslant \frac{\left(2^{p(n)}-1\right)\left(2^{q(n)}-1\right)}{2^{p(n)+q(n)+1}} + \frac{2^{q(n)+1}}{2^{p(n)+q(n)+1}} \\ &= \frac{1}{2} + \frac{2^{q(n)} - 2^{p(n)} + 1}{2^{p(n)+q(n)+1}} \\ &> \frac{1}{2} \quad \text{since } q(n) \geqslant p(n) \ . \end{aligned}$$

Hence P accepts $L(M)$ in the sense of $\mathcal{PP}$. This completes the proof of Assertion (2).

Finally, we have to show (3). Consider two probabilistic Turing machines P and P' such that $\mathsf{L}(\mathsf{P})$ and $\mathsf{L}(\mathsf{P}')$ are accepted in the sense of $\mathcal{ZPP}$. Then it easy to see that a probabilistic Turing machine $\hat{\mathsf{P}}$ accepts $\mathsf{L}(\mathsf{P}) \cap \mathsf{L}(\mathsf{P}')$ ($\mathsf{L}(\mathsf{P}) \cup \mathsf{L}(\mathsf{P}')$) in the sense of $\mathcal{ZPP}$ if $\hat{\mathsf{P}}$ works as follows. It simulates both P and P' and accepts its input x if P and P' accept x (if P or P' accept x).

We leave it as an exercise to show the closure with respect to union and disjunction for the remaining classes. ∎

Exercise 32. *Prove that $\mathcal{ZPP}$ is also closed under complement.*

Next, we show the following characterization for $\mathcal{ZPP}$. In order to this, we introduce the notation co-$\mathcal{C}$ for any of the complexity classes defined in this lecture.

Theorem 9.3. $\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$

Proof. We partition the proof into the usual two parts.

Claim 1. $\mathcal{ZPP} \subseteq \mathcal{RP} \cap \text{co-}\mathcal{RP}$.

Let $\mathsf{L} \in \mathcal{ZPP}$ be arbitrarily fixed. First, recall Markov's inequality, i.e., let T be a random variable such that $\mathsf{E}[\mathsf{T}]$ exists. Then we have: for all real numbers $\alpha \geqslant 1$

$$\Pr(\mathsf{T} \geqslant \alpha \cdot \mathsf{E}[\mathsf{T}]) \leqslant \frac{1}{\alpha} \,. \tag{9.3}$$

Thus, we can bound the run time of a probabilistic Turing machine P on input x by

$$\Pr(\mathit{time}_{\mathsf{P}}(\mathsf{x}) > 2 \cdot \mathsf{E}[\mathit{time}_{\mathsf{P}}(\mathsf{x})]) < \frac{1}{2} \,, \tag{9.4}$$

i.e., by choosing $\alpha = 2$ in (9.3). Now, let P be an expected T time bounded probabilistic Turing machine accepting L. We define a probabilistic Turing machine $\hat{\mathsf{P}}$ as follows.

On input x the PTM $\hat{\mathsf{P}}$ simulates the PTM P at most $2 \cdot \mathsf{T}(|\mathsf{x}|)$ many steps. If P did not accept x while performing this simulation, then $\hat{\mathsf{P}}$ rejects x and stops. On the other hand, if P has accepted x within the time bound of $2 \cdot \mathsf{T}(|\mathsf{x}|)$, then $\hat{\mathsf{P}}$ also accepts x, and stops.

By construction, if $\mathsf{x} \notin \mathsf{L}$ then $\hat{\mathsf{P}}$ will never accept x. On the other hand, if $\mathsf{x} \in \mathsf{L}$ then P always accepts x. However, $\hat{\mathsf{P}}$ may be forced to reject x in case the simulated computation exceeds the time bound of $2 \cdot \mathsf{T}(|\mathsf{x}|)$ many steps. The probability for this event is less than $1/2$ due to the choice of α (cf. (9.4) above). Consequently, $\hat{\mathsf{P}}$ accepts x with probability greater than $1 - 1/2$ that is with probability greater than $1/2$. Thus, by Definition 9.4 we can conclude that $\hat{\mathsf{P}}$ is an $\mathcal{RP}$ Turing machine accepting L, i.e., $\mathsf{L} \in \mathcal{RP}$.

Next, recall that $\mathcal{ZPP}$ is closed under complement. Hence, $\overline{\mathsf{L}} \in \mathcal{ZPP}$, too. So, we can repeat the same proof as above for showing that $\overline{\mathsf{L}} \in \mathcal{RP}$. But this means nothing else than $\mathsf{L} \in \text{co-}\mathcal{RP}$. This proves Claim 1.

Claim 2. $\mathcal{RP} \cap \text{co-}\mathcal{RP} \subseteq \mathcal{ZPP}$.

Assume any language $L \subseteq \Sigma^*$ such that $L \in \mathcal{RP} \cap \text{co-}\mathcal{RP}$. By Definition 9.4 we can conclude that there are PTMs P_L and $P_{\overline{L}}$ witnessing $L \in \mathcal{RP}$ and $\overline{L} \in \mathcal{RP}$. Moreover, these PTMs both obey a polynomial time bound for their runtime, i.e., there are polynomials q_L and $q_{\overline{L}}$ such that $time_{P_L}(x) \leqslant q_L(|x|)$ and $time_{P_{\overline{L}}}(x) \leqslant q_{\overline{L}}(|x|)$ for all $x \in \Sigma^*$. We set $q(n) = \max\{q_L(n), q_{\overline{L}}(n)\}$ for all $n \in \mathbb{N}$.

It remains to construct a PTM P such that P accepts L in the sense of $\mathcal{ZPP}$. This is done as follows.

On input x, the machine P simulates both P_L and $P_{\overline{L}}$ on input x until both machines have stopped.

If P_L has accepted x then also P accepts x. If $P_{\overline{L}}$ has accepted x then P rejects x.

If both P_L and $P_{\overline{L}}$ did not accept x, we repeat the simulation until one the machines P_L or $P_{\overline{L}}$ does accept.

It remains to show that P accepts L in the sense of $\mathcal{ZPP}$ and that the expected running time of P is bounded by a polynomial in the length of the input.

First, we prove the correctness. If P_L has accepted x, then by the definition of $\mathcal{RP}$ we know for sure that $x \in L$. Thus, in this case, P correctly accepts x.

If $P_{\overline{L}}$ accepts x then we can conclude that $x \in \overline{L}$. Consequently, P correctly rejects x.

Next, we estimate the expected running time of P. Note that in case neither P_L nor $P_{\overline{L}}$ did accept x we know nothing. Both cases $x \in L$ and $x \notin L$ are possible. However, one of the machines P_L and $P_{\overline{L}}$ must have made an error. The probability for making an error is for both machines less than $1/2$. Therefore, we can estimate the expected running time of P on input x as follows.

$$\begin{aligned} E[time_P(x)]) &\leqslant \sum_{k=0}^{\infty} \frac{1}{2^k} \left(time_{P_L}(x) + time_{P_{\overline{L}}}(x)\right) \\ &\leqslant 2 \cdot (time_{P_L}(x) + time_{P_{\overline{L}}}(x)) \leqslant 2 \cdot q(|x|) \ . \end{aligned}$$

Consequently, P witnesses $L \in \mathcal{ZPP}$. ∎

We like to conclude our short excursion into the field of randomized computations by mentioning that there is much more. In particular, it is not too hard to prove that there are complete problems for $\mathcal{PP}$. The perhaps easiest $\mathcal{PP}$-complete problem is *MAJ* defined as the set of all Boolean formulae that are satisfied by the majority of possible assignments of the variables occurring in them. But we do *not* know any problem that is complete for $\mathcal{BPP}$ under polynomial time reductions. One reason for the difficulty to find a problem that is complete for $\mathcal{BPP}$ is that the defining property of the class $\mathcal{BPP}$ is *semantic*. That is, for *every* string x over the underlying alphabet, a Turing machine has either to accept x with probability at least $1/2 + \varepsilon$ or it has to

reject it with probability at least $1/2 + \varepsilon$. Given a description of a Turing machine, it is undecidable whether or not it has this property.

Furthermore, it has been a long standing open problem whether or not $\mathcal{PP}$ is also closed under union and intersection. This problem got solved in 1991, but the proof technique used is too complex to be included here. We refer the reader to Beigel *et al.* [1] for further information.

References

[1] RICHARD BEIGEL, NICK REINGOLD AND DANIEL SPIELMAN (1991), PP is closed under intersection. *In* Proceedings of the twenty-third annual ACM symposium on Theory of computing, New Orleans, Louisiana, United States, pp. 1–9.

[2] RŪSIŅŠ FREIVALDS (1981), Probabilistic Two-Way Machines, *In* Mathematical Foundations of Computer Science 1981, Strbske Pleso, Czechoslovakia, August 31 - September 4, 1981, Proceedings. Lecture Notes in Computer Science 118, Springer, pp. 33–45.

[3] JOHN GILL (1977), Computational Complexity of Probabilistic Turing Machines, *SIAM Journal on Computing* **6**, no. 4, 675–695.

[4] K. DE LEEUW, E. F. MOORE, C. E. SHANNON, AND N. SHAPIRO (1956), Computability by probabilistic machines. *In* C. E. Shannon and J. McCarthy, editors, Automata Studies, pages 183–212.

[5] JOHN VON NEUMANN (1961), *Probabilistic logics and synthesis of reliable organisms from unreliable components*, Volume 5. Pergamon Press.

Part 2: Cryptography

"Ceux qui se vantent de lire les lettres chifrées sont de plus grands charlatans que ceux qui se vanteraient d'entendre une langue qu'ils n'ont point apprise."

Voltaire (Dictionnaire philosophique, 1769)

"It may be well doubted whether human ingenuity can construct an enigma of this kind [*a cryptogram*] *which human ingenuity may not, by proper application, resolve."*

E. A. Poe, (*in* The Gold Bug, 1843)

Part of this course is devoted to finding out who of those famous thinkers is closer to the truth.

Lecture 10: Classical Two-Way Cryptosystems

10.1. Introduction

AESYAISLAGFAKEGJWAEHGJLSFLLZSFCFGODWVYW

WAFKLWAF

This lectures mainly clarifies the subject of *cryptology*. Generally speaking, cryptology is about *communication in the presence of adversaries*. Cryptology can be diveded into two major parts, i.e., *cryptography* and *cryptanalysis*. Cryptography is the science or art of secret writing while cryptanalysis is its natural counterpart, that is, the art of reading secret messages. A classic goal of cryptography is *privacy*: two or more parties wish to communicate in a way such that an adversary knows nothing about what was communicated.

The history of cryptology goes back to ancient times. However, for centuries cryptology has been mainly considered as a secret art developed, taught and learned only by those few people that had access to the *black chambers*. Historically, its applications have been mainly restricted to diplomacy and military domains. The invention of radio gave a tremendous impetus to cryptology. On the one hand, it became easy for an adversary to eavesdrop over long distances. Thus, an adversary could easily scan the message exchange of two parties on a regular basis. On the other hand, the ability to communicate over great distances in real time provided serious advantages in a variety of domains. This led to the development of more sophisticated cryptographic encryption schemes. During Word War II, cryptanalytic needs heavily enforced the development of electronic computing devices culminating in the appearance of the first computers. On the other hand, the rapid development of computer technology created a mass market for cryptographic techniques. Nowadays, cryptographic techniques are sought for in industrial applications, governmental applications, and by many citizens for protecting their privacy. Typical applications include electronic signatures, electronic banking, secret exchange of electronic mails and the like. We shall consider them throughout this course in some more detail.

The remaining part of this course is to Public Key Cryptography. We present the general framework for public key cryptosystems and describe several popular realizations of it. Finally, we deal with the rapidly developing area of cryptographical protocols. In particular, we describe methods to flip a coin by telephone, to play poker by telephone, to share a secret and to sign electronically.

10.2. The Basic Model

In the basic model, we consider the communication between two parties, the sender and receiver. The sender wants to transmit a message to the receiver using an insecure channel. We do not specify the physical nature of the channel, i.e., it could be anything, a radio, a computer net, a telephon, a human and the like. By insecure we mean that the message transmitted may be eavesdropped by an adversary.

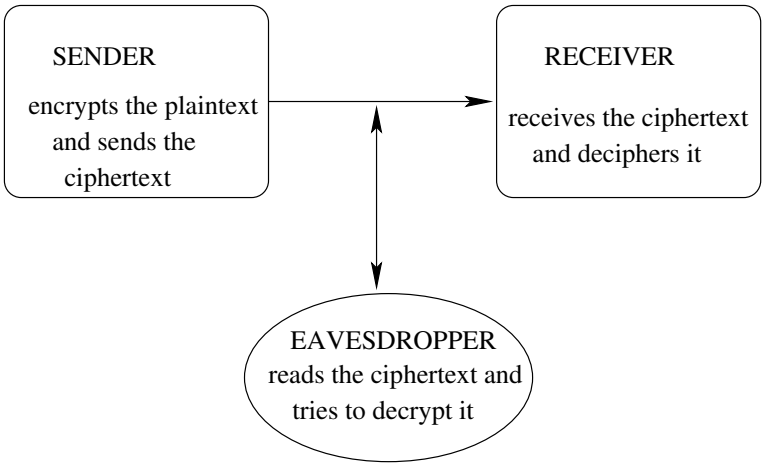

Figure 10.1: The Basic Model

The message we want send is called *plaintext*. However, only the intended recipients should be able to read and to understand the message sent. Thus, messages are sent in disguised form, and the disguised message is called the *ciphertext*. The process of converting a plaintext to a ciphertext is called *enciphering* or *encryption*, and the reverse process is referred to as *deciphering* or *decryption*. We are confronted with the following somehow contradictory requirements. Encryption and decryption should be "easy," i.e., they should be computable using a reasonable amount of space and time. On the other hand, decryption should be "hard," i.e., the adversary should either not be able to decipher the message eavesdropped in principal or it should be computationally infeasible for her to do so. The classical solution to this problem is a *secret-key* cryptosystem. Both sender and receiver agree on a key which they will keep secret (hence the name secret-key cryptosystem). Figure 10.1 displays the classical model of two-way cryptology.

Next, we exemplify this basic model using a cryptosystem invented by Julius Caesar. The underlying idea is fairly simple. We write all the letters of the latin alphabet

$\mathcal{A} = \{\mathsf{A}, \mathsf{B}, \ldots, \mathsf{Z}, \mathfrak{b}\}$ ($\mathfrak{b}$ denotes the blank symbol) in a row and rewrite $\mathcal{A}$ starting with A under the letter D as shown in Figure 10.2.

$$\begin{array}{c} \mathsf{A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z\ \mathfrak{b}} \\ \mathsf{Y\ Z\ \mathfrak{b}\ A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X} \end{array}$$

Figure 10.2: The Caesar system

Now, a plaintext is easily encrypted by replacing each letter in it by the corresponding letter displayed in the second row of Figure 10.2, i.e., A is replaced by Y, B is replaced by Z and so on. The secret key is just the table above. Hence, the decryption can be easily performed by reversing the replacement. That is, each letter in the ciphertext is replaced by the corresponding letter in the first row. As an example consider "WHY" which is encrypted as "TEV." Hence, this cryptosystem fulfills the first two requirements established above, i.e., encryption and deciphering are easy to compute provided the table given in Figure 10.2 is known.

Does it also fulfill the 3rd requirement? What can be said about the complexity of cryptanalysis in this case? For answering this question, we have two distinguish two cases.

Case 1. The cryptosystem itself is unknown.

Case 2. The principal cryptosystem is known but the actual key is is unknown.

In the following, we always assume Case 2. There are several reasons for doing this. First of all, if a cryptosystem is hard two break in Case 2, it is even harder to break in Case 1. Thus, we are on the safer side when assuming Case 2. Second, the experience gained shows that the principal structure of a cryptosystem cannot be kept secret for a long time. Thus, a potential adversary may well know what cryptosystem is used by the communicating parties. Moreover, the principal choice of cryptosystems is not large. Thus, an adversary may well try to attack several of them in parallel.

Moreover, we generally distinguish the following sources of information available to an eavesdropper.

Case 2.1. Ciphertext only.

This scenario refers to a situation in which the adversary has eavesdropped messages encrypted by using the same key. Her task is to decipher the whole messages or at least part of them.

Case 2.2. Ciphertext obtained from known plaintext.

Now, the adversary has additionally access to some message in plaintext (or part of a longer message) and knows its particular encryption. Again, her task is to decrypt the whole message or at least part of the remaining ciphertext. This variant appears most often in practical situations.

Case 2.3. Ciphertext obtained from plaintext *chosen* by the adversary.

In this scenario, the adversary has been able to force the sender to encrypt some plaintext carefully chosen by herself. Again, the task consists in decrypting the whole message. As we shall see later, this scenario is also well conceivable, and part of the design of a cryptosystem has to be devoted to avoid such attacks to a large extent.

Let us now analyze the cryptosystem described above with respect to these 3 cases. For this purpose, it is very helpful to look a little bit closer to the cryptosystem described above. It is easy to see that Caesar's cryptosystem is nothing else than a cyclical shift of the alphabet $\mathcal{A}$. Thus, knowing the cipher of *one letter* is already sufficient to break it. Consequently, in Case 2.3 the adversary has no difficulties at all. The same applies *mutatis mutandis* to Case 2.2. Moreover, there are only 27 cyclical shifts. Thus, even in Case 2.1 the adversary has no principal difficulty to decipher the message received. Just trying all possibilities is feasible and leads to successful encryption. As we shall see a little bit later there are also additional techniques to reduce the number of possibilities considerably.

Next, we ask how to improve the Caesar system. For achieving this goal it is very useful to introduce the general mathematical framework for describing two-way cryptosystems. The first step in inventing a cryptosystem is to "label" all possible plaintext message units and all possible ciphertext message units by means of mathematical objects from which functions can be "easily" constructed. These objects are often the integers in some range. For example, in the approach undertaken above we can label the letters by using the integers from 0 to 26, i.e., A is represented by 0, B is represented by 1, and so on, b is represented by 26. The same applies for ciphertext message units. Thus, we may think of the integers $\{0, 1, \ldots, 26\}$ as of the ring $\mathbb{Z}_{27}$. Now, we may use the operations addition and multiplication modulo 27. The Caesar system used above can be now expressed as $f(x) = x + 24 \bmod 27$. For example, $f(\mathsf{D}) = f(3) = 3 + 24 \bmod 27 = 27 \bmod 27 = 0 = \mathsf{A}$. Decryption is defined by $d(y) = y - 24 \bmod 27 = y + 3 \bmod 27$.

In general, any cryptosystem of the Caesar type can be expressed as $f(x) = x + r \bmod m$, where m is any suitably chosen natural number and $r \in \{0, \ldots, m-1\}$. However, the number m of possible keys is by no means sufficient for nowadays applications. This is a good point to make the following important observation.

Observation 10.1. *Cryptosystems must be designed in a way such that the number of possible keys is huge.*

As a straightforward generalization, one may try cryptosystems defined by

$$f(x) = (ax + r) \bmod m \text{ where } a, m \in \{0, \ldots m-1\} ,$$

where m is again a suitably chosen natural number. For example, we may consider *pairs* of letters (so called 2-grams) as plaintext message units, or more generally, n-grams. Then, $m = 728$ and $m = n^2 - 1$, respectively. Now, there are m^2 possible

keys (a, r). However, some care is necessary. In order to ensure decipherability, the function f must be injective. Consider $m = 27$, $(a, r) = (3, 3)$. Then,

$$f(8) = 3 \cdot 8 + 3 = 27 \equiv 0 \bmod 27$$

and

$$f(17) = 3 \cdot 17 + 3 = 54 \equiv 0 \bmod 27 \ .$$

Thus, f is not injective. The critical point is best elaborated by considering $y = ax + r \bmod m$. Hence, $a^{-1}(y - r) \equiv x \bmod m$, where a^{-1} denotes the multiplicative inverse of a. Consequently, $d(y) = f^{-1}(y) = a^{-1}y + r' \bmod m$, where $r' = -a^{-1}r \bmod m$. Therefore, we have to ensure that a^{-1} does really exist. By Theorem 3.4 we already know that the equation $ax \equiv 1 \bmod m$ is solvable if and only if $\gcd(a, m) = 1$. Moreover, if $ax \equiv 1 \bmod m$ is solvable, then the solution is uniquely determined.

Thus, in order to keep the number of possible keys as large as possible it is recommendable to choose m as a prime number. Now, we can prove the following theorem characterizing the difficulty to compute the secret key.

Theorem 10.1. *Let $p \in \mathbb{N}$ be a prime number, let $a, r \in \{0, \ldots, p-1\}$ and let $x_1, x_2 \in \{0, \ldots, p-1\}$ be two different numbers such that $f(x_1)$, $f(x_2)$ are known. Then the secret key (a, r) of the cryptosystem $f(x) = ax + r \bmod p$ can be easily computed.*

Proof. Let $n_1 =_{df} f(x_1)$ and $n_2 =_{df} f(x_2)$ as well as x_1 and x_2 be all known. Hence, the following equations must be simultaneously satisfied:

$$n_1 \equiv (ax_1 + r) \bmod p \tag{10.1}$$

$$n_2 \equiv (ax_2 + r) \bmod p \ . \tag{10.2}$$

Subtracting (10.2) from (10.1) immediately yields

$$n_1 - n_2 \equiv a(x_1 - x_2) \bmod p \ ,$$

and thus

$$a \equiv (n_1 - n_2)(x_1 - x_2)^{-1} \bmod p \ .$$

Since p is prime, the multiplicative inverse $(x_1 - x_2)^{-1}$ always exists provided $x_1 \not\equiv x_2$. Having thus computed a the parameter r can be easily obtained from (10.1) or (10.2), and the theorem is shown. ∎

Exercise 33. *Determine the complexity of the algorithm given in Theorem 10.1.*

Theorem 10.1 implicitly says that the security of a cryptosystem cannot be solely based on a huge number of possible keys. We would like to illustrate this insight. Let us again take our alphabet $\mathcal{A}$ and as the set of all possible keys we consider all permutations of $\mathcal{A}$. This would be the most general version of the Caesar system. Thus, we have 27! many keys, and since $27! \leqslant 8 \cdot 10^{27}$ just trying them all is not feasible. Even if we could test 10^9 many permutations per second, this exaustive testing would take roughly 10^{11} years.

So, at first glance, everything looks fine. Unfortunately, there is a "but," and in this case it sounds "but there is frequency analysis." The background of frequency analysis is the observation that letters appear with different frequencies in natural language. For example, in German we have the following picture.

E	18,46 %	R	7,14 %	T	5,22 %
N	11,42 %	S	7,04 %	U	5,01 %
I	8,02 %	A	5,38 %	D	4,94 %

Note that there is no absolute table for the relative frequencies of letters, since they vary in dependence on the subjects. For instance, if you compute frequencies in stock market reports and book of tales, then you get different values. Nevertheless, in German texts the letters E and N always have the highest frequency.

Now, the idea of frequency analysis is to compute the frequencies in the ciphertext and to try a mapping with respect to the table displayed above. It works very often quite well. Moreover, this method can be successfully applied to satisfy the assumptions in Theorem 10.1.

So, why don't we give it a try.

Exercise 34. *Here is some nice text from the club of cryptomanic students waiting to be deciphered: (you have to find a key* $(\mathfrak{a}, \mathfrak{r})$ mod 27 *)*

TJLKSCLVSOLTUBTLSPbTOVSMTL
MUTLKIOOQTLJUFNLJITZTBLGTCSMTL
MXCFNLHSXTBLXBMLDTCMSXTBL
TUBTBLSPbTOJQIOOTBLZSXTBL
MSLKXCMTLJUTLOTUMTCLVUQQTBL
DIBLTUBTVLVTJJTCLMXCFNGTJFNBUQQTBL
JUTLHIBBQTLBXCLBIFNLEUJFNTBL
ZTULVCLHIVVQLUVVTCLKSJLMSEKUJFNTB

Hint: Suppose that the blank symbol has the highest frequency in the plaintext.

This example also shows that one has to exclude the blank symbol from the plaintext before enciphering it. Here is another example showing the same just for fun.

MIBQLMCTSVLYIXCLOUbTLOUDTLYIXCLMCTSV

GCSbbUQU

Exercise 35. *Decipher the text above!*

10.3. Polyalphabetic Cryptosystems

MIWEMOZAYGUSJGHMVQAEBPYK

RDLOKINUUYIPN

Until now, we have considered cryptosystems that enciphered all plaintext message units using one and the same rule. Such cryptosystems are referred to as *monoalphabetic* systems. In contrast, in the following we study cryptosystems working as

follows. The first plaintext message unit is enciphered using Rule 1, the second plaintext message unit is enciphered using Rule 2, ..., the kth plaintext message unit is enciphered applying Rule k. In case the plaintext contains more than k plaintext message units, one applies the rules modulo k, i.e., the $k+1$st plaintext message unit is then enciphered by Rule 1, the $k+2$nd plaintext message unit is encrypted using Rule 2, and so on. We exemplify this idea by having a closer look at one of the oldest and well studied polyalphabetic systems - the VIGENÈRE system. This system is named after its inventor Vigenère (1523 - 1596). The basic idea consists in applying the Caesar systems $d \leqslant 26$ times. Since we have already gained some knowledge concerning the power of frequency analysis, from now on, we shall exclude the blank symbol from the alphabet used.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 10.3: The VIGENÈRE Tableau

The VIGENÈRE system is based on the tableau displayed in Figure 10.3. Additionally, the communicating parties have to agree upon a *key word* w of length d containing every letter at most ones. For example, we may use the key word *MAGIC*.

The plaintext message units are the letters A, B, ..., Z. The enciphering is performed as follows. Suppose, we want to encrypt the word *CRYPTOLOGY*.

First, search the row starting with the first letter of the plaintext, i.e., in our example with C. Then, look for the column starting with the first letter of the key word, i.e., in our example with M. At position (C, M) of the VIGENÈRE Tableau we find O which is the first letter of the ciphertext.

The ith letter of the plaintext is enciphered by searching the row starting with the ith letter of the plaintext and the column starting with the ith letter of the key word. The letter found at the intersection of this row and column in the VIGENÈRE Tableau constitutes the ith letter of the ciphertext. In case the plaintext is longer than the keyword, the keyword is written several time one behind the other.

Thus, enciphering our plaintext *CRYPTOLOGY* with the key word *MAGIC* yields *OREXVALUOA*.

Note that *CRYPTOLOGY* contains the letter O two times. However, the first O is encrypted by A, while the second one is enciphered by U. The reason for this nice behavior of the VIGENÈRE cryptosystem is that we used different rules to encipher the letters in the plaintext and, in particular, to encrypt the two occurrences of O.

Finally, decryption is performed by reversing the algorithm described above. That is, the ith letter of the ciphertext is deciphered by determining the ith letter of the key word (or in its appropriate repetition). Then, one scans the column starting with this letter until one has found the row containing the ith letter in the ciphertext. The first letter in this row is the wanted ith letter in the plaintext.

Exercise 36. *Decipher the message at the beginning of this subsection that has been eavesdropped from the club of* kryptomanic *students.*

Before dealing with the problem of breaking ciphertext encrypted by using the VIGENÈRE Tableau, we remark that there are several similar tableaus that can be easily reproduced, too. One the better known ones is the so-called BEAUFORT Tableau obtained from the VIGENÈRE TABLEAU by replacing the ith row of it by the ith row of it in reverse order (cf. Figure 10.4). Alternatively, one can design similar tableaus by using any other reasonable alphabet, e.g., the cyrillic one, the hiragana or katakana, the arabic letters a.s.o.

Next, we deal with possible cryptological attacks. Let $w = s_0 \dots s_{d-1}$ be the unknown key word of length d. Clearly, each key word uniquely defines a Vigenère Substitution. Moreover, if the encrypted plaintext has been long enough then the substitution becomes periodically with period length d. We distinguish the following two cases of cryptanalysis.

Case 1. d is known.

In this case, the following lemma is the basis of any cryptological attack.

Lemma 10.2. *Let d be the length of the key word used. Then, for every key word of length d, the corresponding Vigenère Substitution can be decomposed into d monoalphabetic substitutions.*

Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B
B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D
D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E
E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F
F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G
G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H
H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I
I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J
J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K
K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L
L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M
M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N
N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O
O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P
P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q
Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R
R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S
S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T
T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U
U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V
V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W
W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X
X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y
Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z

Figure 10.4: The BEAUFORT Tableau

Proof. Let $w = s_0 \dots s_{d-1}$ be any key word of length d, and let $k_0 k_1 \dots k_m$ be the plaintext to be enciphered. Now, when writing the plaintext in blocks of length d below the key word we obtain:

$$\begin{array}{cccc} s_0 & s_1 & \dots & s_{d-1} \\ k_0 & k_1 & \dots & k_{d-1} \\ k_d & k_{d+1} & \dots & k_{2d-1} \\ k_{2d} & k_{2d+1} & \dots & k_{3d-1} \\ \vdots & & & \\ k_{\ell d} & k_{\ell d+1} & \dots & k_m \end{array}$$

Hence, all plaintext message units in column $i \in \{0, \dots, d-1\}$ are enciphered by the same monoalphabetic substitution defined by letter s_i of the key word. More precisely, the first letter of the alphabet $\mathcal{A}$ is mapped to s_i; thus canonically defining a shift operation for the remaining letters. ∎

Consequently, if d is known then the cryptanalysis reduces to d simple monoalphabetical attacks. However, in practical situations d is often unknown. Obviously, there

are only $|\mathcal{A}|$ many possible length, since we have required the key word to contain each letter at most ones. Thus, we may simply try all possibilities which is not a big problem having a computer on hand. However, in 1860 the German cryptanalyst F. W. Kasiski developed a method allowing to determine $\mathfrak{d}$ with high probability provided the ciphertext is sufficiently long. Because of its importance for understanding security issues of cryptosystems, we present Kasiski's algorithm here.

10.4. Kasiski's Algorithm

It is the periodicity of the repeating key which leads to the weaknesses in this method and its vulnerabilities to cryptanalysis. Kasiski's general solution of *repeated key* Vigenère ciphers starts from the fact that the same pairings of message and key symbols produce the same cipher symbols. These repetitions can be recognized by the cryptanalyst. Thus, the general algorithm can be described as follows:

Step 1. Search all words $\nu_0, \dots, \nu_\ell$ in the ciphertext that appear at least twice in the ciphertext, i.e., search all ν_i such that the ciphertext can be presented as $w_i\nu_i q_i \nu_i r_i$, where w_i, q_i, r_i are also words over the cipher alphabet.

Step 2. For each ν_i found, $i = 0, \dots \ell$, compute all divisors of $|\nu_i q_i|$.

Step 3. Order the divisors found in Step 2 by their frequency. Starting with the most frequent one try for each divisor a monoalphabetic attack until a "meaningful" plaintext has been discovered.

A	V	X	Z	H	H	C	S	B	Z	H	A	L	V	X	H	F	M	V	T	L	H	I	G	H
K	A	L	B	R	V	I	M	O	F	H	D	K	T	A	S	K	V	B	M	O	S	L	A	C
G	L	G	M	O	S	T	P	F	U	L	Q	H	T	S	L	T	C	K	L	V	N	T	W	W
H	B	W	M	S	X	S	G	A	V	H	M	L	F	R	V	I	T	Y	S	M	O	I	L	H
P	E	L	H	H	L	L	I	L	F	B	L	B	V	L	P	H	A	V	W	Y	M	T	U	R
A	B	A	B	K	V	X	H	H	B	U	G	T	B	B	T	A	V	X	H	F	M	V	T	L
H	I	G	H	P	N	P	Z	W	P	B	Z	P	G	G	V	H	W	P	G	V	B	G	L	L
R	A	L	F	X	A	V	X	T	C	L	A	Q	H	T	A	H	U	A	B	Z	H	T	R	S
B	U	P	N	P	Z	W	P	B	Z	H	G	T	B	B	T	P	G	M	V	V	T	C	S	M
V	C	L	T	O	E	S	O	L	A	C	O	L	K	B	A	V	M	V	C	Y	L	K	L	A
C	G	L	G	B	M	H	A	L	G	M	V	J	X	P	G	H	U	Z	R	H	A	B	Z	S
K	H	P	E	L	H	B	U	M	F	L	H	T	S	P	H	E	K	B	A	V	T	J	C	N
W	Z	X	V	T	L	A	C	G	L	G	H	U	H	H	W	H	A	L	B	M	O	S	K	V
C	F	J	O	G	U	C	M	I	S	A	L	O	M	L	R	I	Y	C	I	L	F	E	F	I
G	S	S	L	Z	W	M	P	G	O	L	F	R	Z	A	T	S	Z	G	L	J	X	Y	P	X
Z	H	B	U	U	R	D	W	M	O	H	A	L	V	X	H	F	M	V	T	L	H	I	G	H

Figure 10.5: A ciphertext eavesdropped.

We illustrate the application of Kasiski's algorithm using the following example due to A. Salomaa [4]. Suppose, the ciphertext displayed in Figure 10.5 has been eavesdropped.

In Step 1 we find the words $\nu_0 = \mathsf{HALVXHFMVTLHIGH}$ (appearing twice) having $|\nu_0 \mathsf{q}_0| = 375$, $\nu_1 = \mathsf{VXHFMVTLHIGH}$ (appearing thrice) having $|\nu_1 \mathsf{q}_{1,0}| = 129$ (distance between first and second appearance), $|\nu_1 \mathsf{q}_{1,1}| = 246$ (distance between second and third appearance) and $|\nu_1 \mathsf{q}_{1,2}| = 375$ (distance between first and third appearance). Additionally, VXH appears in the 6th row twice having distance 12, and so does VX in the first row. Moreover, AVX appears thrice (distances 141, 39), and VX gives additionally 180, and HAL can be found four times (having the successive distances 246, 60, and 69). Computing the divisors of those lengths we obtain:

for 375: 1, 3, 5, 25, 125, 15, 75, 375,

for 129: 1, 3, 43, 129,

for 246: 1, 2, 3, 41, 6, 82, 123, 246,

for 180: 1, 2, 3, 4, 6, 5, 10, 15, 20, 45, 12, 36, 180,

for 141: 1, 3, 47, 141,

for 60: nothing new, because 60 divides 180,

for 39: 1, 3, 13, 39,

for 69: 1, 3, 23, 69, and

for 12: nothing new.

s_0 s_1 s_2	s_0 s_1 s_2	s_0 s_1 s_2	s_0 s_1 s_2	s_0 s_1 s_2	s_0 s_1 s_2	s_0 s_1 s_2
A V X	L Q H	Y M T	A V X	A V M	W Z X	L F R
Z H H	T S L	U R A	T C L	V C Y	V T L	Z A T
C S B	T C K	B A B	A Q H	L K L	A C G	S Z G
Z H A	L V N	K V X	T A H	A C G	L G H	L J X
L V X	T W W	H H B	U A B	L G B	U H H	Y P X
H F M	H B W	U G T	Z H T	M H A	W H A	Z H B
V T L	M S X	B B T	R S B	L G M	L B M	U U R
H I G	S G A	A V X	U P N	V J X	O S K	D W M
H K A	V H M	H F M	P Z W	P G H	V C F	O H A
L B R	L F R	V T L	P B Z	U Z R	J O G	L V X
V I M	V I T	H I G	H G T	H A B	U C M	H F M
O F H	Y S M	H P N	B B T	Z S K	I S A	V T L
D K T	O I L	P Z W	P G M	H P E	L O M	H I G
A S K	H P E	P B Z	V V T	L H B	L R I	H
V B M	L H H	P G G	C S M	U M F	Y C I	
O S L	L L I	V H W	V C L	L H T	L F E	
A C G	L F B	P G V	T O E	S P H	F I G	
L G M	L B V	B G L	S O L	E K B	S S L	
O S T	L P H	L R A	A C O	A V T	Z W M	
P F U	A V W	L F X	L K B	J C N	P G O	

Figure 10.6: Rewriting the ciphertext in three columns

Thus, 3 is the most frequent divisor found, since it divides all distances. Moreover, since several words have been pretty long, it is highly improbable that this is just by

Letter	s_0	s_1	s_2	Letter	s_0	s_1	s_2
A	12	5	9	N	0	0	4
B	4	9	12	O	6	4	2
C	2	11	0	P	10	7	0
D	2	0	0	Q	0	2	0
E	1	0	4	R	1	3	5
F	1	10	2	S	5	13	0
G	0	13	10	T	6	4	13
H	15	14	11	U	9	1	1
I	1	7	3	V	14	11	2
J	2	2	0	W	2	3	6
K	1	5	4	X	0	1	12
L	27	1	13	Y	4	0	1
M	2	2	17	Z	7	5	2

Figure 10.7: Counting the number of occurences of each letter in s_0, s_1 and s_2

E	12,31 %	O	7,94 %	S	6,59 %
T	9,59 %	N	7,19 %	R	6,03 %
A	8,05 %	I	7,18 %	H	5,14 %

Figure 10.8: Statistical information for English text

chance. Consequently, we conjecture the key word length to be 3. In order to perform the monoalphabetical attacks, we rewrite the ciphertext in three columns as described in Lemma 10.2, and obtain Figure 10.6 above.

Now, for each of the columns s_0, s_1, s_2, we count the number of occurrences of each letter, and get the result shown in Figure 10.7.

Furthermore, conjecture the plaintext to be written in English. Therefore, we use the statistical information available for general English text provided by the table shown in Figure 10.8.

However, the ciphertext received has been pretty short. Thus, instead of conjecturing E to be the letter having the highest frequency, we refine our approach as follows. Looking at Figure 10.8, we recognize that the triple RST is the only of consecutive letters that all have high frequency. Therefore, we search for triples of consecutive letters having simultaneously high frequency. In the first column of Figure 10.7 we find TUV and YZA. Assuming $\mathsf{R} \to \mathsf{T}$, $\mathsf{S} \to \mathsf{U}$, and $\mathsf{T} \to \mathsf{V}$ results in conjecturing a monoalphabetic right shift by two positions. Thus, Y, Z, and A would be the image of W, X, and Y, respectively. Consequently, the letters W, X, and Y must appear 4, 7, and 12 times, respectively, in the plaintext. This seems highly unlikely. Therefore, we favor $\mathsf{R} \to \mathsf{Y}$, $\mathsf{S} \to \mathsf{Z}$, and $\mathsf{T} \to \mathsf{A}$ resulting in:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
H I J K L M N O P Q R S T U V W X Y Z A B C D E F G

Analogously, for the second column of Figure 10.7 we find ABC and FGH (possibly ZAB and GHI, too; but they are less probable). Using similar arguments as above, ABC is less probable than FGH. Thus, we continue working with

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
O P Q R S T U V W X Y Z A B C D E F G H I J K L M N

Finally, in the third column KLM and FGH are possible candidates. First, we favor KLM; thus obtaining:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
T U V W X Y Z A B C D E F G H I J K L M N O P Q R S

Consequently, our three conjectures $\mathsf{t}_0(\mathsf{x}) = \mathsf{x} + 7 \bmod 26$, $\mathsf{t}_1(\mathsf{x}) = \mathsf{x} + 14 \bmod 26$, and $\mathsf{t}_2(\mathsf{x}) = \mathsf{x} + 19 \bmod 26$ provide the key word HOT. Trying to decipher the ciphertext using this key word successively delivers

THESTOVEISTHEHEARTOFSAUNA

WHENYOUTHROW...;

that is a meaningful text about sauna. Deciphering it completely and inserting the appropriate blanks and interpunction symbols, the whole text reads as follows:

"The stove is the heart of sauna. When you throw water on the stones, the air becomes more humid and feels hotter. You are thus able to experience both dry and humid heat in sauna. The art of sauna building is not discussed here. The most common mistake in building a sauna is to have too small a stove with too few stones. If the stove is only a miserable tiny metal box with a couple of stones on the top, then the room cannot be heated properly unless it is very small. Never be stingy with the heart of sauna."

Moreover, we see that VXH constitutes the cipher of HEA. However, the different appearances of VXH in the ciphertext stem from HEART, HEATING and THEART. Thus, we have been a bit in luck doing our deciphering. Usually, a longer text is needed.

Finally, some more historical remarks are in order here. Actually, Kahn [2] credits Giovan Batista Belaso who published a booklet in 1553 for having "proposed the use of a literal, easily remembered, and easily changed key ... for a polyalphabetic cipher," for what we know today as the Vigenère cipher. According to Kahn [2], Vigenère himself developed a far more sophisticated system, an "autokey" that uses the plaintext as its own key. For example, if we want to encrypt

```
THIS IS A SECRET MESSAGE
```

we choose a secret seed key character, say "D," and we write:

```
autokey:  DTHISISASECRETMESSAG
message:  THISISASECRETMESSAGE
```

You can see how the autokey consists of the seed character followed by the plaintext message itself shifted once to the right. Now, the key and message characters are looked up in a tableau to produce the enciphered message, not entirely unlike the usual, so-called "Vigenère cipher." However, there is s one important difference. While the tableau still consists of standard (shifted) alphabets, Vigenère proposed scrambling the row and column indexing alphabets at the top and side. This scrambling plus the seed character would form what we would consider the "secret key" nowadays.

If we still use the unscrambled tableau, we get the following ciphered message

```
ciphered: WAPAAASSWGTVXFQWKSGK
```

Decryption is rather interesting. Using the seed character "D," the intended recipient can decipher the first ciphertext character to get the first plaintext character — this plaintext character is now the key character to use to decrypt the second ciphertext character, and so on.

Exercise 37. *Use Kasiski's algorithm to decipher the following message,*

```
KSMEH ZBBLK SMEMP OGAJX SEJCS FLZSY
```

where the blanks have only been introduced for better readability.

Finally, the following references are recommended for further information throughout this course of introductory cryptology.

References

[1] G. BRASSARD (1988), *Modern Cryptology*, Lecture Notes in Computer Science 325, Springer-Verlag.

[2] D. KAHN (1967), *The Codebreakers: The Story of Secret Writing*, Macmillan.

[3] N. KOBLITZ (1987), *A Course in Number Theory and Cryptology*, Springer-Verlag.

[4] A. SALOMAA (1990), *Public-Key Cryptology*, EATCS Monographs on TCS, Springer-Verlag.

Lecture 11: Public Key Cryptography

Starting with this lecture, we shall deal with more contemporary ideas in cryptology. Until now, we mainly considered two way cryptosystems. In such cryptosystems the security of the communication has been mainly established by a private key. As we have seen, breaking the key enabled the cryptanalyst to decipher the message eavesdropped. However, this classical key management requires the exchange of the secret key which may be well imaginable if the number of participants is small. But what if, for instance, people like to ensure their privacy when communicating over the internet. Imagine a bank with tens of thousands customers all over the word who like to access their accounts via their computers at home. It seems absolutely hopeless to exchange frequently secret keys with all customers. But most customers have a much larger range of applications than simply accessing their accounts. Just to mention some few more things, e.g., shopping over the net using a credit card, frequently exchange of email with varying addressees, or using different computers via `telnet`. In the latter case, a user is required to identify herself by providing a password. The password is, however, transmitted over the net as plaintext, and only after having arrived at the host computer it is enciphered and checked. Thus, a potential adversary may eavesdrop it before arrival. So, please use `ssh` instead of `telnet`.

But still, there is more. Another important problem is *authentication*. The main problem addressed here is to ensure that a message received indeed originates from the source it pretends to be have sent off. In classical communication via letters, this problem has been solved by using hand written signatures (in the western hemisphere) or a "hanko," i.e., a personal seal (e.g. in China, Japan). Thus, we need something equivalent, i.e., an electronic signature.

All those real and potential applications stimulated a huge amount of research during the last three decades. In 1976, Diffie and Hellman [2] proposed a new approach, i.e., *public key cryptography*. Starting with the observation that a key in classical two-way cryptosystems has actually two separate tasks, i.e., enciphering and deciphering, Diffie and Hellman proposed to replace this double task by two keys. One key is used for encryption, and one key for decryption. The revolutionary idea, however, was to make the key for encryption *publicly available*. Clearly, this is really unimaginable in classical cryptology. The key for decryption is kept secretly by the receiver.

11.1. The General Scheme of Public Key Cryptography

The general scenario can be described as follows. Assume ℓ communicating parties $M_1, \ldots, M_\ell$. Each party M_i chooses and publishes its *public key* k_i, and keeps its *secret key* $\tilde{k}_i$ private. Suppose, party M_i wishes to send a message to party M_j. Then M_i looks for M_j's public key in the list of all public keys. Next, M_i enciphers its message using k_j and sends it out. The receiver M_j exploits his private key $\tilde{k}_j$ and deciphers the message received from M_i.

This sounds really challenging. The only problem is how to realize this nice idea. Obviously, there must be some connection between the public and the private key, since otherwise it is very hard to imagine how the deciphering can be performed. Thus, we have to require that, given the public key, it must be extremely hard to compute the private one. Additionally, computing the cipher must be easy, while deciphering has to remain extremely hard, too, without knowing the private key. These requirements directly lead to the idea of *one-way functions*.

Definition 11.1. *Let* X, Y *be non-empty sets. A* ***one-way function*** f *is an injective function* $f\colon X \to Y$ *such that* $f(x)$ *can be computed in time polynomial in the length* $|x|$ *of* x *for all* $x \in X$ *but there is no algorithm computing* $f^{-1}(y)$ *efficiently for any interesting fraction of arguments* $y \in \operatorname{range}(f)$.

Unfortunately, no one has yet proved the existence of one-way functions. Complexity theory is still not ready to handle this extremely difficult problem. Moreover, classical complexity theory mainly deals with *worst-case* complexity what is by no means ideal from the viewpoint of cryptology. The reasons for this are as follows.

(1) A problem having high worst-case complexity may be anyway easily solvable for most of its instances. Furthermore, no *non-linear lower bound* for a particular problem has been proved yet.

(2) For all practical purposes it is sufficient to possess an efficient *probabilistic* algorithm computing f^{-1}. That means, even if we would know that no deterministic algorithm computes f^{-1} for almost all inputs in polynomial time, we cannot conclude the relevant cryptosystem to be secure.

(3) Even worse, having a proof that no probabilistic algorithm computes f^{-1} for almost all inputs in polynomial time is not sufficient to derive reasonable conclusions concerning the security of the relevant cryptosystem. It still may be possible to invert f for almost all inputs of *practical* length. For example, if we would have a proved tight lower bound of $n^{\log\log n}$, then for all inputs y of length $|y| \leqslant 2^{2^{10}}$ the inversion of f can be performed in time less than or equal to $|y|^{10}$.

(4) Since all practically appearing inputs are below some length, even *non-uniform* families of different algorithms inverting f may be interesting for a cryptanalyst.

There are, however some functions f which are widely considered to be good candidates for one-way functions, e.g. modular exponentiation (the inverse is computing the discrete logarithm), computing the product of prime numbers (the inverse is factoring a given number into its prime factors), and computing $M = \sum_{i=0}^{m} x_i a_i$, where $(a_0, \ldots, a_m) \in \mathbb{N}^{m+1}$ and $(x_0, \ldots, x_m) \in \{0,1\}^{m+1}$ (the inverse is the general knapsack problem).

Next we describe how to apply one-way functions to the solution of public key cryptography. Clearly, we cannot directly apply one-way functions f for enciphering

messages. Additionally, we have to incorporate an idea how the receiver can circumvent the difficulty of inverting f. As outlined above, solely the receiver possesses the additional information provided by her secret key. This additional information should enable her to decrypt the ciphertext. The following definition formalizes this idea. The sets X and Y stand for the plaintexts and ciphertexts, respectively. Furthermore, we use K_1 and K_2 to denote the set of public keys and private keys, respectively.

Definition 11.2 (Trap-Door Function).
A ***trap-door function*** $f\colon X \times K_1 \to Y$ *is a function satisfying the following requirements.*

(i) $h_{k_1} = f(\cdot, k_1)$ *is a one-way function for every* $k_1 \in K_1$,

(ii) *there exists a polynomial* p *such that the time to compute* $h_{k_1}(x)$ *is uniformly bounded by* $p(|x|)$ *for all* $k_1 \in K_1$,

(iii) *there exist a one-way function* $d\colon K_2 \to K_1$ *and a polynomial time computable function* $g\colon K_2 \times Y \to X$ *such that* $y = f(x, k_1)$ *implies* $x = g(d^{-1}(k_1), y)$ *for all* $x \in X$, $k_1 \in K_1$, *and* $y \in Y$.

The information $d^{-1}(k_1)$ constitutes the *trap-door* enabling the receiver to decipher the message obtained. Thus, the general scenario for public key cryptography outlined above can be realized as follows. Each party $M_1, \ldots, M_\ell$ is equipped with algorithms for computing f, g, and d. Furthermore, we assume $|K_2| \geqslant \ell$. Now, party M_i chooses its private key $\tilde{k}_i \in K_2$ such that $\tilde{k}_i \neq \tilde{k}_j$ for $i \neq j$, where $i, j \in \{1, \ldots, \ell\}$. How to realize this requirement is discussed later. Then, she computes $k_i = d(\tilde{k}_i)$ and *publishes* it. The message exchange is performed using the following protocol. Suppose M_i wishes to send a message x to M_j.

(1) M_i computes $y = f(x, k_j)$ using M_j's public key k_j, and sends y over a public channel to M_j.

(2) M_j receives y and uses her private key $\tilde{k}_j$ to compute $x = g(\tilde{k}_j, y)$.

We proceed by providing an example for a concrete public key cryptosystem.

11.2. Merkle and Hellman's Public Key Cryptosystem

Within the Merkle and Hellman's public key cryptosystem plaintext is encoded into bit-vectors of length n, i.e., $b = (b_0, \ldots, b_{n-1})$, $b_i \in \{0, 1\}$. For example, we may encode the 26 letters of the Latin alphabet by using 00000 for A, 00001 for B, ..., and 11001 for Z. Thus, each letter comprises 5 bits. For error detecting, it may be recommendable to add some check bits using, for example, a Hamming code. For keeping our examples small, we neglect the issue of error detecting here and use the bit strings given above.

The public key is a knapsack vector $a = (a_0, \ldots, a_{n-1})$, $a_i \in \mathbb{N}$. How to choose a is described below. The enciphering c of a plaintext b is computed by

$$c = ab^T = \sum_{j=0}^{n-1} a_j b_j \tag{11.1}$$

(* b^T denotes the transpose of b *).

The result is a number c between 0 and $\sum_{j=0}^{n-1} a_j$. This number c is represented as a bit string of length $\ell = \lceil \log(1 + \sum_{j=0}^{n-1} a_j) \rceil$ including leading zeros.

Now we describe the trap-door information and how to choose a. The trap-door is a pair $(w, m) \in \mathbb{N} \times \mathbb{N}$ which should be large and has to satisfy

$$w < m \text{ and } \gcd(w, m) = 1 , \tag{11.2}$$

$$\text{For all } \hat{a}_j =_{df} (w^{-1} a_j) \bmod m \;\; \hat{a}_j > \sum_{i=0}^{j-1} \hat{a}_i \text{ holds for all } j = 1, \ldots, n-1 , \tag{11.3}$$

$$m > \sum_{i=0}^{n-1} \hat{a}_i . \tag{11.4}$$

Therefore, party M chooses large numbers w and m satisfying (11.2). Then, a vector $(\hat{a}_0, \ldots, \hat{a}_{n-1})$ is chosen such that Condition (11.3) is fulfilled. Furthermore, one has to check Condition (11.4). Finally, M computes $a = (a_0, \ldots, a_{n-1})$ by setting $a_i = (\hat{a}_i w) \bmod m$ for all $i = 0, \ldots, n-1$. The vector a is published, and the pair (w, m) is kept secretly.

The deciphering is done using the following procedure *dec*.

(1) Compute $\hat{c} = (w^{-1} c) \bmod m$,

(2) Compute $b = (b_0, \ldots, b_{n-1})$ as follows:

If $\hat{c} \geqslant \hat{a}_{n-1}$ then set $b_{n-1} = 1$ else set $b_{n-1} = 0$

For $j = n-2, n-3, \ldots, 0$, if

$\hat{c} - \sum_{i=j+1}^{n-1} \hat{a}_i b_i \geqslant \hat{a}_j$ then set $b_j = 1$ else set $b_j = 0$.

The following example illustrates the computation to be performed.

Example 11.1. Let $n = 5$, $m = 8443$, and $w = 2550$. Furthermore, party M chooses $\hat{a} = (171, 196, 457, 1191, 2410)$. One easily verifies Conditions (11.2) through (11.4). The published vector is then $a = (5457, 1663, 216, 6013, 7439)$. Moreover, the modular inverse of $w = 2550$ is $w^{-1} = 3950 \bmod 8443$. Let $b = (0, 1, 0, 1, 1)$; then $c = 1663 + 6013 + 7439 = 15115$, and the message sent is $c = 11101100001011$.

Now, suppose we have received 15115. Thus, we compute successively

(1) $\hat{c} = (w^{-1} c) \equiv 3950 \cdot 15115 \equiv 3797 \bmod 8443$.

(2) Since $3797 > 2410$ we set $b_4 = 1$.

(3) Next, we compute $3797 - 2410 = 1387 > 1191$. Therefore, we set $b_3 = 1$.

Now, $3797 - (2410 + 1191) = 196$ which is smaller than 457. Hence, we get $b_2 = 0$.

Furthermore, since $3797 - (2410 + 1191) = 196$ we set $b_1 = 1$.

Finally, $3797 - (2410 + 1191 - 196) = 0$, and thus $b_0 = 0$.

Exercise 38. *Prove the correctness of the deciphering procedure dec described above.*

We continue with some remarks concerning the complexity of the problems involved. The difficult problem used in the design of the trap-door function f is the knapsack or subset sum problem defined as follows. Given a number M and a vector $(a_0, \ldots, a_{n-1}) \in \mathbb{N}^n$ decide whether or not there exists a vector $(b_0, \ldots, b_{n-1}) \in \{0,1\}^n$ such that $M = \sum_{j=0}^{n-1} a_j b_j$. This problem is known to be $\mathcal{NP}$-complete (cf. Lecture 8). However, several subclasses of this problem are known for which it is easy to solve the decision problem. For example, if $a_i < a_{i+1}$ for all $i = 0, \ldots, n-2$, then the knapsack problem can be solved by using at most n subtractions as outlined in our deciphering procedure. Another subclass consists of all vectors $(a_0, \ldots, a_{n-1}) \in \mathbb{N}^n$ for which all a_i are powers of 2. In the latter case, one has simply to compute the binary representation of M. A more sophisticated subclass has been described in Lagarias and Odlyzko [3], i.e., subset sum problems having density less than .645. The density of a knapsack vector a is defined by $d(a) = \frac{n}{\log_2(\max a_i)}$. In this case, the problem is solved for almost all instances using the Lenstra/Lenstra/Lovász Basis Reduction Algorithm.

Thus, special care has to be taken. In particular, Merkle and Hellman [4] hoped that the modular transformation of the trap-door knapsack $\hat{a}$ will result in almost all cases in a hard one. And indeed, A. Shamir [6] proposed a polynomial time method for breaking the Merkle and Hellman [4] public key cryptosystem. Since then, more sophisticated methods have been proposed to design public key cryptosystem that are based on the difficulty of the subset sum problem. Since all proposed systems are not very satisfying we shall continue our course by looking at the most widely used public key cryptosystems that are based on the difficulty of number theoretic problems.

Recall our knowledge concerning the difficulty of computing discrete roots. First, assume the modulus m to be prime. For this case there exist well-known efficient probabilistic algorithms (Las Vegas type) for computing discrete roots in expected polynomial time (in the length of the input), see Theorem 5.9. Another algorithm has been proposed by Adleman, Manders and Miller [1]. In case of a composite modulus one may directly apply the Chinese Remainder Theorem for taking some discrete root provided the prime factorization of the modulus is known. Thus, in this case the problem of taking discrete roots remains feasible. On the other hand, the problem

of finding the least square root is $\mathcal{NP}$–complete. If the prime factorization of the modulus is not known there is no known algorithm for efficiently computing discrete roots. This difficulty is used in the following public key cryptosystem.

11.3. The RSA Public Key Cryptosystem

Next, we present the public key cryptosystem invented by R. Rivest, A. Shamir and L. Adleman [5] (abbr. RSA cryptosystem). Let A (= Alice) be anybody wishing to participate at the RSA cryptosystem. Then, Alice has to do the following.

(1) Choose randomly two huge primes p_A and q_A (at least 200 bits each, and the bigger prime should preferably have some more digits than the smaller one).

(2) Compute $n_A = p_A q_A$ and calculate

$$\varphi(n_A) = \varphi(p_A)\varphi(q_A) = (p_A - 1)(q_A - 1) = n_A - p_A - q_A + 1 \ .$$

(3) Choose randomly any number $e_A \in \{1, \ldots, \varphi(n_A)\}$ such that

$$\gcd(e_A, \varphi(n_A)) = 1 \ .$$

(4) Compute $d_A = e_A^{-1} \bmod \varphi(n_A)$. Publish $K_A = (n_A, e_A)$ and keep p_A, q_A, and d_A secretly.

Now, assume any other participant B (=Bob) wishing to communicate secretly with Alice. Then, Bob codes his plaintext into a binary number w. Using Alice's public key $K_A = (n_A, e_A)$ Bob calculates $c = w^{e_A} \bmod n_A$ and sends c. Alice deciphers c by computing $c^{d_A} \bmod n_A$.

Theorem 11.1. $w = c^{d_A} \bmod n_A$.

Proof. By assumption, $c = w^{e_A} \bmod n_A$. First, assume $\gcd(w, n_A) = 1$; then applying the Theorem of Euler

$$c^{d_A} \equiv (w^{e_A})^{d_A} \equiv w^{e_A d_A} \equiv w^{(e_A d_A) \bmod \varphi(n_A)} \equiv w \bmod n_A \ ,$$

since $e_A d_A \equiv 1 \bmod \varphi(n_A)$.

What can be said if $\gcd(w, n_A) \neq 1$? Suppose that exactly one of the two primes p and q does divide w, say p. Then the Little Theorem of Fermat is telling us

$$w^{q-1} \equiv 1 \bmod q \ .$$

Taking into account that $\varphi(n) = (p-1)(q-1)$, we can conclude

$$w^{\varphi(n)} \equiv (w^{q-1})^{p-1} \equiv 1^{p-1} \equiv 1 \bmod q \ .$$

Moreover, $ed \equiv 1 \bmod \varphi(n)$, and therefore $ed = j\varphi(n)+1$ for some positive integer j. Consequently,

$$w^{ed} \equiv w \bmod q \ .$$

But the last congruence is also true modulo p, since, by assumption, p divides w, and thus $w^{ed} - w \equiv 0 \bmod p$. Hence, we can conclude $w^{ed} \equiv w \bmod n$.

Finally, if both p and q divide w, then we trivially have $w^{ed} - w \equiv 0 \bmod p$ as well as $w^{ed} - w \equiv 0 \bmod q$, and thus $w^{ed} \equiv w \bmod n$. ∎

What can be said concerning the security of the RSA cryptosystem? By its construction, breaking the RSA cipher is as most as hard as finding discrete roots modulo the composite number n_A. As we have seen above, computing discrete roots must be judged as feasible if the prime factorization of n_A is known. Therefore, the cryptanalysis of the RSA cryptosystem is as most as hard as factoring. However, there is no known efficient algorithm for factoring a large composite number except for quantum computers which are currently not available (cf. Shor [7]).

However, some care must be taken be choosing the primes p and q. Obviously, the chosen primes should be nowhere listed. Thus, testing primality is such an important problem. Moreover, one has to avoid primes of special form, e.g., $p = 2^e \pm 1$. As mentioned above, the difference between p and q should be large. For seeing this, we prove the following theorem.

Theorem 11.2. *Let p and q be primes such that $p > q$ and $p - q = O((\log p)^c)$ for a "moderate" constant $c \in \mathbb{N}$. Then, there exists an efficient algorithm for factoring $n = pq$.*

Proof. Consider

$$\frac{(p+q)^2}{4} - n = \frac{p^2 + 2pq + q^2 - 4pq}{4} = \frac{(p-q)^2}{4} \tag{11.5}$$

Thus, the left side is a perfect square. Then the following method may be applied for factoring n.

(1) Compute $\sqrt{n}$ within a precision of $\lfloor (\log n + 2)/2 \rfloor$ bits.

(2) Check for $x = \lfloor \sqrt{n} \rfloor + 1,\ \lfloor \sqrt{n} \rfloor + 2,\ \ldots$ whether or not $x^2 - n$ is a perfect square. If it is, output $p = x + \sqrt{x^2 - n}$ and $q = x - \sqrt{x^2 - n}$.

The correctness of the above algorithm can be shown as follows. First, assume that $\sqrt{n}$ has been computed within a precision of $\lfloor (\log n + 2)/2 \rfloor$ bits. Then we have $\lfloor \sqrt{n} \rfloor$. (Exercise !). Next, observe that

$$\frac{p+q}{2} > \sqrt{pq} = \sqrt{n} \tag{11.6}$$

by applying the well-known inequality between arithmetic and geometric mean. Consequently, $\lfloor \sqrt{n} \rfloor < (p+q)/2$ (* note that $(p+q)/2$ is always an integer *). Thus, the algorithm must terminate by finding a perfect square and the first x found must fulfill $x^2 \leqslant \frac{(p+q)^2}{4}$ by (11.5) and (11.6). Thus, $x \leqslant \frac{p+q}{2}$.

Case 1. $x = \dfrac{p+q}{2}$.

Let $z^2 = x^2 - n$. By (11.5) we directly obtain $z = \frac{p-q}{2}$. Hence, $x + z = p$ and $x - z = q$.

Case 2. $x < \dfrac{p+q}{2}$.

Since $z^2 = x^2 - n$ we obtain $n = x^2 - z^2 = (x+z)(x-z)$. Thus, $p = x + z$ and $q = x - z$, and therefore, $(p+q)/2 = x$, a contradiction.

This proves the correctness.

Finally, we show that the algorithm above is efficient. The computation of $\sqrt{n}$ up to the desired precision can be easily performed using the Newton method, i.e.,

$$x_{\ell+1} := x_\ell - \frac{x_\ell^2 - n}{2x_\ell} \text{ for } \ell = 0, \ldots, \lfloor (\log n + 2)/2 \rfloor$$

with $x_0 = n$. Taking into account that $\sqrt{n} > q = p - (p - q) \approx p - (\log p)^c$ we see by (11.5) that

$$\frac{p+q}{2} = p - \frac{p-q}{2} \approx p - \frac{(\log p)^c}{2} > \sqrt{n}\,. \tag{11.7}$$

Hence, $(p+q)/2$ is only "a bit" greater than $\sqrt{n}$. Consequently, the algorithm has to try at most $(\log p)^c/2$ many candidates until its search terminates. But this is a polynomial in the length of the input, and hence we are done. ∎

Next, we formally formulate the problem of finding discrete logarithms.

Problem 11.4. Discrete Logarithms

Input: Prime number $p \in \mathbb{N}$, $b \in \mathbb{Z}_p^*$, and a generator α for $\mathbb{Z}_p^*$.

Problem: Compute the index x such that $x = \operatorname{dlog}_\alpha b$.

So far, there is no known algorithm efficiently computing discrete logarithms for any standard model of sequential or parallel computation. On the other hand, Quantum computer would be able to compute discrete logarithms in polynomial time (cf. Shor [7]).

Next, we present a public key cryptosystem based on the difficulty to compute discrete logarithms.

11.4. The Diffie-Hellman Public Key Cryptosystem

This public key cryptosystem requires a system designer. The system designer chooses a huge prime q (preferably more than 1000 bits) and a generator α for $\mathbb{Z}_q^*$. The prime q and the generator α are global information, and thus known to everybody participating in this public key cryptographic system.

Next, we describe how the public and the secret key, respectively, are chosen by any participant. Subsequently, we provide the protocol used.

Let A be any participant. A chooses randomly a number $x_A \in \{1, \ldots, q-1\}$ and computes $y_A = \alpha^{x_A} \bmod q$. That is, $x_A = \mathrm{dlog}_\alpha y_A$. Participant A publishes y_A as her public key and keeps x_A secretly.

Now, let A and B (= Bob) be any two participants who wish to communicate. Assume, B likes to send a message to A. Then the following key is used.

B computes the key $K_{AB} = y_A^{x_B} \bmod q$.

Assume, A likes to send a message to B. Then, A computes $K_{BA} = y_B^{x_A} \bmod q$.

The following theorem shows the usefulness of these keys.

Theorem 11.3. $K_{AB} = K_{BA}$

Proof.

$$\begin{aligned} K_{AB} &= y_A^{x_B} \equiv (\alpha^{x_A})^{x_B} \equiv \alpha^{x_A x_B} \\ &\equiv (\alpha^{x_B})^{x_A} \equiv y_B^{x_A} \equiv K_{BA} \bmod q \ . \end{aligned}$$

∎

Note, however, that A and B compute K_{BA} and K_{AB}, respectively, using *different* information. That is, A computes K_{BA} using her own secret key x_A and the public key y_B published by B while Bob calculates K_{BA} from his secret key x_B and Alice's public key y_A. On the other hand, any cryptanalyst does neither possess x_A nor x_B, hence she must compute K_{BA} solely from y_A and y_B. Since $K_{AB} = y_A^{\mathrm{dlog}_\alpha y_B} \bmod q$ this is at most as hard as computing discrete logarithms. Moreover, so far no easier method is known for computing K_{BA} from y_A and y_B. Therefore, it is widely believed that computing K_{BA} from y_A and y_B is hard.

Next, we describe how the parties A and B can communicate. Taking Theorem 11.3 into account, two possibilities are imaginable. First, the system designer additionally provides any sufficiently advanced classical two way cryptosystem, for example the DES. Then, any pair of users wishing to secretly communicate may use the key K_{BA}. Thus, *no key exchange* is required in *advance*. Note that public key cryptosystems are relatively slow compared to classical cryptosystems (at least to our present stage of technology and theoretical knowledge). Thus, it is sometimes more realistic to use them in the limited role in conjunction with a classical cryptosystem in which the actual messages are transmitted as described above.

Second, the communication is performed by using directly the Diffie-Hellman system. The underlying idea is best explained using a bag having two locks. Each partner possesses exclusively one key for one of the two locks. Initially, both locks are unlocked. Now, A puts the message w into the bag and locks *her lock* with *her key*. The bag is taken by a messenger who delivers the bag to B. Obviously, B is *not* able to *unlock* the bag right now, since he posses only the key for the other lock. Therefore, B locks *his* lock using *his* key, and returns the bag to the messenger who is returning it to A. Now, A may *unlock* her lock, but the bag remains anyway *locked*.

Finally, the messenger delivers the bag again to B. Now, B may *unlock* the bag using *his* key, and thus, he finally has access to the secret message w.

The protocol described above has the following advantage. The public messenger always delivered a locked bag. On the other hand, A and B could exchange a secret message without *exchanging* any *key*.

The impact of this method can be hardly overestimated. Looking back into the history of cryptography, we see that the cryptography community unanimously agreed, for thousands of years, that the only way for two parties to establish secure communications was to first exchange a secret key. This was so much common wisdom that nobody questioned it. If the recipient did not have a secret key giving her the information needed to encrypt the message efficiently, how could she be in a better position than an eavesdropper?

As we have mentioned, this idea got published in 1976. Interestingly enough, there is an old Bell Labs paper from October, 1944, titled "Final Report on Project C43", describing a clever method of secure telephone conversation between two parties without any pre-arrangement. If John calls Mary, then Mary can add a random amount of noise to the phone line to drown out John's message in case any eavesdroppers are listening. However, at the same time Mary can also record the telephone call, then later play it back and subtract the noise she had added, thereby leaving John's original message for only her to hear. While there were practical disadvantages to this method, it suggested that the logical possibility existed: there might be methods of establishing secure communications without first exchanging a shared secret key.

And in 1997 it was revealed that there has been secret work in the UK Goverment Communication Headquartes in the early seventies done by Ellis, Cocks and Williamson. Ellis started thinking about the problem to avoid a secret key in the sixties. In this time he also discovered the old Bell Labs paper from 1944 mentioned above which motivated him even more. He then developed an existence proof that the concept of avoiding a secret key was possible with mathematical encryption. His findings were published in a secret CESG report titled "The Possibility of Non-Secret Encryption" in January 1970. The quest was then to find a practical example.

The first workable mathematical formula for non-secret encryption was discovered by Clifford Cocks, which he recorded in 1973 in a secret CESG report titled "A Note on Non-Secret Encryption". This work describes a special case of the RSA algorithm, differing in that the encryption and decryption algorithms are not equivalent, and without mention of the application to digital signatures. A few months later in 1974, Malcolm Williamson discovered a mathematical expression based on the commutativity of exponentiation that he recorded in a secret report titled "Non-Secret Encryption Using A Finite Field", and which describes a key exchange method similar to that discovered by Diffie, Hellman, and Merkle. It is not known to what uses, if any, the GCHQ work was applied.

Now, we outline the formal realization of the idea described above to use a bag with two locks. For that purpose, a small modification of the choices for x_A and x_B,

respectively, has to be made. Again, A and B randomly choose a number x_A and x_B between 1 and $q-1$. *Additionally*, they must ensure that $\gcd(x_A, q-1) = 1$ and $\gcd(x_B, q-1) = 1$, respectively. This can be easily done by using the Euclidean algorithm, i.e., if the randomly chosen number does not fulfill this requirement a new number is randomly chosen until one is found that is relatively prime to $q-1$. Moreover, as shown in the proof of Theorem 3.4, the Euclidean algorithm can be also used for computing $x_A^{-1} \bmod (q-1)$ and $x_B^{-1} \bmod (q-1)$, respectively.

Suppose, A wishes to send B a message, and let w be A's plaintext.

(i) A sends $w^{x_A} \bmod q$ to B,

(ii) B returns $w^{x_A x_B} \bmod q$ to A,

(iii) A computes $\hat{w} \equiv (w^{x_A x_B})^{x_A^{-1}} \bmod q$ and sends the result $\hat{w}$ to B,

(iv) B deciphers $\hat{w}$ by calculating $\hat{w}^{x_B^{-1}} \bmod q$.

The correctness of the above algorithm is an immediate consequence of the Theorem of Euler.

This finishes our lecture, and the only thing remaining is the midterm problem for cryptology. Below three problems are provided to warm you up, and you are requested to solve at least two. Then, we present the midterm problem for cryptology which you should solve to prove your successful participation in this course.

Advanced Exercises

Advanced Exercise 1. *Prove or disprove the following: If the coefficient d_A for deciphering received messages within the RSA cryptosystem could be computed in deterministic polynomial time from the public key K_A then one could factor n_A in probabilistic polynomial time.*

Advanced Exercise 2. *Consider the following knapsack vector $a = (a_1, \ldots, a_n)$ with $a_i = P/p_i$, where the p_i are pairwise distinct primes and $P = \prod_{i=1}^{n} p_i$.*

Prove or disprove: Using such knapsack vectors, one can design a public key cryptosystem of the Merkle and Hellman type presented in Lecture 11. If applicable, discuss which information should be kept secretly, and which information may serve as public key. Describe the enciphering and deciphering algorithm, respectively. What can you say concerning the security of your cryptographic system. Otherwise prove the inappropriateness of the proposed knapsack vectors.

Advanced Exercise 3. *Consider the following public key cryptosystem. A huge prime p is known to every participant. Each member chooses e and d such that $ed \equiv 1 \bmod (p-1)$. The message transfer is done by the same protocol as in the*

Diffie and Hellman public key cryptosystem. Discuss the usefulness of this systems as well as its security.

Midterm Problem for Cryptology

The following so-called midterm problem is an exercise you *should* solve. There will also be a final problem.

Midterm Problem. *A user of the RSA cryptosystem has published the following information:*

$\mathsf{K} = (32193235888665727, 147893)$. *You have eavesdropped the following message:*

0101101111110000011011100110100000011011100000101101111

1010101010111101010000010011001011101011010000100100111

1011001111001100001111000101111001010010000000010011010

0000110100000111000010111101111100110111001111001110001

and wonder what does it mean. Fortunately for you, this user has ignored at least one of the advices given throughout Lecture 11, so you have a good chance for breaking the code. Good Luck!

References

[1] L. ADLEMAN, K. MANDER AND G. MILLER (1977), On taking roots in finite fields, *in* Proc. 18th Annual Symposium on Foundations of Computer Science, pp. 175 – 177, IEEE Computer Society Press.

[2] W. DIFFIE AND M.E. HELLMAN (1976), New directions in cryptography, *IEEE Transactions on Information Theory* **IT-22** No. 6, 644 – 654.

[3] J.C. LAGARIAS AND A.M. ODLYZKO (1983), Solving low-density subset sum problems, *in* Proc. 24th IEEE Annual Symposium on Foundations of Computer Science.

[4] R.C. MERKLE AND M.E. HELLMAN (1978), Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory* **IT-24**, 525 – 530.

[5] R. RIVEST, A. SHAMIR AND L. ADLEMAN (1978), A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21**, 120 – 126.

[6] A. SHAMIR (1982), A polynomial time algorithm for breaking the Merkle-Hellman cryptosystem, *in* Proc. 23rd IEEE Annual Symposium on Foundations of Computer Science, 145 – 152.

[7] P.W. SHOR (1994), Algorithms for quantum computation: Discrete log and factoring, *in* Proc. 35th Annual Symposium on Foundations of Computer Science, pp. 124 – 134, IEEE Computer Society Press.

Lecture 12: Authentication, Cryptographic Protocols

We finished the last lecture by showing how realize the idea to use a bag with two locks. Though our solution looked good, it is not perfect. The problem is a possible attack by a third party in the middle between Alice and Bob. Thus, we need authentication.

12.1. Authentication

The security of the communication between two parties can be affected as follows. Suppose, an eavesdropper C (=Claire) could pretend to be the true receiver. Then she could easily apply the same procedure as Bob does using her own numbers x_C and x_C^{-1}. Looking at our informal description of how to use the bag and the two locks given in Lecture 11, this would refer to the scenario that both parties have a lock and a key for it. Thus, as long as Alice cannot be sure that the lock on the bag is really Bob's there is always the danger that some eavesdropper has applied her lock. This leads us directly to the problem of *authentication*. That is, we are looking for a method that may serve as a *digital signature* but it should be more resistant to forgery than the usually used hand written signatures. The following method is due to El-Gamal and based on the Diffie-Hellman public key cryptosystem.

Recall that q is a huge prime, and that α is a generator of $\mathbb{Z}_q^*$. Furthermore, Bob's public key is y_B.

To send his signature S, Bob chooses a random integer k with $\gcd(k, q-1) = 1$. Then Bob calculates $r = \alpha^k \bmod q$ and solves the following congruence for the unknown x: $\alpha^S \equiv y_B^r r^x \bmod q$, and sends Alice the pair (r, x) along with S.

Now, Alice can easily verify that $\alpha^S \equiv y_B^r r^x \bmod q$, and she is happy, secure in her confidence that Bob did send the message S.

It remains to argue that Bob can perform efficiently all the computations necessary, and that Alice can be really happy.

Obviously, r can be efficiently computed using the algorithm described in the proof of Theorem 4.3 and so can α^S. But what about solving $\alpha^S \equiv y_B^r r^x \bmod q$? Taking into account that $y_B \equiv \alpha^{x_B} \bmod q$, we obtain, by putting it all together:

$$\alpha^S = \alpha^{x_B r} \alpha^{kx} \equiv \alpha^{x_B r + kx} \bmod q \ . \tag{12.1}$$

Thus, applying the Theorem of Euler to (12.1) we obtain the condition

$$S \equiv x_B r + kx \bmod (q-1)$$

and hence, $x \equiv (S - x_B r)k^{-1} \bmod (q-1)$. Now, we see why the number k has been required to satisfy $\gcd(k, q-1) = 1$. Clearly, all the needed computations can be efficiently performed by Bob.

Finally, Alice can be sure to have obtained Bob's signature, since solving the congruence $\alpha^S \equiv y_B^r r^x \bmod q$ in order to determine x requires the knowledge of x_B which is kept secretly by Bob. Forging Bob's signature is as complicated as computing discrete logarithms. Now, we also understand why the parties have to publish their key y. It is either for using it for the key exchange described in the previous lecture, or for just performing the authentication as described above.

So far, we have provided an authentication scheme for avoiding an attack by a third party in the middle between Alice and Bob.

The only thing that kind of remains open is how S is chosen by Bob. Using the protocol provided above, the best choice is to use the whole message Bob wishes to send as signature S. One advantage of the protocol described above is the probabilistic element introduced by the random choice of k. On the other hand, there is also a serious disadvantage, since the ciphertext to be send it now blown up.

Before we are going to look at other digital signature schemes, we would like to put them in the more general context of cryptographic protocols which are defined as follows. We shall return to digital signature in Lecture 14.

12.2. Cryptographic Protocols

Definition 12.1. ***Cryptographic protocols*** *describe algorithms used for the communication between different parties, adversaries or not.*

By definition, cryptographic protocols apply cryptographic transformations. Consequently, they are at most as secure as the underlying cryptosystem. Usually, we shall use public key cryptosystems for cryptographic protocols. However, the goal of the protocol is usually something beyond the simple secrecy of message transmission. For example, the communicating parties may want to share parts of their secrets to achieve a common goal, or they like to convince the other parties that they know a particular secret *without* providing even a single bit of the secret on hand. Protocols realizing such goals have considerably changed our understanding about what is impossible when several parties, adversaries or not, are communicating with each other.

For seeing how digital signatures fit into the domain of protocols just consider the following very general task. A private conversation should be established between two individual users of an information system or a communication network. We do not make any assumption concerning whether or not these two individual users have ever communicated with each other before. Clearly, having a public key cryptosystem on hand, we can solve this problem. First, our users publish their *public key*. Then, messages send to user A are encrypted by using A's public key. But even if the cryptosystem is considered to be secure, we still have to deal with the problem that a user C might pretend to be the user B when sending a message to A. To prevent the occurrence of such situations, some convention of *signing* messages has to be added to the protocol.

Before going into details, some more remarks are mandatory here. One always has to *separate* security properties of the underlying cryptosystem from those of the protocol. When doing this, the possible adversaries should be kept in mind. In most communication protocols, an adversary belongs to one of the following three types.

(1) Communicating parties who try to cheat. Later we shall meet two types of cheaters, i.e., passive and active.

(2) Passive eavesdroppers. They may obtain information not intended for them, but are otherwise harmless.

(3) Active eavesdroppers. Besides obtaining secret information (as passive eavesdroppers do), they may mess up the whole protocol.

In our problem above where C tries to impersonate B we have an adversary of Type (3), i.e., an active eavesdropper. For having an example for Type (1), just imagine that some people like to play poker by telephone. Clearly, somebody might by tempted to cheat. We shall come back to this point later. Looking at typical application of cryptography, it should be also clear that adversaries of Type (2) may cause huge trouble, e.g., in military or diplomatic applications, or in banking.

For the sake of illustrating the difference between an active and passive eavesdropper let us look at the RSA cryptosystem. We claim that it vulnerable against attacks with chosen ciphertext.

This can be seen as follows. Suppose an eavesdropper E has received

$$c = m^e \bmod n \ .$$

Of course, now the eavesdropper is interested in knowing m. Let A be the legal receiver of m. It is meaningful to assume that A will not decrypt c for E (otherwise there wouldn't be any secure cryptosystem). But now, E can modify c as follows.

Using some randomly chosen $x \in \mathbb{Z}_n^*$ and computing

$$\hat{c} = cx^e \bmod n = \hat{m}^e \bmod n \ ,$$

E can send $\hat{c}$ to A. Suppose E additionally succeeds to get A to decrypt this message for him. Then E gets $\hat{m}$, and thus he knows the original message m, too, since

$$\hat{c} = cx^e \bmod n = m^e x^e \bmod n = (mx)^e \bmod n \ .$$

Hence, by construction of RSA, E knows that $\hat{m} = mx \bmod n$. So all what is left is to multiply $\hat{m}$ by the modular inverse x^{-1} of x modulo n.

This is a good point to highlight another potential danger of using plain public key cryptosystems. In the preceding example we have not said how E succeeds to get A to decrypt $\hat{c}$ for him. Assuming A is unexperienced, E may have pretended to be B (the original sender). Then, after having encrypted $\hat{c}$, the legal receiver A was confused,

since $\hat{\mathsf{m}}$ did not make any sense to her. Thus, A sent $\hat{\mathsf{m}}$ back to E instead of sending it to B. As we have seen, this is very *dangerous* and should be *avoided at all.* A much better way to recover from the confusion would have been to send just a request to B to resend the message.

For making the need of protocols more transparent, we provide two more examples of weak points that may occur when using plain RSA which also apply *mutatis mutandis* to many other plain public key cryptosystems.

Suppose Alice wants to send orders to her stock broker Bob. An eavesdropper would like to know Alice's order. Furthermore, suppose the eavesdropper has good reason to belive that m is one of the following three messages:

- $\mathsf{m}_1 =$ "buy IBM"
- $\mathsf{m}_2 =$ "sell IBM"
- $\mathsf{m}_3 =$ "hold IBM"

The eavesdropper can compute the encryptions c_1, c_2, and c_3 of the three messages for himself, and when Alice is sending an encryption of one of these three messages, say m_2, the eavesdropper simple compares the ciphers and knows it is m_2. This example shows that plain RSA can leak partial information.

Next, suppose Alice wants to submit a number m, representing her bid, to Bob. Bob is accepting many bids, and will choose the lowest bid.

Suppose the eavesdropper is a competitor, too, and wants to underbid Alice by 10%. If we make the reasonable assumption that Alice's bid is made in round numbers and thus amounts to be a multiple of 10. Then the eavesdropper can intercept Alice's *encrypted* message c. Now, he computes

$$\hat{\mathsf{c}} = \mathsf{c} \cdot \left(9 \cdot 10^{-1}\right)^{\mathsf{e}} \bmod \mathsf{n} \ ,$$

where 10^{-1} denotes the modular inverse of 10 modulo n. This inverse exists, since n is the product of two large primes and $10 = 2 \cdot 5$. Hence $\gcd(10, \mathsf{n}) = 1$. So, we indeed have $\hat{\mathsf{m}} = 0.9 \cdot \mathsf{m}$. In this way, Alice's competitor can underbid Alice by 10%, *without* knowing anything about the value of Alice's bid.

Thus flipping some bits in the ciphertext will also flip some bits in the message. This type of weakness is usually called ***malleability***, and is a weakness that should not occur.

More generally speaking, encryption is often identified with "secure envelope" or a locked box that cannot be opened without destroying it. This metaphor has a very compelling and convenient touch and is often used by engineers. However, an encryption scheme can at best approximate a "secure envelope." This is a fundamental fact, and we provide some more arguments to support it.

Ciphertexts are bit strings (electronically represented) and not physical envelopes. This is obvious and trivial, but one has to think about the consequences.

- First, the bit string representing a ciphertext can be *observed* by an eavesdropper. An ideal "secure envelope" leaks no information about the message it contains. For example, if Alice sends two messages to Bob by using a "secure envelope," an eavesdropper cannot tell whether or not these messages are identical or not. The same should hold then for an encryption scheme. But this requirement alone rules out any *deterministic* encryption scheme, i.e., encryption schemes that encrypt the same message always in the same way.

- Second, ciphertexts can easily be *replicated*, whereas messages contained in "secure envelopes" cannot. There is really nothing we can do about this. Thus, higher level protocols using encryption must deal with the fact that this can happen.

- Third, ciphertexts can easily be modified, creating other ciphertexts as we have seen above. We can do many things to a ciphertext such as flipping some bits from '1' to '0' or vice versa. Even if an encryption scheme is secure, as we have seen, flipping bits in the ciphertext may flip bits in the message. Using the terminology introduced above, we see that malleability cannot be tolerated in many applications. Obviously, malleability has no counterpart in the world of ideal "secure envelopes."

- Fourth, any bit string is potentially a ciphertext, i.e., the encryption of some message. As we have seen, the fact can also be misused by an adversary who actively participates in a protocol by sending its own messages to other parties. Such a *chosen ciphertext* attack has also no counterpart in the world of ideal "secure envelopes."

Clearly, we want to overcome these difficulties. But this is easier said than done. For example, when thinking about avoiding attacks from a man in the middle, one can start from the idea that a receiver should acknowledge the receipt of the encrypted message. Thus, if a third party has pretended to be A and has send a message to B, but B is acknowledging it to A, A can immediately inform B that something is wrong.

Suppose E_A, E_B, E_C, ... are the public encryption algorithms of parties A, B, C, ... and D_A, D_B, D_C, ... are the decryption algorithms kept secretly by A, B, C, ... Furthermore, let the following protocol be agreed upon. For sending a message from A to B the following steps have to be performed.

(1) A sends the triple $(\mathsf{A}, \mathsf{E}_\mathsf{B}(w), \mathsf{B})$ to B, where w is the message.

(2) B deciphers w by using D_B and sends the triple $(\mathsf{B}, \mathsf{E}_\mathsf{A}(w), \mathsf{A})$ back to A.

At first glance, this protocol looks well designed. But there are some dangers with it.

Suppose an active eavesdropper C who has caught the message for B. Since he is knowing the structure, he is changing the triple $(\mathsf{A}, \mathsf{E}_\mathsf{B}(w), \mathsf{B})$ to $(\mathsf{C}, \mathsf{E}_\mathsf{B}(w), \mathsf{B})$

and sends it to B. Following the protocol, B returns the message $(\mathsf{B}, \mathsf{E}_\mathsf{C}(w), \mathsf{C})$ to C. Consequently, C can decipher it and knows w. Of course, A is waiting for the acknowledgment and may inform B that it did not arrive. Now, B and A can realize that something went wrong, but it is too late. C already does possess w.

A better variant might be the following *Challenge-Response* protocol.

Assumption: A and B have a common secret key k and have agreed to use the cryptosystem f.

A is communicating with someone from whom she expects it is B. For verifying this, the following protocol is used.

(1) A randomly generates a number r and sends it to B.

(2) The communication partner (hopefully B) encrypts r by using the secret key k in the cryptosystem f and sends $\mathsf{f}(\mathsf{r}, \mathsf{k})$ back to A.

(3) A computes $\mathsf{f}(\mathsf{r}, \mathsf{k})$ by himself and compares the computed value with the received one. If they are identical, A assumes that she is indeed communicating with B. If the values are not equal, A concludes that her partner is *not* B.

Please think about this protocol and its security.

We continue our course by looking at some more cryptographic protocols. In order to have an example for a more complex protocol we are going to explain how to play poker per telephone, how to flip a coin per telephone, and how to realize partial disclosure of secrets.

We start with the poker protocol.

12.3. Playing Poker per Telephone

Before elaborating the protocol, we have to think about the demands that such a protocol should fulfill. So, we continue by listing the necessary demands. Note that we do not claim this list to be exhaustive.

(i) All hands (sets of five cards) are equally likely.

(ii) The hands of player A and B are disjoint.

(iii) Both players know their own cards but have no information about the opponent's hand.

(iv) It is possible for each of the players to find out the eventual cheating of the other player.

Next, we propose a protocol. A cryptosystem, classical or public-key is used. However, neither the encryption methods E_A and E_B nor the decryption methods D_A

and D_B are publicized. Furthermore, we assume *commutativity* in any composition of E's and D's. The mutual order is immaterial.

Before the actual play, both players A and B agree about the names $w_1, \ldots, w_{52}$ of the 52 cards. The names are chosen in a way such that the cryptosystem is applicable in the sense needed in the sequel. For instance, if E_A and E_B operate on integers in a certain range then each w_i, $i = 1, \ldots, 52$, should be an integer in this range.

Now we are ready to describe the protocol. Player A acts as the dealer but the roles of A and B can be interchanged. The protocol consists of the following five steps.

Protocol Poker

Step 1: Player B shuffles the cards, encrypts them using E_B, and sends them to A. That is, player A receives a random permutation of $E_B(w_1), \ldots, E_B(w_{52})$.

(* A can now only verify that all cards are in the game *)

Step 2: Player A chooses 5 cards from the sequence received at random and sends them back to B as they are. These 5 cards are B's hand. A also encrypts them by using E_A and sends them to B for checking purposes.

Step 3: Player A again chooses 5 cards from the remaining cards and encrypts them by applying E_A. The result is again sent to B, i.e., B receives $E_A(E_B(w_{i_j}))$, $j = 1, \ldots, 5$.

(* these five cards will be A's hand. *)

Step 4: Player B applies to these five cards $E_A(E_B(w_{i_j}))$, $j = 1, \ldots, 5$, its own deciphering algorithm D_B. Then he sends the result back to A. That is, A receives

$$D_B(E_A(E_B(w_{i_j}))) = E_A(D_B(E_B(w_{i_j}))) = E_A(w_{i_j}) \ .$$

(* that is the point where we need commutativity *)

Step 5: Player A applies its own deciphering algorithm D_A to the five cards $E_A(w_{i_j})$. Now, he also nows his hand and the game starts.

Let us now see how Requirements (i) through (iv) are fulfilled. As already stated, both players know their own hand. The hands will also be disjoint. B can immediately check that the items given in Step 3 are different from those received in Step 2.

No conclusive evidence can be presented concerning the remaining Requirements from (i) through (iv). The matter largely depends on how truly one-way functions have been chosen for the encryption algorithms E_A and E_B. For example, it might be impossible to find w_i on the basis of $E_B(w_i)$ but, still, some partial information about w_i could be found. If, for instance, w_i is a sequence of bits, the last bit could be found from $E_B(w_i)$. Such partial information could tell A that all aces are within a certain subset of $E_B(w_1), \ldots, E_B(w_{52})$. Then, he clearly would deal B's cards

from outside this subset and his own cards from inside the subset. In this case also Requirement (i) and (iii) would be partially violated.

These reflections also show why all algorithms E_A and E_B as well as D_A and D_B must be kept secretly. Otherwise, A could also compute $E_B(w_1), \ldots, E_B(w_{52})$ and would have perfect knowledge about the cards.

We can also derive a conclusion concerning the plaintext space of any public-key cryptosystem. It must be so huge that no one can encrypt the possible plaintexts in advance and can perform decryption by simply searching through all resulting ciphertexts.

For further illustration of the difficulties to prove that our requirements are fulfilled, let us consider a more concrete scenario.

Let us assume that A and B have agreed about a huge prime p and to represent the cards as numbers chosen from $\{2, \ldots, p-1\}$. Each player chooses secretly for himself an encryption and decryption exponent e_A, d_A and e_B, d_B, respectively, such that

$$e_A \cdot d_A \equiv e_B \cdot d_B \equiv 1 \bmod p - 1 \ .$$

Then encryption and decryption are done in an RSA like fashion, i.e., $E_I(w) = w^{e_I} \bmod p$ for $I = A, B$ and decryption of a cipher c is done by computing $D_I(w) = c^{d_I} \bmod p$ for $I = A, B$. Then, we can prove the following claim.

Claim. The property to be or not to be a quadratic residue is inherited when using this type of encryption.

Proof. Let w be a quadratic residue modulo p. Then we have $\left(\frac{w}{p}\right) = 1$, where $\left(\frac{w}{p}\right)$ denotes the Legendre symbol. By the theorem of Euler, we then know

$$\left(\frac{w}{p}\right) \equiv w^{\frac{p-1}{2}} \equiv 1 \bmod p \ .$$

Therefore, we also have

$$\left(\frac{w^{e_A}}{p}\right) \equiv (w^{e_A})^{\frac{p-1}{2}} \equiv \left(w^{\frac{p-1}{2}}\right)^{e_A} \equiv 1^{e_A} \equiv 1 \bmod p \ .$$

That is, w^{e_A} is a quadratic residue modulo p if and only if w is a quadratic residue modulo p. ∎

If one player has discovered this property she can cheat the other player. For example, the numerical values of the four aces may be all a quadratic residue modulo p. When using the protocol above, clearly A will never send a quadratic residue modulo p to B. Again, the hands are no longer equally likely and (iii) is also violated.

This simple example shows that one cannot take too much care. It is very complicated to prove non-trivial theorems about the security of protocols. We shall come back to this issue later. Before doing it, we shall have a look at other protocols.

Lecture 13: More Cryptographic Protocols

As already mentioned in the last lecture, we aim to take a closer look at some more advanced ccryptographic protocols. We start this lecture by looking at the problem to flip a coin per telephone.

13.1. Flipping a Coin per Telephone

As a matter of fact, the coin flipping problem is the problem which initiated the whole area. In 1981, Manuel Blum presented this problem and a solution to it (cf. Blum [2]).

Blum described the scenario as follows. Suppose Alice and Bob and are going to get a divorce. They already live in cities far apart of each other and they don't want to see each other again. For deciding who will obtain the new car, they have agreed to flip a coin. Of course, they don't like to make their choice, say choosing head, and then hearing from the other end of the phone: "I am flipping the coin,. . . , the outcome is tail. I am so sorry for you."

This scenario shows already the main difficulty. We do not only have to realize the coin flip but also a method for verifying its outcome by the other party.

So, how can we attack this problem. Let us have a look at the proposals made.

Proposal 1 (Blum/Micali)

Let X be a finite set of numbers containing as much even numbers as odd ones, and let $f: X \to Y$ be a one-way function. Furthermore, assume Alice and Bob have agreed to use f. Then, the following protocol is used.

Step 1: Alice chooses at random an element $x \in X$, computes $y = f(x)$ and sends y to Bob.

Step 2: Bob guesses whether or not x is even or odd and sends his guess to Alice.

Step 3: Alice tells Bob whether or not his guess was right and proves her claim by sending x to Bob, too.

Step 4: Bob verifies Alice's claim by computing $f(x)$ and comparing it to y.

At first glance, this protocol looks good. But we are already warned. So, let us ask if a participant of this protocol can cheat.

For doing it, we assume that f is indeed a one-way function (cf. Definition 11.1). This is a good place to see why we have required one-way functions to be injective. If not, there could be two numbers x and x' such that x is even, x' is odd and $f(x) = f(x')$.

But still, the definition of one-way function does not imply that we cannot compute that last bit of x. If we could, we already have to whole information needed.

So, we cannot prove anything about the protocol above. Therefore, Blum [2] proposed the following more advanced protocol for flipping a coin per telephone.

Protocol *CF*

Step 1: Alice chooses two huge primes p and q sends their product $n = pq$ to Bob.

Step 2: Bob chooses randomly a number s from $\{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$. Furthermore, he computes $z = s^2 \bmod n$ and sends z to Alice.

Step 3: Alice computes the four discrete roots $\pm x$ and $\pm y$ of z modulo n. Let x' be the smaller number of $x \bmod n$ and $-x \bmod n$ and let y' be defined analogously.

Step 4: Alice looks for the smallest bit position i in which x' and y' differ. Then she guesses one of these numbers and communicates her guess to Bob by telling him: "The ith bit of your number is 0" and "The ith bit of your number is 1," respectively.

Step 5: Bob tells Alice whether or not her guess was correct.

Step 6: Bob sends his number s to Alice.

Step 7: Alice tells Bob the factorization of n.

This protocol looks more complex than the previous one. It is also not obvious whether or not it is correct and fair. Thus, we have to analyze it carefully and have also to check whether or not it is secure. Finally, we have tell whether or not all steps can be executed efficiently.

If Alice is making her guess randomly and if Bob is choosing his number s indeed randomly from the set $\{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$, then the probability that Alice wins is clearly $1/2$.

Furthermore, it is not a good idea for Bob not to choose his number randomly (provided the protocol is executed repeatedly), since a certain preference for some numbers would offer Alice a possibility to possibly increase her chance of winning.

So, the most important question we have to study right here is whether or not Bob can possibly cheat, if he is changing s after having sent z to Alice. In order to avoid being detected as cheater, Bob should possess x' as well as y'. Taking into account that

$$\begin{aligned} (x')^2 &\equiv z \bmod n \\ (y')^2 &\equiv z \bmod n \text{ , we get} \\ (x')^2 - (y')^2 &\equiv 0 \bmod n \text{ .} \end{aligned}$$

Furthermore, we have $x' \not\equiv y' \bmod n$. This clearly implies $x' - y' \not\equiv 0 \bmod n$. Additionally, it is not hard to see that we also have $x' + y' \not\equiv 0 \bmod n$. Thus, putting it all together we directly arrive at

$$(x')^2 - (y')^2 \equiv (x' - y')(x' + y') \equiv 0 \bmod n \text{ .}$$

This is possible if and only if

$$\begin{aligned} \gcd(n, x' + y') &= p \text{ or} \\ \gcd(n, x' + y') &= q \, . \end{aligned}$$

Thus, if Bob is able to cheat he is able to factorize n, too. Therefore, the security of our second protocol is based on the difficulty to factorize. We summarize our knowledge by the following theorem.

Theorem 13.1. *The protocol **CF** is secure provided factoring is difficult.*

We have elaborated this point here in some more detail, since it also shows why Alice is sending just one bit in Step 4 and not x' or y'.

So, it remains to show that the protocol can be executed efficiently. Since Alice is knowing the factorization of n, it suffices to argue that Alice can efficiently compute discrete square roots modulo a prime. Again, we refer to Lecture 5, where we studied Berlekamp's [1] procedure for taking discrete square roots modulo a prime. This algorithm is a Las Vegas method and has an expected running time that is polynomially bounded in the length of the input a and the modulus p.

Next, we look at another problem of high practical relevance, i.e., partial disclosure of secrets.

13.2. Partial Disclosure of Secrets

Suppose two or more persons possess a *secret*. In order to achieve a common goal, they have to exchange parts of their *secrets without disclosing the secret itself*.

More formally, we can model this situation as follows.

Let $A_1, \ldots, A_t$, $t \geqslant 2$, be the participants. They all know the definition of a function $f(x_1, \ldots, x_t)$. Each variable can take values from the set $\{1, \ldots, m\}$, where $m \in \mathbb{N}^+$. The values of f are natural numbers, too. Thus, we could present f in a table. Now, each participant A_i is knowing a specific value $a_i \in \{1, \ldots, m\}$ but does *not* possess any information about a_j, $j \neq i$. The common goal the participants wish to achieve is to compute

$$f(a_1, \ldots, a_t)$$

without disclosing any of the values a_i to any of the participants.

That is, we want to design a protocol achieving the following. After execution of the protocol, each participant A_i is knowing $f(a_1, \ldots, a_n)$ but non of the participants has disclosed more information about his a_i than can be deduced from knowing $f(a_1, \ldots, a_t)$.

Clearly, this is the best we can hope for. Just consider the situation that f is addition and that the value of $f(a_1, \ldots, a_t) = t$. Then, obviously each participant must have input 1. To have another example, let us consider

$$f(x_1, x_2, x_3) = \begin{cases} 1 & , \quad \text{if none } x_i \text{ is prime,} \\ \min\{x_1, x_2, x_3\}, & \text{otherwise.} \end{cases}$$

Now, suppose that $a_2 = 19$ and that $f(a_1, a_2, a_3) = 17$. Then A_2 can conclude that one of the values a_1 or a_3 must have been 17. But if $a_2 = 4$ and $f(a_1, a_2, a_3) = 1$, then A_2 does not know anything about the values a_1 and a_2. Hopefully, this example clearly illustrated what is meant by partial disclosure of a secret.

The trivial solution to this problem would be to use a neutral referee every participant is trusting in. Then everybody sends his a_i to the referee who in turn sends $f(a_1, \ldots, a_t)$ back to all participants. Afterwards the referee is deleting all a_i, $i = 1, \ldots, t$.

Looking at practical needs, we see that such a trusted referee would be seldom available.

So, in the literature, we find a variety of proposals for protocols solving this problem or variations thereof. In general, it is, however, very difficult to prove useful assertions concerning the security of such protocols. The difficult point here is that some participants may conspirate for cheating the remaining partners. On the other hand, the range of potential applications of such protocols is very large, e.g., secret voting. If we look at secret voting then it also possible to include a veto right for some participants. If the voting is negative, then nobody knows whether or not it was a majority vote or if somebody used his veto.

For having another example, we again may look at Alice and Bob. They want to figure out who has more money on the personal bank account without disclosing the actual account balance. In this example, we may also assume that both balances are upper bounded by some natural number c (also known to both).

Next, we shall present a protocol satisfying the following conditions. Before executing the protocol, Alice is exclusively knowing her balance i and Bob is exclusively knowing his balance j.

After execution of the protocol both know if $i \geqslant j$ or $j > i$, but they do not know nothing more about i and j. We choose any fixed public-key cryptosystem (given by E_A, E_B, D_A, D_B as usual) as well as a bound S. Then, the protocol works as follows.

Protocol *Compare*

Step 1: Bob chooses randomly a large number $x \geqslant S$ and computes $k = E_A(x)$ as well as $k - j$.

Step 2: Bob sends $k - j$ to Alice.

Step 3: Alice computes for herself $y_u = D_A(k - j + u)$ for $1 \leqslant u \leqslant S$.

Then Alice chooses randomly a big prime p. (The approximate size of p is somewhat smaller than than the size of x. So, Alice and Bob have to agree on the approximate sizes of x and p in advance). Alice computes for herself $z_u = y_u \bmod p$ for $1 \leqslant u \leqslant S$.

Next, Alice verifies that for all u and $v \neq u$ the condition

$$|z_u - z_v| \geqslant 2 \quad \text{and} \quad 0 < z_u < p - 1 \tag{13.1}$$

is satisfied.

If this condition is not satisfied, then Alice chooses randomly a new prime p and iterates the construction.

Step 4: Alice sends Bob the sequence $z_1, z_2, \ldots, z_i, z_{i+1} + 1, z_{i+2} + 1, \ldots, z_c + 1$ and p (in this order).

Step 5: Bob checks if the jth number in the sequence $\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_c, \hat{z}_{c+1}$ received is satisfying

$$\hat{z}_j \equiv x \bmod p \ .$$

If it is, Bob concludes "$i \geqslant j$."

Otherwise, Bob concludes "$i < j$."

Step 6: Bob sends his conclusion to Alice.

We continue by checking the correctness of the protocol given.

Lemma 13.2. *Protocol* ***Compare*** *is correct.*

Proof. We distinguish the following cases.

Case 1: $i \geqslant j$

Then the jth number in the sequence $\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_c, \hat{z}_{c+1}$ received satisfies $\hat{z}_j = z_j$. By construction we therefore obtain

$$\hat{z}_j \equiv y_j \bmod p \ .$$

Now, it suffices to show that $y_j \equiv x \bmod p$.

By construction we have

$$\begin{aligned} k &= E_A(x) \text{ (cf. Step 1) and} \\ y_j &= D_A(k - j + j) \text{ (cf. Step 3) and thus} \\ y_j &= D_A(k) = D_A(E_A(x)) = x \ . \end{aligned}$$

Hence, we can conclude $y_j \equiv x \bmod p$.

Case 2: $i < j$

Then, by construction, we have $\hat{z}_j = z_j + 1$. Consequently,

$$\hat{z}_j \equiv z_j + 1 \not\equiv z_j \equiv y_j = x \bmod p$$

and thus $\hat{z}_j \not\equiv x \bmod p$.

This shows that the inequality concluded by Bob does imply the checked congruence as necessary condition.

On the other hand, the validity of the checked congruence also implies the inequality concluded by Bob provided all numbers $\hat{z}_u$ are in the range between 0 and $p - 1$ and no number appears twice.. The latter two conditions are satisfied by the test (13.1) performed by Alice. ∎

Some more remarks are in order here. First, Step 3 has been designed in the complex way described above to ensure that Bob can *only* conclude the inequality. For seeing this, suppose Alice would send the sequence

$$y_1, y_2, \ldots, y_i, y_{i+1} + 1, y_{i+2} + 1, \ldots, y_c + 1 \ .$$

Then Bob could compute

$$E_A(y_1), E_A(y_2), \ldots, E_A(y_i), E_A(y_{i+1} + 1), E_A(y_{i+2} + 1), \ldots, E_A(y_c + 1) \ .$$

Additionally, he can compute $k-j+1, k-j+2, \ldots, k-j+c$. This information suffices to compute i. You are advised to show this as an exercise.

Second, the only information exchanged between Bob and Alice occurs in Step 2 (Bob to Alice) and Step 4 (Alice to Bob). We have no idea how this information can be used to determine i and j by Bob and Alice, respectively. We also have no idea how this information can be used to derive more than $i > j$ or $i \leqslant j$. However, we do not have a proof for it.

Third, some form of misuse is always possible. Suppose Bob is only interested in learning whether or not Alice possesses more than \$10 000. Then Bob could set $j = 10000$ and not to his true balance. If he applies this trick by also using binary search (possibly by changing his identity as often as necessary), then he could even figure out what Alice's balance really is.

This brings us back to the problem of digital signature which we shall study in some more detail in the next lecture.

Next, we look at the problem how to share a secret.

13.3. Threshold Schemes

"Three may keep a secret, if two of them are dead."

Benjamin Franklin

The problem we want to consider goes back to Liu [3] who stated it as follows.

> Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of keys to the locks each scientists must carry?

Shamir [4] showed that the smallest solution comprises 462 locks at all and 252 keys per scientist. These numbers look large and thus we want to explain how Shamir [4] arrived at them. Let us assume that all keys and all locks have numbers printed on them. The locks are numbered by pairwise different numbers, and without loss of generality we shall assume that they are numbered $\ell = 1, \ldots, n$.

A key with number ℓ can open the lock with number $\mathfrak{m}$ if and only if $\ell = \mathfrak{m}$. For the sake of presentation, we first look at a smaller number of scientists, i.e., we consider the case of 4 persons A, B, C, D. Moreover, we vary the condition of how many persons are needed to open the cabinet from 1 through 4. The figures below display the resulting $(\mathsf{k}, 4)$ threshold schemes for $\mathsf{k} = 1, 2, 3, 4$.

	key1
A	◯
B	◯
C	◯
D	◯

Figure 13.1: $(1, 4)$ threshold scheme

	key1	key2	key3	key4
A	◯	◯	◯	
B	◯	◯		◯
C	◯		◯	◯
D		◯	◯	◯

Figure 13.2: $(2, 4)$ threshold scheme

	key1	key2	key3	key4	key5	key6
A	◯	◯	◯			
B	◯			◯	◯	
C		◯		◯		◯
D			◯		◯	◯

Figure 13.3: $(3, 4)$ threshold scheme

	key1	key2	key3	key4
A	◯			
B		◯		
C			◯	
D				◯

Figure 13.3: $(4, 4)$ threshold scheme

The idea is easily explained. The number of rows is always 4, i.e., equal to the number of persons. The number of columns equals the number of locks needed. We display who is getting a key for the lock in the corresponding column (marked by a circle).

First, we ask for the smallest number of locks needed. The answer is provided by the following claim which we present in full generality.

Claim 1. *The smallest number of locks needed is* $\binom{n}{k-1}$ *and the smallest number of keys needed is* $\binom{n-1}{k-1}$.

This can be seen as follows. We have to ensure that $k-1$ or fewer persons cannot open the cabinet. That means, for any choice of $k-1$ persons out of the n ones, there must be a lock for which these $k-1$ persons do *not* have a key. There are $\binom{n}{k-1}$ many possibilities to choose $k-1$ persons out of n which gives the lower bound for the number of locks.

Next, we show this number of locks to be sufficient, too. Obviously, at least one key from every lock has to be given to some person, since otherwise this lock cannot be opened at all. Moreover, by symmetry, it is easy to see that everybody must receive the same number of locks.

Next, let us take any k persons. We have to ensure that they have at least one key for every lock. If we take any subset

$$T = \{i_1, \ldots, i_{k-1}\} \subseteq \{1, \ldots, n\}$$

then there is a lock L_T for which none of the persons $i \in \{i_1, \ldots, i_{k-1}\}$ does have a key. Hence, in the complement of T there are $n - (k-1)$ many persons. Moreover, if we pick any person from this complement, then she must have a key for the lock L_T, since otherwise there would be a subset of k persons which cannot open the door. Thus, we arrive at the following number of keys

$$\frac{\binom{n}{k-1} \cdot (n-(k-1))}{n} = \binom{n-1}{k-1} .$$

Now, we distribute the keys as follows. We start with lock $\binom{n}{k-1}$ and take the lexicographically first subset of $k-1$ persons. These persons do not get a key for this lock, while everybody in the complement does. Then we take lock $\binom{n}{k-1-1}$ and the lexicographically second subset of $k-1$ persons. Again, everybody in the subset is not getting a key for the lock $\binom{n}{k-1-1}$, while everybody in the complement does. This procedure is repeated until all locks and subsets have been handled.

Finally, if we take any k persons, an easy application of the pigeonhole principle shows that in each column there must be row (marked by one of the k persons chosen) that has a circle in it. This shows the claim, and we are done.

Hence, in the quoted problem, we need $\binom{11}{5} = 462$ many locks, and $\binom{10}{5} = 252$ keys per person for the quoted problem.

Furthermore, it should be mentioned that $\binom{n}{k}$ becomes maximal for $k = n/2$, if n is even, and for $k = (n+1)/2$, if k is odd. Moreover, $\binom{n}{n/2} = O(4^{n/2})$, and thus it grows exponentially in n. Thus, a (k, n) threshold scheme, for being practically applicable, has to give up the intuitive appealing idea of locks and keys. Instead, we shall look for different possibilities to share a secret.

For doing this, let us first give the general definition of a shared secret and of a (k, n) threshold scheme given by Shamir [4]. Assume we have n persons $P_1, \ldots, P_n$ and a secret datum D which we want to divide into n pieces $D_1, D_2, \ldots, D_n$.

Definition 13.1. *We say that n participants k-divide a secret, where $1 < k \leqslant n$ provided the following 3 conditions are satisifed.*

(1) *Each participant P_i possesses an information D_i which is not known to any other participant P_j, $i \neq j$ for $j \in \{1, \ldots, n\}$.*

(2) *The knowledge of any k or more of the $D_1, D_2, \ldots, D_n$ pieces allows us to compute the whole datum D easily,*

(3) *the knowledge of any $k-1$ or fewer $D_1, D_2, \ldots, D_n$ pieces leaves D completely undetermined.*

A set $\{D_1, \ldots, D_n\}$ satisfying (2) and (3) is said to be a (k, n) threshold scheme.

The pieces D_i of information are referred to as a share. The example at the beginning of this chapter provides at least evidence that (k, n) threshold schemes can be realized. However, while the idea of using locks is intuitively appealing we still have to outline how to simulate them by appropriately chosen problems that meet the wanted complexity theoretic requirements in Items (2) and (3). Furthermore, we aim to find a simulation such that the the number of simulated "locks" does no longer grow exponentially.

The following construction is based on Mignotte's threshold sequences. A sequence $m_1 < \cdots < m_n$ of pairwise relatively prime and positive numbers is said to be a (k, n) threshold sequence if

$$m_1 \cdot m_2 \cdot \ldots \cdot m_k > m_n \cdot m_{n-1} \cdot \ldots \cdot m_{n-k+2} \qquad \text{(A)} \ .$$

Now, suppose, we have a (k, n) threshold sequence. We set

$$\begin{aligned} M &= m_1 \cdot m_2 \cdot \ldots \cdot m_k \ , \text{ and} \\ N &= m_n \cdot m_{n-1} \cdot \ldots \cdot m_{n-k+2} \ . \end{aligned}$$

The secret is then any number D satisfying $N \leqslant D \leqslant M$. Now, the pieces for each participant are defined by

$$D_i = D \bmod m_i \ ,$$

that is, P_i obtains D_i and nothing else, $i = 1, \ldots, n$.

Thus, Condition (1) is fulfilled by construction. We have to show that the Conditions (2) and (3) of Definition 13.1 are satisfied, too.

Let $D_{i_1}, \ldots, D_{i_k}$ be any subset of k elements from $\{D_1, \ldots, D_n\}$. By the Chinese remainder theorem, the system

$$x \equiv D_i \bmod m_i \ , \qquad i \in \{i_1, \ldots, i_k\}$$

has a uniquely determined solution $\hat{D}$ modulo $\prod_{j=1}^{k} m_{i_j}$. By the definition of a (k, n) threshold sequence and the choice of D we obtain

$$D \leqslant M = m_1 \cdot m_2 \cdot \ldots \cdot m_k \leqslant \prod_{j=1}^{k} m_{i_j} \ .$$

Since $\hat{D} \equiv D_i \equiv D \bmod m_i$ for all $i \in \{i_1, \ldots, i_k\}$, we see that any k participants of the secret sharing scheme can compute the secret datum D. This proves (2).

We continue with the proof of Condition (3), i.e., we show Condition (3) to be satisfied. Let $\{D_{i_1}, \ldots, D_{i_{k-1}}\}$ be any subset of $k-1$ elements from $\{D_1, \ldots, D_n\}$. Again, we may apply the Chinese remainder theorem, and obtain

$$\hat{D} = e_{i_1} \cdot D_{i_1} + \cdots + e_{i_{k-1}} \cdot D_{i_{k-1}} \bmod \prod_{j=1}^{k-1} m_{i_j} \ . \tag{13.2}$$

Obviously, (13.2) is the congruence containing all information we have. Nevertheless, (13.2) leaves many possibilities for D. Therefore, we continue by estimating the number of possibilities.

The biggest product of $k-1$ numbers chosen from $\{m_1, \ldots, m_n\}$ is N. The smallest product of k numbers chosen from $\{m_1, \ldots, m_n\}$ is M. Since $D \equiv \hat{D} \bmod m_i$ for all $i \in \{i_1, \ldots, i_{k-1}\}$, we also have

$$D \equiv \hat{D} \bmod \prod_{j=1}^{k-1} m_{i_j} \ ,$$

(remember that the numbers m_i, $i \in \{1, \ldots, n\}$ are pairwise relatively prime), i.e., D and $\hat{D}$ differ by a multiple of $\prod_{j=1}^{k-1} m_{i_j}$. Consequently, one can try all

$$D = \hat{D} + \prod_{j=1}^{k-1} m_{i_j}, \qquad D = \hat{D} + 2 \cdot \prod_{j=1}^{k-1} m_{i_j}, \ldots,$$

possibilities. This gives a lower bound of

$$\frac{M - N - 1}{N}$$

many possibilities. Of course, the true value of D can be only determined, if one has an oracle for testing these possibilities. For example, it is well imaginable that one has only 3 trials to test D (like passwords).

Thus, it remains to show that one can always choose (k, n) threshold sequences in a way such that $(M - N - 1)/N$ is large. Let $\pi(x)$ be the number of all primes less than or equal to x. The prime number theorem is telling us that

$$\frac{x}{\ln x} < \pi(x) < \frac{5}{4} \cdot \frac{x}{\ln x} \qquad \text{for all } x \geqslant 114 \ ,$$

i.e., there is constant c such that

$$\pi(x) \leqslant c \cdot \frac{x}{\log x} \ .$$

Now, let $\pi(n, \alpha)$ be the number of all primes in the interval (p_n^α, p_n), where p_n is the n-th prime number and $\alpha \in (0, 1)$. Then, we can show the following lemma.

Lemma 13.3. *Let $n \in \mathbb{N}$ with $n \geqslant 2$. For every k with $2 \leqslant k \leqslant n$ there are arbitrarily big numbers y such that*

$$\pi\left(y, \frac{k^2-1}{k^2}\right) > n \ .$$

Before proving the lemma, we show how to get the desired result from it. We choose y such that $\pi(y, \frac{k^2-1}{k^2}) > n$. That is, in the interval

$$(p_y^{(k^2-1)/k^2}, p_y]$$

there are at least n prime numbers. Let $m_1, \ldots, m_n$ be the first n primes in this interval.

Claim. $m_1, \ldots, m_n$ *form a* (k, n) *threshold sequence.*

The condition $m_1 < m_2 < \cdots < m_n$ is obvious. Moreover,

$$\begin{aligned} M = \prod_{i=1}^{k} m_i \ &> \ m_1^k \ \geqslant \ \left(p_y^{\frac{k^2-1}{k^2}}\right)^k \\ &= \ p_y^{\frac{(k+1)(k-1)}{k}} \ > \ p_y^{k-1} \\ &\geqslant \ m_n \cdot m_{n-1} \cdot \ldots \cdot m_{n-k+2} = N \ , \end{aligned}$$

where the last inequality holds, since $m_n \leqslant p_y$ and thus $m_n^{k-1} \leqslant p_y^{k-1}$. This proves the claim.

Finally, we obtain

$$\frac{M-N}{N} \ \geqslant \ \frac{p_y^{\frac{k^2-1}{k}} - p_y^{k-1}}{p_y^{k-1}} \ = \ p_y^{\frac{k-1}{k}} - 1 \ .$$

Consequently, one can start with a lower bound B for $(M - N - 1)/N$. Then one searches for p_z such that

$$p_z^{\frac{k-1}{k}} - 1 \geqslant B \ .$$

By the lemma above, then there exists a $y \geqslant z$ such that one can form the wanted (k, n)-threshold sequence from $m_1, \ldots, m_n$.

Finally, we have to show the lemma above. Let k, n be arbitrarily fixed, and let $\alpha \in (0, 1)$. By the prime number theorem we know that $p_m = O(m \log m)$. Hence, $p_m^\alpha = O(m^\alpha (\log m)^\alpha)$. Furthermore,

$$\pi(m, \alpha) = \pi(p_m) - \pi(p_m^\alpha) \ .$$

We choose c_1 such that $\pi(p_m) \geqslant c_1 m$ and c_2 such that

$$\begin{aligned}\pi(p_m^\alpha) &\leqslant c_2 \cdot \frac{m^\alpha (\log m)^\alpha}{\log(m^\alpha (\log m)^\alpha)} \leqslant c_2 \cdot \frac{m^\alpha (\log m)^\alpha}{\log m^\alpha} \\ &= c_2 \cdot \frac{m^\alpha (\log m)^\alpha}{\alpha \cdot \log m} \ .\end{aligned}$$

Thus, we obtain

$$\begin{aligned}\pi(m,\alpha) &= \pi(p_m) - \pi(p_m^\alpha) \geqslant c_1 \cdot \frac{m \log m}{\log m} - c_2 \cdot \frac{m^\alpha (\log m)^\alpha}{\alpha \cdot \log m} \\ &\geqslant c\left(\frac{m \log m}{\log m} - \frac{m^\alpha (\log m)^\alpha}{\alpha \cdot \log m}\right) \\ &= \frac{c}{\log m}\left(m \log m - \frac{m^\alpha (\log m)^\alpha}{\alpha}\right) \\ &= \frac{c \cdot m \log m}{\log m}\left(1 - \frac{1}{\alpha \cdot m^{1-\alpha}(\log m)^{1-\alpha}}\right) \\ &= c \cdot m \underbrace{\left(1 - \frac{1}{\alpha \cdot m^{1-\alpha}(\log m)^{1-\alpha}}\right)}_{=:X} \ .\end{aligned}$$

The expression X converges to 1 as m tends to infinity. Consequently, for all sufficiently large values of m we see that $\pi(m,\alpha) > n$. Thus, setting $\alpha = (k^2-1)/k^2$, the lemma follows. ∎

References

[1] E.R. BERLEKAMP (1970), Factoring polynomials over large finite fields, *Mathematics of Computations* **24**, 713 – 735.

[2] M. BLUM (1981), Coin flipping per telephone: A protocol for solving problems impossible. SIGACT News **15**, 23–27.

[3] C.L. LIU (1968), *Introduction to Combinatorial Mathematics.*, McGrawHill, New York.

[4] A. SHAMIR (1979), How to Share a Secret, *Communications of the ACM* **22**, pp. 612 – 613.

Lecture 14: Digital Signatures

In this lecture, we want to deal in some more detail with the problem of digital signatures. As usual, we consider two parties A and B with possibly conflicting interests. Typically, the parties could be a bank and its customer, any two parties wishing to do business over the internet, diplomats from countries with different interests, and so on. If you are doing business on the internet you require security and trust, since you cannot see the person you are dealing with, you cannot see any document proving the partner's identity and you cannot even know if the web site you are connected to belongs to the society it says. To answer these juridical demands, the European Union adopted a community framework for electronic signatures some time ago (directive 1999/93/EC of the European Parliament and the council of December 13, 1999, on a community framework for electronic signatures) that has been implemented in various European countries. The European directive is used for business in which European partners (persons or societies) or public administrations are involved. It also means that if a Japanese or an American organization enters into an electronic contract with a European society it has to respect European requirements to ensure the contract is valid.

Before discussing further details, let us also stress the important point that there is another fundamental difference between conventional and digital signatures besides that you see the partner signing a document (or let somebody else see it for you, i.e., a notary). If you copy a conventionally signed document, then there are usually ways for distinguishing the copied document and the original one. But a copy of a signed digital document is *identical* to the original one. So, if A sends a message to B authorizing B to withdraw 1000 € form A's bank account then the intention is usually that B is doing is once and not all the time B feels the need of getting 1000 €. Since the identity of the digital copy and the digital original cannot be prevented, the *message itself* should contain the necessary information such as a date, the clear statement *once*, and so on.

Nowadays, a digital signature is usually based on public-key cryptographic systems. European law distinguishes between an electronic signature (also called *weak digital signature*) which is used for *authentication*. That is, such a signature should prove that the person who sent the text is the electronic signature's holder. However, you cannot be sure that the person who sent the message is also the *key owner*. The key owner does not only have the means to sign a message appropriately but has also the *explicit right* to use it. For seeing the difference, we look at a typical example. Usually a key holder would be a server that creates signatures on, for example, a company's software. The company or employee would be the key owner. So, someone in the company could hack the server and sign something contentious using the company's authority. Furthermore, an electronic signature does not guarantee the *integrity* of the message signed. That is, a third party may have altered the text sent without having changed the signature. Of course, this is usually not what we want. We also

want to be sure that the text received is the same that was sent, and that no hacker had changed it.

To summarize, authentication guarantees that the message received, say from A, has been really sent by A. It should be at least very difficult if not impossible for a third party C to pretend to be A. *Integrity* guarantees that the message received is the same as the message sent. So, no third party and also not the legal recipient should be able to forge a message and to pretend to have received it in properly signed form from A.

Putting these requirements together leads to an *advanced electronic signature*. In terms of law an advanced electronic signature must fulfill the following requirements.

(1) it is uniquely linked to the signatory;

(2) it is capable of identifying the signatory;

(3) it is created using means that the signatory can maintain under his sole control; and

(4) it is linked to the data to which it relates that any subsequent change of the data is detectable.

In some sense, these requirements are contradictory. For verifying that the message received is from A, as claimed, B should know at least something about A's signature. For B not being able to manipulate a signed message received from A (or for a third party C aiming the same), neither B nor any third party should know too much about A's signature.

14.1. Realizing Advanced Digital Signatures

So, let us first see how these requirements can be fulfilled simultaneously, at least in principle, when using a public-key cryptosystem. As before, we denote by E_A, E_B, ..., and D_A, D_B, ..., the encryption and decryption algorithms (keys) used by the parties A, B, ... Then the following ***Protocol DS*** can be used. Let us assume that A sends a message to B.

Protocol DS

Step 1: First, A applies to message w she wants to send her decryption algorithm D_A obtaining $\hat{w} = D_A(w)$. Then she computes

$$c = E_B(\hat{w})$$

and sends c to B.

Step 2: First, B applies D_B to the message c received, i.e., B computes $\hat{c} = D_B(c)$. Then B computes

$$w = E_A(\hat{c}) \ .$$

Observe that the protocol is correct, since by associativity we have

$$\mathsf{E}_A(\mathsf{D}_B(\mathsf{E}_B(\mathsf{D}_A(w)))) = \mathsf{E}_A(\mathsf{D}_A(w)) = w \ .$$

Furthermore, taking into account that *only* A knows D_A neither B nor a third party C can forge A's signature. This is that case at least if plaintexts are meaningful messages written in some natural language. Then the probability is negligible that some text not obtained from by D_A from a meaningful plaintext would translate into something meaningful. Also, A cannot deny having sent the signed message to B, since A is the only one knowing D_A.

There is also another advantage of the method described above. If the underlying public-key cryptosystem is indeed satisfactory, then the application of D_A changes the whole text and not only the name of the sender A. Thus, even if many messages are exchanged it seems hard to get some knowledge concerning A's signature. For a better understanding of this point you should do the following exercise.

Exercise 39. *Discuss the pros and cons of the* ***Protocol DS*** *provided above to the idea of sending* $(\text{name}, \mathsf{D}_A(\text{name}))$ *as signature.*

Moreover, let us ask why A first applies D_A and then E_B. She could also first apply E_B and then D_A. This would require that B is also changing the order of applications, i.e., first E_A and then D_B. Consequently, the protocol would be still correct. Does this mean that we have two possibilities for designing our advanced digital signature scheme?

For seeing the difference let us assume that C is an eavesdropper. So, C catches the message from A and makes sure that it is not directly delivered to B. If we use the second version, then C may itself apply E_A and has now $\mathsf{E}_B(w)$. This gives C the possibility to sign the message with its own name by applying D_C to it. If C transmits $\mathsf{D}_C(\mathsf{E}_B(w))$ to B then B would verify to have received the message from C instead of having received it from A. Thus, though the original plaintext remains unchanged the identity of the sender (that is A) is gone. Because of this potential difficulty, our ***Protocol DS*** was designed in a way that sending happened before encryption.

Our ***Protocol DS*** has also the advantage that only the legal recipient can read it provided D_B is kept secretly. This property is usually referred to as *confidentiality*.

But still, we have a problem. The ***Protocol DS*** does not take care of two issues that are very important. A can still deny to have sent the message and B can deny to have received it. In terms of law these two issues are summarized by the term *non-repudiation*. A digital signature satisfying authentication, integrity, confidentiality and non-repudiation is usually called *strong digital signature* or *undeniable digital signature*.

In order to achieve this goal, one has to combine the ***Protocol DS*** with a *challenge response protocol* as described in Lecture 12.

14.2. An Undeniable Digital Signature Scheme

We describe here an undeniable digital signature scheme that was introduced by Chaum and van Antwerpen in 1989. It consists of three components: a *signing* algorithm *sig*, a *verification protocol* and a *disavowal protocol*. Again we assume that A sends a message to B. The new point is that A's cooperation is required to verify a signature made by the signer A. This protects A against the possibility that that documents signed by her are duplicated and distributed electronically without her approval. But what prevents A from disavowing a signature made by her at an earlier time? A might claim that a valid signature is a forgery, and either refuse to verify it, or carry out the verification in a way such that the valid signature will not be verified. That is the point where the disavowal protocol comes into play. Using this disavowal protocol, A can prove that a signature not made by her is indeed a forgery. Now, if A refuses to take part in this disavowal protocol, court will take this as evidence that the signature given has been made by A.

Next, we present the formal protocol.

Protocol CvA

Let $p = 2q + 1$ be a prime such that q is prime and the discrete log problem in $\mathbb{Z}_p$ is intractible. Let $\alpha \in \mathbb{Z}_p^*$ be an element of order q. Let $1 \leqslant a \leqslant q - 1$ and define $\beta = \alpha^a \bmod p$. Furthermore, by G we denote the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q. Note that G consists of the quadratic residues modulo p. The values p, α and β are *public* and a is kept *secretly* by A.

The plaintext messages x are assumed to be elements of G and so are the ciphers (as we shall see in a moment).

A signs the plaintext message x by computing

$$y = sig(x) = x^a \bmod p$$

Then she sends (y, x) to B.

The verification (for $x, y \in G$) is done by executing the following protocol.

Step 1: B chooses randomly $e_1, e_2 \in \mathbb{Z}_q^*$.

Step 2: B computes (the challenge) $c = y^{e_1}\beta^{e_2} \bmod p$ and sends it to A.

Step 3: A computes the modular inverse a^{-1} of a modulo q and then $d = c^{a^{-1}} \bmod p$ and sends it to B.

Step 4: B accepts y as a valid signature if and only if

$$d \equiv x^{e_1}\alpha^{e_2} \bmod p \ .$$

`end`

We should explain the roles of p and q in this scheme. The scheme lives in $\mathbb{Z}_p$ but we need to be able to perform computations in a multiplicative subgroup G of $\mathbb{Z}_p^*$ of prime order. In particular, we need to be able to compute inverses modulo $|G|$. This is the reason why $|G|$ should be prime. It is convenient to take $p = 2q + 1$ where q is prime. In this way, the subgroup is as large as possible. This is desirable, since plaintexts and ciphers are both elements of G.

We first prove that B will accept a valid signature. In the following computations, all exponents are assumed to be reduced modulo p.

First, observe that

$$d \equiv c^{a^{-1}} \equiv y^{e_1 a^{-1}} \beta^{e_2 a^{-1}} \bmod p \ .$$

Since $\beta \equiv \alpha^a \bmod p$ we have

$$\beta^{a^{-1}} \equiv \alpha \bmod p \ .$$

Similarly, $y = x^a \bmod p$ implies that $y^{a^{-1}} \equiv x \bmod p$. Hence,

$$d \equiv x^{e_1} \alpha^{e_2} \bmod p$$

as desired.

Example 14.1. We take $p = 467$. Thus, $q = (467 - 1)/2 = 233$. Then 2 is a generator (a primitive root) of $\mathbb{Z}_p^*$. We can conclude that $2^2 = 4$ is a generator of G, the quadratic residues modulo p. Thus, we take $\alpha = 4$. Let $a = 101$ be A's secret number. Then

$$\beta = 4^{101} \equiv 449 \bmod 467 \ .$$

A wishes to sign the message $x = 119$. Thus she computes

$$y = 119^{101} \equiv 129 \bmod 467 \ .$$

Next, suppose B wants to verify the signature y. Suppose, B has chosen at random $e_1 = 38$ and $e_2 = 397$. Then B computes

$$c = 129^{38} \cdot 449^{397} \equiv 13 \bmod 467 \ .$$

A in turn first computes the modular inverse a^{-1} of 101 modulo 233 which is 30. Then she calculates

$$d = c^{a^{-1}} \bmod 467 \equiv 13^{30} \equiv 9 \bmod 467 \ .$$

Finally, B checks the response by verifying that

$$119^{38} \cdot 4^{397} \equiv 9 \bmod 467 \ .$$

Hence, B accepts A's signature as valid.

end Example

Next, we prove that A cannot fool B into accepting a fraudulent signature as valid, except with a very small probability.

Theorem 14.1. *If* $y \not\equiv x^a \bmod p$*, then* B *will accept* y *as a valid signature for* x *with probability* $1/q$.

Proof. First we observe that each possible challenge c corresponds to exactly q ordered pairs (e_1, e_2). This is because y and β are both elements of the multiplicative group G of prime order q.

Now, when A receives the challenge c she has no way of knowing which of the q possible pairs (e_1, e_2) B has been used to construct c. We claim that, if $y \not\equiv x^a \bmod p$, then any possible response $d \in G$ that A might make is consistent with exactly one of the q possible ordered pairs (e_1, e_2).

Since α generates G, we can write any element g of G as a power of α, say $g = \alpha^z$ where the exponent z is determined uniquely modulo q. So, we can write

$$c = \alpha^i \quad d = \alpha^j \quad x = \alpha^k \quad \text{and} \quad y = \alpha^\ell \ ,$$

where $i, j, k, \ell \in \mathbb{Z}_q$ and all arithmetic is done modulo q. Consider the following two congruences:

$$\begin{aligned} c &\equiv y^{e_1}\beta^{e_2} \bmod p \\ d &\equiv x^{e_1}\alpha^{e_2} \bmod p \end{aligned}$$

This system is equivalent to the following system:

$$\begin{aligned} i &\equiv \ell e_1 + a e_2 \bmod q \\ j &\equiv k e_1 + e_2 \bmod q \ . \end{aligned}$$

Now, we are assuming that

$$y \not\equiv x^a \bmod p \ ,$$

so it follows that

$$\ell \not\equiv ak \bmod q \ .$$

Hence, the coefficient matrix of this system of congruences modulo q has non-zero determinant. Therefore, there is a unique solution to the system. That is, every $d \in G$ is the correct response for exactly for exactly one of the q possible ordered pairs (e_1, e_2). Consequently, the probability that A gives B a response d that will be verified is is exactly $1/q$, and the theorem is shown. ∎

Finally, we turn our attention to the disavowal protocol. This protocol consists of two runs of the verification protocol (see below).

The Disavowal Protocol

Step 1: B chooses $e_1, e_2 \in \mathbb{Z}_q^*$ at random.

Step 2: B computes $c = y^{e_1}\beta^{e_2} \bmod p$ and sends it to A.

Step 3: A computes a^{-1} modulo q and then $d = c^{a^{-1}} \bmod p$ and sends it to B.

Step 4: B verifies that $d \equiv x^{e_1}\alpha^{e_2} \bmod p$.

Step 5: B chooses $f_1, f_2 \in \mathbb{Z}_q^*$ at random.

Step 6: B computes $C = y^{f_1}\beta^{f_2} \bmod p$ and sends it to A.

Step 7: A computes a^{-1} modulo q and then $D = C^{a^{-1}} \bmod p$ and sends it to B.

Step 8: B verifies that $D \equiv x^{f_1}\alpha^{f_2} \bmod p$.

Step 9: B concludes that y is a forgery if and only if

$$(d\alpha^{-e_2})^{f_1} \equiv \left(D\alpha^{-f_2}\right)^{e_1} \bmod p \ .$$

Example 14.2. As before, we take $p = 467$, $q = 233$, $\alpha = 4$, $a = 101$ and $\beta = 449$.

Suppose the message $x = 286$ is signed with the bogus signature $y = 83$, and that A wants to convince B that the signature is invalid.

Furthermore, suppose B begins choosing at random values $e_1 = 45$ and $e_2 = 237$. B then computes

$$c = y^{e_1}\beta^{e_2} \equiv 83^{45}449^{237} \equiv 305 \bmod 467 \ ,$$

and A responds with

$$d = c^{a^{-1}} \equiv 305^{30} \equiv 109 \bmod 467 \ .$$

Then B computes

$$286^{45}4^{237} \equiv 149 \bmod 467 \ .$$

Since $149 \neq 109$, B proceeds to Step 5 of the protocol. Now, suppose B chooses at random $f_1 = 125$ and $f_2 = 9$. Then B computes

$$C = 83^{125}449^{9} \equiv 270 \bmod 467 \ ,$$

and A responds with

$$D = C^{a^{-1}} \equiv 270^{30} \equiv 68 \bmod 467 \ .$$

Now B verifies that

$$68 \not\equiv 286^{125}4^{9} \equiv 25 \bmod 467 \ .$$

Thus, B performs the consistency check in Step 9 and obtains

$$\left(109 \cdot 4^{-237}\right)^{125} \equiv 188 \equiv \left(68 \cdot 4^{-9}\right)^{45} \bmod 467 \ .$$

Thus, the consistency check succeeds and B is convinced that the signature is *not* valid.

end Example

Steps 1 through 4 and Steps 5 through 8 comprise two unsuccessful runs of the verification protocol. Step 9 is a "consistency check" that enables B to determine if A is forming her responses in the manner specified in the protocol.

We have to show two things at this point.

(1) A can convince B that an invalid signature is a forgery.

(2) A cannot make B believe that a valid signature is a forgery except with a very small probability.

First, we show the following.

Theorem 14.2. *If* $y \not\equiv x^{a} \bmod p$, *and* B *and* A *follow the disavowal protocol, then*

$$(d\alpha^{-e_2})^{f_1} \equiv \left(D\alpha^{-f_2}\right)^{e_1} \bmod p \ .$$

Proof. Using the facts that

$$\begin{aligned} d &\equiv c^{a^{-1}} \bmod p \\ c &\equiv y^{e_1}\beta^{e_2} \bmod p \quad \text{and} \\ \beta &\equiv \alpha^{a} \bmod p \ , \end{aligned}$$

we have that

$$\begin{aligned} (d\alpha^{-e_2})^{f_1} &\equiv \left((y^{e_1}\beta^{e_2})^{a^{-1}}\alpha^{-e_2}\right)^{f_1} \bmod p \\ &\equiv y^{e_1f_1}\beta^{e_2a^{-1}f_1}\alpha^{-e_2f_1} \bmod p \\ &\equiv y^{e_1f_1}\alpha^{e_2f_1}\alpha^{-e_2f_1} \bmod p \\ &\equiv y^{e_1f_1} \bmod p \ . \end{aligned}$$

Using the facts that $D \equiv C^{a^{-1}} \bmod p$, $C \equiv y^{f_1}\beta^{f_2} \bmod p$ and $\beta \equiv \alpha^{a} \bmod p$, a similar computation establishes that

$$\left(D\alpha^{-f_2}\right)^{e_1} \equiv y^{e_1f_1} \bmod p \ ,$$

and therefore the consistency check in Step 9 succeeds. ∎

Now we look at the possibility that A might attempt to disavow a valid signature. In this situation we do *not* assume that A follows the protocol. That is, A might not construct d and D as specified by the protocol. Hence, in the following theorem, we only assume that A is able to produce values d and D which satisfy the conditions in Steps 4, 8, and 9 of the ***Disavowal Protocol*** presented above.

Theorem 14.3. *Suppose* $y \equiv x^a \bmod p$ *and* B *follows the* ***Disavowal Protocol****. If*

$$d \not\equiv x^{e_1}\alpha^{e_2} \bmod p$$

and

$$D \not\equiv x^{f_1}\alpha^{f_2} \bmod p$$

then the probability that

$$(d\alpha^{-e_2})^{f_1} \not\equiv \left(D\alpha^{-f_2}\right)^{e_1} \bmod p \ .$$

is $1 - 1/q$.

Proof. The proof is done indirectly. Suppose the following is satisfied

$$\begin{aligned} y &\equiv x^a \bmod p \\ d &\not\equiv x^{e_1}\alpha^{e_2} \bmod p \\ D &\not\equiv x^{f_1}\alpha^{f_2} \bmod p \\ (d\alpha^{-e_2})^{f_1} &\equiv \left(D\alpha^{-f_2}\right)^{e_1} \bmod p \ . \end{aligned}$$

We shall derive a contradiction as follows. The consistency check (cf. Step 9) can be rewritten in the following form.

$$D \equiv d_0^{f_1}\alpha^{f_2} \bmod p \ ,$$

where

$$d_0 = d^{1/e_1}\alpha^{-e_2/e_1} \bmod p$$

is a value that depends only on steps 1 through 4 of the ***Disavowal protocol***.

Applying Theorem 14.1, we conclude that y is a valid signature for d_0 with probability $1 - 1/q$. But we are assuming that y is a valid signature for x. That is, with high probability we have

$$x^a \equiv d_0^a \bmod p$$

which implies that $x = d_0$.

However, the fact that

$$d \not\equiv x^{e_1}\alpha^{e_2} \bmod p$$

means that that

$$x \not\equiv d^{1/e_1}\alpha^{-e_2/e_1} \bmod p \ .$$

Since

$$d_0 \equiv d^{1/e_1}\alpha^{-e_2/e_1} \bmod p$$

we conclude that $x \neq d_0$ and we have a contradiction. ∎

Appendix for Complexity

In this appendix, we provide a bit more material for further reading as well as some proofs that had to be omitted due to the lack of time.

15.1. A Lower Bound for the Complexity of Accepting Palindromes

We introduce a technique that may be used to prove lower bounds. For doing this, let us define ***traces*** of Turing computations (sometimes also called crossing sequences). We consider the computation process of a Turing machine M that has received the string $w = x_1 \cdots x_n$, $x_i \in \Sigma$, as input and which is started in the initial configuration described above. Let us fix a border on the tape at position j (cf. Figure 15.1 for the case $j = 2$).

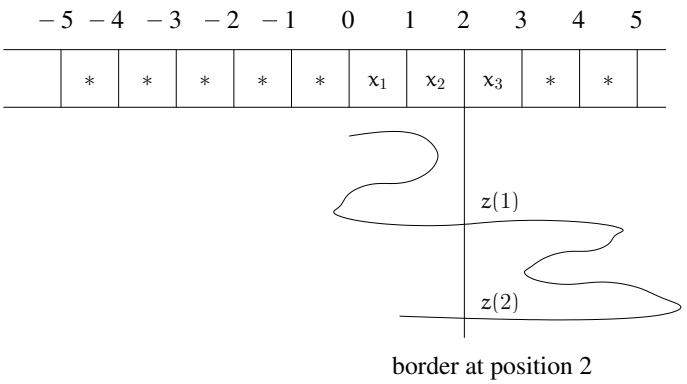

Figure 15.1: A fragment of the trace at border 2.

The trace of M on input w at position j is a word $\mathfrak{z}$ over the set of state symbols defined as follows.

(0) If the read-write head is never crossing the border j, then $\mathfrak{z} = \lambda$.

(1) Assume the read-write head is crossing the border j exactly s times. The first time the border is crossed M is in state $z(1)$, the second time in in state $z(2)$, ..., and the sth time in state $z(s)$. Then, we set $\mathfrak{z} = z(1)z(2)\cdots z(s)$.

We denote the trace of M at position j on input w by $\mathrm{TR}_M(w, j)$. Of course, if $j > 0$, then the first time the head crosses the border at position j is a move from left to right, the second time from right to left, and so on.

If $j \leqslant 0$, then the first time the head crosses the border at position j is a move from right to left, the second time from left to right, and so on.

If M's computation on input w does not stop, then it could also happen that the trace at some position j becomes infinite. But we are going to consider terminating computations only. Thus, all traces will have finite length which is denoted by $|\mathrm{TR}_M(w,j)|$.

The importance of traces is expressed by our first observation, where $\mathbb{Z}$ denotes the set of all integers.

Observation 1. $\mathrm{T}_M(w) \geqslant \sum_{j\in\mathbb{Z}} |\mathrm{TR}_M(w,j)|$.

The observation clearly holds, since every time the head crosses the border, a step has to be performed. Therefore, if we can prove that for every machine accepting a language L there must be infinitely many strings such that at sufficiently many positions sufficiently long traces occur, then we can obtain a lower bound for the time needed to accept L.

So, how can we prove this? Suppose that you know the trace $z(1)\cdots z(s)$ of a Turing machine M on input w at position j. For the sake of presentation, let us assume $j > 0$. Now, cut the tape at position j and remove the right part of the tape cut. Then you can still simulate the whole computation of M on the left part of the tape. As long as M did not cross the border j, let the machine work as before. If it is crossing the border for the first time (in state $z(1)$), stop the machine, put the head back on the left part of tape on the rightmost position, and switch M's state to $z(2)$. Now, it will work again on the left part of the tape. Even more importantly, it will behave precisely the same way as if we have not cut the tape. This is true, since we restarted M is state $z(2)$ and M has no other possibility than its state to memorize something it has done on the right part of the tape that may be needed while working on the left part of it.

Moreover, let M be any Turing machine, let $w = x_1\cdots x_m$ and $\tilde{w} = \tilde{x}_1\cdots\tilde{x}_n$ be any strings, and let $i,j \in \mathbb{N}$ with $0 < i < m$, $0 < j < n$ be such that $\mathrm{TR}_M(w,i) = \mathrm{TR}_M(\tilde{w},j)$. Furthermore, let $\tilde{w}_0^j = \tilde{x}_1\cdots\tilde{x}_j$ and $\tilde{w}_j^n = \tilde{x}_{j+1}\cdots\tilde{x}_n$ as well as $w_0^i = x_1\cdots x_i$ and $w_i^m = x_{i+1}\cdots x_m$. Then

$$\mathrm{TR}_M(\tilde{w},j) = \mathrm{TR}_M(w,i) = \mathrm{TR}_M(\tilde{w}_0^j w_i^m, j) = \mathrm{TR}_M(w_0^i \tilde{w}_j^n, i)\ .$$

Furthermore, consider M's behavior when started on input $\tilde{w}_0^j w_i^m$. We distinguish the following cases.

Case 1. $|\mathrm{TR}_M(\tilde{w},j)|$ is even.

Then the read-write head is left from position j when M terminates its computation. Moreover, the head observes in its final position a $|$ if $\tilde{w} \in L(M)$ and a $*$ provided $\tilde{w} \notin L(M)$.

Case 1. $|\mathrm{TR}_M(\tilde{w},j)|$ is odd.

Then M stops its computation right from position j. Moreover, the head observes in its final position a $|$ if $w \in L(M)$ and a $*$ provided $w \notin L(M)$.

Now, the scenario that $\tilde{w}$ and w are equivalent with respect to acceptance by M is of particular importance. Here, by equivalence with respect to acceptance by M we mean that either both $\tilde{w}, w \in L(M)$ or both $\tilde{w}, w \notin L(M)$.

Taking the latter observation into account, we directly get the following ***Replacement Lemma***. In this lemma we use the notations introduced above.

Lemma 15.1. *Let M be any Turing machine. Assume $\tilde{w} = \tilde{w}_0^j \tilde{w}_j^n$ and $w = w_0^i w_i^m$ to be strings such that $\tilde{w}$ and w are equivalent with respect to acceptance by M and such that $TR_M(\tilde{w}, j) = TR_M(w, i)$. Then, the four strings $\tilde{w}$, w, $\tilde{w}_0^j w_i^m$ and $w_0^i \tilde{w}_j^n$ are pairwise equivalent with respect to acceptance by M.*

Now, we are ready to prove the desired lower bound for L_{pal}. Note that following theorem is due to Jānis Bārzdiņš.

Theorem 15.2. *For every deterministic one-tape Turing machine M accepting L_{pal} there is a constant $c_M > 0$ such that $T_M(n) \geqslant c_M n^2$ for all but finitely many $n \in \mathbb{N}$.*

Proof. For avoiding unnecessary technical details let us assume* that n is divisible by 4.

Claim 1. *Let M be any deterministic one-tape Turing machine accepting L_{pal}. Then there is a constant D_M such that for all sufficiently large n there are strings $w \in L_{pal}$ satisfying*

$$|TR_M(w, j)| \geqslant D_M \cdot n \text{ for all } j = \frac{n}{4} + 1,\ \frac{n}{4} + 2,\ \ldots,\ \frac{n}{2}\ .$$

Before proving Claim 1, we show that it directly implies the assertion of the theorem. Using Observation 1, we get

$$T_M(w) \geqslant \sum_{j=\frac{n}{4}+1}^{\frac{n}{2}} |TR_M(w, j)| \geqslant \frac{n}{4} D_M n = \frac{D_M}{4} n^2\ ,$$

and hence the theorem follows by setting $c_M = D_M/4$.

We continue by showing Claim 1. Let $M = [B, Z, A]$ be such that M accepts L_{pal}. We shall prove that $D_M = 1/(8 \log_2 k)$ will do, where $k = |Z|$. Let $N(j, n)$ for $\frac{n}{4}+1,\ \frac{n}{4}+2\ \ldots,\ \frac{n}{2}$ be the number of palindromes w over $\{a, b\}$ of length n for which

$$|TR_M(w, j)| < \frac{n}{8 \log_2 k}\ .$$

We claim that $N(j, n) \leqslant 2^{\frac{3n}{8}}$. Since there are $2^{\frac{n}{2}}$ many palindromes, this will prove Claim 1.

*Reprove the theorem without making this assumption.

Let $\sigma_1, \ldots, \sigma_p$ be all pairwise different traces of length less than $\frac{n}{8 \log_2 k}$. By $A(\alpha)$ we denote the number of all palindromes of length n which have trace σ_α in position j. Then, we clearly have

$$N(j, n) = A(1) + A(2) + \cdots A(p) \ .$$

First, let us estimate p. Since $|Z| = k$ and traces are strings over Z, we immediately obtain

$$\begin{aligned} p &\leqslant k + k^2 + \cdots + k^{\frac{n}{8 \log_2 k} - 1} \\ &= \frac{k^{\frac{n}{8 \log_2 k}} - 1}{k - 1} - 1 \\ &\leqslant k^{\frac{n}{8 \log_2 k}} = 2^{\frac{n}{8}} \ . \end{aligned}$$

Next, we estimate $A(\alpha)$. Let $\tilde{w} = \tilde{w}_0^j \tilde{w}_j^n$ and $w = w_0^j w_j^m$ be palindromes of length n and let $j \leqslant n/2$. If M generates in position j the same traces on input $\tilde{w}$ and on input w, then $\tilde{w}_0^j = w_0^j$ by the replacement lemma.

Therefore, all strings contributing to $A(\alpha)$ must have the same initial part, say v. The number of palindromes with initial part v is $2^{(n/2)-j}$, i.e., $A(\alpha) \leqslant 2^{(n/2)-j}$.

Consequently,

$$\begin{aligned} N(j, n) &\leqslant p \cdot 2^{\frac{n}{2} - j} \leqslant 2^{\frac{n}{8}} \cdot 2^{\frac{n}{2} - \frac{n}{4}} \\ &= 2^{\frac{3n}{8}} \ . \end{aligned}$$

Thus, Claim 1 is proved and as shown above the theorem follows. ∎

15.2. Time Complexity Gap for Accepting Non-regular Languages

We start this subsection by asking what can be accepted by deterministic one-tape Turing machines if we allow traces bounded by any fixed constant $c > 0$. For answering this question, let us introduce the following notations, where $M = [B, Z, A]$ stands for any deterministic one-tape Turing machine, and $w \in \Sigma^*$ as well as $n \in \mathbb{N}$.

$$\begin{aligned} TR_M(w) &= \max\{|TR_M(w, j)| \mid j \in \mathbb{Z}\} \\ TR_M(n) &= \max\{TR_M(w) \mid |w| = n\} \end{aligned}$$

First, we prove the following lemma.

Lemma 15.3. *Let M be any deterministic one-tape Turing machine. If there is a constant $c > 0$ such that $TR_M(n) \leqslant c$ for all $n \in \mathbb{N}$ then $L(M) \in \mathcal{REG}$.*

Proof. Without loss of generality, we assume that the input string w is read at least once. We consider the *echo mapping* on input w of $M = [B, Z, A]$. Let $z(2), z(4), \cdots, z(2n)$ be any fixed finite sequence of states from Z. We define the echo $z(1), z(3), \ldots, z(2n-1)$ of $z(2), z(4), \cdots, z(2n)$ of M on input w as follows.

M is started on w as usual. By assumption, M leaves the input sometimes for the first time on the right hand side. We let $z(1)$ be the state when crossing position $|w|$. Now, M is put into state $z(2)$ and the head is put back to the cell where the last symbol of the input was or still is. After some time, M's head crosses again position $|w|$. We let $z(3)$ be the state the machine is taking when this event happens, then put M into state $z(4)$ and put the head again back to cell where the last symbol of the input was or still is, and so on. If M is not crossing often enough the position $|w|$ to the right, then the echo is *not* defined. For $\mathfrak{z} \in \mathbb{Z}^*$ we use $\mathrm{echo}(w, \mathfrak{z})$ to denote the echo of $\mathfrak{z}$ on input w of M.

Next, we define a relation $\sim$ over Σ^* as follows. Let $u, v \in \Sigma^*$. We set

$$u \sim v \text{ iff } \forall \mathfrak{z}[\mathfrak{z} \in \mathbb{Z}^* \ \wedge \ |\mathfrak{z}| \leqslant c \longrightarrow \mathrm{echo}(u, \mathfrak{z}) = \mathrm{echo}(v, \mathfrak{z})] \ .$$

One easily verifies that $\sim$ is an equivalence relation. Thus, Σ^* is partitioned by $\sim$ into equivalence classes. Let $\Sigma^*_{/\sim}$ denote this set of equivalence classes.

Claim 1. $|\Sigma^*_{/\sim}| < \infty$.

Since Z is finite, we directly get that $|\{\mathfrak{z} \mid \mathfrak{z} \in Z^* \ \wedge \ |\mathfrak{z}| \leqslant c\}| < \infty$. Moreover, the length of $\mathrm{echo}(w, \mathfrak{z})$ is, for all $\mathfrak{z} \in Z^*$ with $|\mathfrak{z}| \leqslant c$ also bounded by c, by the definition of the echo mapping. Hence, for every $w \in \Sigma^*$ there are only finitely many echo mappings with length less than or equal to c. But the set of all echo mappings on input w of M completely determines the equivalence class generated by w. Thus, $|\Sigma^*_{/\sim}|$ is finite and Claim 1 follows.

Moreover by assumption we have $\mathrm{TR}_M(n) \leqslant c$ for all $n \in \mathbb{N}$. Hence, if $u \sim v$ then

$$\mathrm{TR}_M(uw, |u|) = \mathrm{TR}_M(vw, |v|) \quad \text{for all } w \in \Sigma^* \ .$$

Therefore, we may conclude that

$$uw \in L(M) \quad \text{if and only if} \quad vw \in L(M) \text{ for all } w \in \Sigma^* \ .$$

But this means that $L(M)$ satisfies the Nerode relation $\sim_N$, where

$$u \sim_N v \text{ iff } \forall w[w \in \Sigma^* \longrightarrow uw \in L(M) \leftrightarrow vw \in L(M)] \ .$$

Since $u \sim v$ implies $u \sim_N v$ as shown above, we conclude that $|\Sigma^*_{/\sim_N}|$ is finite, too. By the Nerode Theorem, it follows that $L(M) \in \mathcal{REG}$. ∎

Next, we are going to sharpen Lemma 15.3 by proving a gap for $\mathrm{TR}_M(n)$. Recall that for $f, g\colon \mathbb{N} \to \mathbb{N}$ we write $f(n) = o(g(n))$ if and only if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \ .$$

Lemma 15.4. *Let M be a deterministic one-tape Turing machine and assume $\mathrm{TR}_M(n) = o(\log n)$. Then there exists a constant $c > 0$ such that $\mathrm{TR}_M(n) < c$ for all $n \in \mathbb{N}$.*

Proof. The proof is done indirectly. Suppose to the contrary that there is an infinite sequence of strings $(\nu_i)_{i\in\mathbb{N}}$ such that $\mathrm{TR}_M(\nu_{i+1}) > \mathrm{TR}_M(\nu_i)$ for all $i \in \mathbb{N}$. Moreover, without loss of generality we can choose the strings ν_i in a way such that for all $i \in \mathbb{N}$ we have

$$\mathrm{TR}_M(u) < \mathrm{TR}_M(\nu_i) \quad \text{for all strings } u \text{ with } |u| < |\nu_i| \,, \tag{15.1}$$

i.e., for no shorter string the maximum trace length $\mathrm{TR}_M(\nu_i)$ does occur.

Claim 1. *Then among all traces* $\mathrm{TR}_M(\nu_i, j)$ *with* $0 \leqslant j \leqslant |\nu_i|$ *there are no three identical ones.*

Suppose the converse. Then there are j_1, j_2, j_3 with $0 \leqslant j_1 < j_2 < j_3 \leqslant |\nu_i|$ such that

$$\mathrm{TR}_M(\nu_i, j_1) = \mathrm{TR}_M(\nu_i, j_2) = \mathrm{TR}_M(\nu_i, j_3) \,.$$

Now, let j_*, $0 \leqslant j_* \leqslant |\nu_i|$ be such that $\mathrm{TR}_M(\nu_i, j_*) \geqslant \mathrm{TR}_M(\nu_i, j)$ for all $j \in \{0, \ldots, |\nu_i|\}$. If there is more than one j_*, we take the smallest one. Next, we distinguish the following cases.

Case 1. $j_* = j_1$.

We can delete all symbols in ν_i between j_2 and j_3 without changing $\mathrm{TR}_M(\nu_i, j_*)$, thus obtaining a contradiction to (15.1).

Case 2. $j_* \neq j_1$.

Then either $j_* \in (j_1, j_2)$, $j_* \in (j_2, j_3)$ or it does not fall in any of the two intervals (j_1, j_2) and (j_2, j_3). In any case, there is one interval such that $j_* \notin (j_a, j_b)$, $a, b \in \{1, 2, 3\}$, $a \neq b$. Thus, again we can delete all symbols between j_a and j_b without changing $\mathrm{TR}_M(\nu_i, j_*)$ and obtain again a contradiction to (15.1). This proves Claim 1.

Consequently, the length of the strings ν_i is bounded by twice the number of *different* traces of length at most $\mathrm{TR}_M(\nu_i)$. We continue by estimating $|\nu_i|$. Let $M = [B, Z, A]$ and recall that $|Z| \geqslant 2$.

$$\begin{aligned} |\nu_i| &\leqslant 2 \cdot \sum_{j=0}^{\mathrm{TR}_M(\nu_i)} |Z|^j \\ &= 2 \cdot \frac{|Z|^{\mathrm{TR}_M(\nu_i)+1} - 1}{|Z| - 1} \end{aligned}$$

Thus, taking logarithms to the base $|Z|$ directly yields

$$\log_{|Z|} |\nu_i| \leqslant \log_{|Z|} \left(\frac{1}{|Z| - 1} \right) + \mathrm{TR}_M(\nu_i) + 1 \,,$$

and therefore

$$\frac{\ln 2}{\ln |Z|} \cdot \log_2 |\nu_i| \leqslant \frac{\ln 2}{\ln |Z|} \cdot \log_2 \left(\frac{1}{|Z| - 1} \right) + \mathrm{TR}_M(\nu_i) + 1 \,,$$

Hence, there is a constant $c > 0$ such that

$$\log_2 |\nu_i| \leqslant c \cdot \mathrm{TR}_M(\nu_i)$$

and consequently

$$\log_2 |\nu_i| = 0(\mathrm{TR}_M(\nu_i)) \ .$$

But the latter assertion directly implies

$$\frac{\mathrm{TR}_M(n)}{\log n} \geqslant \frac{\mathrm{TR}_M(\nu_i)}{\log |\nu_i|} \geqslant \frac{1}{c} \neq 0 \ ,$$

a contradiction to $\mathrm{TR}_M(n) = o(\log n)$. ∎

Now we are ready to show the first concrete complexity gap which has been discovered by J. Hartmanis and B. Trachtenbrot independently of each other.

Theorem 15.5. *Let M be a deterministic one-tape Turing machine and assume $T_M(n) = o(n \log n)$. Then $L(M) \in \mathcal{REG}$.*

Proof. Let $M = [B, Z, A]$ and suppose, $L(M) \notin \mathcal{REG}$. By Lemmata 15.3 and 15.4 we can conclude that $\mathrm{TR}_M(n)$ is unbounded. As in the proof of Lemma 15.4 we can choose an infinite sequence $(\nu_i)_{i \in \mathbb{N}}$ of strings such $\mathrm{TR}_M(\nu_{i+1}) > \mathrm{TR}_M(\nu_i)$ for all $i \in \mathbb{N}$ and such that (15.1) is fulfilled. Moreover, as the proof of Lemma 15.4 shows, then there is a constant $c > 0$ such that

$$\log_2 |\nu_i| \leqslant c \cdot \mathrm{TR}_M(\nu_i) \ .$$

Furthermore, by construction there are at least $(1/2)|\nu_i|$ many different traces in positions $j = 0, \ldots, |\nu_i|$. The number of traces of length k is $|Z|^k$. Let us assume that the traces of minimum possible length do occur. Now, we determine the minimum a such that

$$\sum_{k=0}^{a} |Z|^k \geqslant \frac{1}{2}|\nu_i| \ .$$

Hence, it should hold

$$\frac{|Z|^{a+1} - 1}{|Z| - 1} \geqslant \frac{1}{2}|\nu_i| \ ,$$

giving us the condition

$$|Z|^{a+1} \geqslant \frac{1}{2}|\nu_i|(|Z| - 1) + 1 \ .$$

Taking logarithms to the base $|Z|$ directly yields

$$a + 1 \geqslant \log_{|Z|} \left(\frac{1}{2}|\nu_i|(|Z| - 1) + 1 \right) \ .$$

Since $|Z| \geqslant 2$ and since $\log n$ is monotonically increasing, we therefore obtain

$$a \geqslant \log_{|Z|} \left(\frac{1}{2}|\nu_i| \right) - 1 \ .$$

Consequently, by lower bounding $T_M(|v_i|)$ by the sum of the lengths of all traces of minimum length that must occur, we obtain the following.

$$\begin{aligned}
T_M(|v_i|) &\geqslant 2 \cdot \sum_{k=0}^{a} k \cdot |Z|^k \geqslant 2a \cdot |Z|^a \\
&\geqslant 2 \cdot \left(\log_{|Z|} \left(\frac{1}{2}|v_i| \right) - 1 \right) \left(|Z|^{\log_{|Z|}\left(\frac{1}{2}|v_i|\right)-1} \right) \\
&= 2 \cdot \left(\log_{|Z|} \left(\frac{1}{2}|v_i| \right) - 1 \right) \left(\frac{|Z|^{\log_{|Z|}\left(\frac{1}{2}|v_i|\right)}}{|Z|} \right) \\
&= \left(\log_{|Z|} \left(\frac{1}{2}|v_i| \right) - 1 \right) \left(\frac{|v_i|}{|Z|} \right) \\
&= \frac{1}{|Z|} \left(|v_i| \log_{|Z|} \left(\frac{1}{2}|v_i| \right) - |v_i| \right) \\
&= \frac{1}{|Z|} \left(|v_i| \log_{|Z|} |v_i| - |v_i|(1 + \log_{|Z|} 2) \right) .
\end{aligned}$$

We can therefore conclude $T_M(n) \neq o(n \log n)$, a contradiction. Hence, $L(M)$ must be regular. ∎

15.3. Space Complexity Gaps for Accepting Non-regular Languages

First, we show that enlarging the space available to any constant does not allow to accept non-regular languages. That is, we again obtain a gap-theorem.

Theorem 15.6. *Let $c \geqslant 0$ be any constant and let M be a deterministic 2-tape Turing machine such that $S_M(n) \leqslant c$ for all $n \in \mathbb{N}$. Then $L(M)$ is regular.*

Proof. For proving this theorem, we first show that there is a constant $\hat{c}$ such that M can work at most $\hat{c}n$ many steps on inputs of length n without reaching a cycle.

The desired constant $\hat{c}$ can be estimated as follows. Let $M = [B, Z, A]$ be given. Hence, there are $|B|^k$ many pairwise different strings of length k over B. Consequently, there are at most

$$\sum_{k=0}^{c} |B|^k = \frac{|B|^{c+1} - 1}{|B| - 1}$$

many pairwise different strings of length less than or equal to c which can be written on M's work tape. For writing a string of length k on its work tape, M needs k steps. Thus, M needs

$$c \cdot \frac{|B|^{c+1} - 1}{|B| - 1}$$

many steps for writing all possible strings on its work tape. Furthermore, for estimating $\hat{c}$ it suffices to assume that M can write all these strings in every of its states on its work tape. Hence, there are at most

$$c \cdot |Z| \cdot \frac{|B|^{c+1} - 1}{|B| - 1}$$

many pairwise different steps that can be performed by M on every input symbol read on its inputs tape. Setting

$$\hat{c} = c \cdot |Z| \cdot \frac{|B|^{c+1} - 1}{|B| - 1}$$

now directly yields that M can work at most $\hat{c}n$ many steps without reaching a cycle.

Next, we can conclude that there is a deterministic one-tape Turing machine $\widehat{M}$ such that $T_{\widehat{M}}(n) = \hat{c}n$ for all $n \in \mathbb{N}$ and $L(\widehat{M}) = L(M)$. The Turing machine $\widehat{M}$ is easily obtained by encoding all possible inscriptions on M's work tape into states and the changes M can make on these inscriptions into state transitions. Here it is crucial that there are only

$$c \cdot \frac{|B|^{c+1} - 1}{|B| - 1}$$

many possible inscriptions. Finally, by Theorem 15.5 we directly get $L(M) \in \mathcal{REG}$. ∎

For making further progress we extend the definition of traces made for one-tape Turing machines (cf. Page 163) by using configurations (cf. Definition 6.8).

Definition 15.1. *Let $k \in \mathbb{N}$, $k \geqslant 1$, let M be a k-tape Turing machine and let $w \in \Sigma^*$ be any input to M. We define the **trace** of M on input w at position j to be $TR_M(w, j)$, where*

$$\begin{aligned} TR_M(w, j) \;=\; & \text{the string formed out of configurations of } M \text{ such that the} \\ & i\text{th position of } TR_M(w, j) \text{ is the configuration } M \text{ is in, when its} \\ & \text{head on the input tape is crossing the border } j \text{ for the } i\text{th time .} \end{aligned}$$

Furthermore, $TR_M(w)$ and $TR_M(n)$ are defined as before, i.e., for every input string $w \in \Sigma^*$ and all $n \in \mathbb{N}$ we set

$$\begin{aligned} TR_M(w) &= \max\{|TR_M(w, j)| \mid j \in \mathbb{Z}\} \\ TR_M(n) &= \max\{TR_M(w) \mid |w| = n\} . \end{aligned}$$

Note that for $k = 1$ we just obtain the definition of traces previously made for one-tape Turing machines.

Exercise 40. *Prove the following generalization of the Replacement Lemma.*

Let M be a deterministic Turing machine. If $TR_M(uvw, |uv|) = TR_M(uvw, |u|)$ then $TR_M(uv^iv^jw, |uv^i|) = TR_M(uw, |u|)$ and thus, $uv^iv^jw \in L(M)$ iff $uw \in L(M)$.

Next, we are going to show a complexity gap for the complexity measure "space complexity" with respect to the acceptability of non-regular languages.

Theorem 15.7. *Let M be any deterministic 2-tape Turing machine such that $S_M(n) = o(\log\log n)$. Then $L(M)$ is regular.*

Proof. By Theorem 15.6 it suffices to show that there is a constant $c > 0$ such that $S_M(n) < c$. We continue indirectly.

Suppose there is an infinite sequence $(v_i)_{i\in\mathbb{N}}$ of strings such that

$$S_M(v_{i+1}) > S_M(v_i) \quad \text{for all } i \in \mathbb{N} . \tag{15.2}$$

Then we can choose the strings v_i in a way such that for all $i \in \mathbb{N}$

$$S_M(u) < S_M(v_i) \quad \text{for all strings } u \text{ with } |u| < |v_i| .$$

Now we observe that there are no two equal traces among all traces $TR_M(v_i, j)$, where $0 < j < |v_i|$. To see this, suppose there are positions j_1 and j_2 with $0 < j_1 < j_2 < |v_i|$ and $TR_M(v_i, j_1) = TR_M(v_i, j_2)$. Then we could remove all symbols between j_1 and j_2 in v_i without reducing the amount of space needed, a contradiction to (15.2).

By Exercise 40, without loss of generality, we can assume that each single trace does not contain two identical configurations, since the part of M's work done between such identical configurations cannot influence the acceptance of the input string given.

Provided $M = [B, Z, A]$ uses ℓ cells on its work tape, there are at most

$$c_\ell = |Z| \cdot \frac{|B|^{\ell+1} - 1}{|B| - 1} \cdot \ell$$

many possible configurations of length at most ℓ. By Exercise 40, we can conclude that there are at most $c_\ell^{c_\ell}$ many different traces that can be built from the possible configurations. Therefore

$$|v_i| \leqslant c_\ell^{c_\ell} \ , \ \text{where } \ell = S_M(v_i) \ ,$$

since there are no two identical traces among all traces $TR_M(v_i, j)$, where $0 < j < |v_i|$.

Since $c_\ell = |Z| \cdot \frac{|B|^{\ell+1} - 1}{|B| - 1} \cdot \ell$, there exists an r such that

$$|v_i| \leqslant r^{r^\ell} \ .$$

Consequently,

$$\log\log|v_i| \leqslant c \cdot \ell = c \cdot S_M(v_i) \ .$$

Thus,

$$1 \ \leqslant \ c \cdot \frac{S_M(v_i)}{\log\log|v_i|} \ ,$$

a contradiction to $S_M(n) = o(\log\log n)$.

This contradiction has been caused by (15.2), and hence (15.2) cannot hold. But this means nothing else than $S_M(n) \leqslant c$ for some constant $c > 0$. Now, Theorem 15.6 implies $L(M) \in \mathcal{REG}$. ∎

Note that the gap established in Theorem 15.7 cannot be improved. For seeing this, we consider the language

$$L = \{1 * 10 * 11 * \cdots * bin(k) \mid k \in \mathbb{N}\} ,$$

where $bin(k)$ denotes the binary representation of the number k.

By using the Nerode Theorem, it is immediately obvious that $L \notin \mathcal{REG}$, since the definition of the Nerode Relation directly implies $w_1 \sim_L w_2$ iff $w_1 = w_2$. But we have the following lemma.

Lemma 15.8. *There is deterministic Turing machine* M *such that* $S_M(n) = \log\log n$ *and* $L(M) = \{1 * 10 * 11 * \cdots * bin(k) \mid k \in \mathbb{N}\}$.

Proof. Since $|1 * 10 * 11 * \cdots * bin(k)| \geqslant k$, it suffices to argue that an input string of the form $1 * 10 * 11 * \cdots * bin(k)$ can be accepted in space $\log\log k$. For doing this, one has to decide for a sequence of symbols of the form $bin(n_1) * bin(n_2)$ with $n_1, n_2 \leqslant k$ whether or not $n_2 = n_1 + 1$. This can be done by comparing the dual representations bit by bit. Hence, when crossing the separation symbol $*$, one has only to memorize the actual bit position which can be done in space $\log\log k$. Finally, one has to check whether or not the input starts with 1. ∎

Applying the techniques developed so far, we can prove the following assertion.

Theorem 15.9. *For every deterministic Turing machine* M *such that* $L(M) = \{0^n 1^n \mid n \in \mathbb{N}\}$ *the condition* $S_M(n) \neq o(\log n)$ *is satisfied.*

Proof. By Corollary 6.6, without loss of generality, we can M assume to be a deterministic 2-tape Turing machine. As in the proof of Theorem 15.7, provided $M = [B, Z, A]$ uses ℓ cells on its work tape, there are at most

$$c_\ell = |Z| \cdot \frac{|B|^{\ell+1} - 1}{|B| - 1} \cdot \ell$$

many possible configurations of length at most ℓ. Again, we choose r sufficiently large for getting

$$|Z| \cdot \frac{|B|^{\ell+1} - 1}{|B| - 1} \cdot \ell \leqslant r^\ell .$$

Suppose, $S_M(n) = o(\log n)$. Then there exists a sufficiently large m such that $S_M(2m) < \log_r m$.

Consequently, M on input any string of length $2m$ generates less than m different configurations. But this means, when working on the left block 0^m the machine M must be twice in the same configuration. Hence, these configurations appear periodically with a period length less than m. Therefore, M must also accept $0^m 1^{m+m!}$, since $m!$ is a multiple of all these possible period lengths. This contradiction to $L(M) = \{0^n 1^n \mid n \in \mathbb{N}\}$ proves the theorem. ∎

Exercise 41. *Prove or disprove* $\{0^n 1^n \mid n \in \mathbb{N}\} \in \mathit{SPACE}(\log n)$.

Exercise 42. *Show that there are neither non-trivial* S*-constructible functions nor non-trivial weakly* S*-constructible functions below* $\log\log n$*, where trivial means constant functions.*

Next, we shall define further complexity measures for nondeterministic Turing machines.

Definition 15.2. *Let* M *be a nondeterministic Turing machine and let* $w \in \Sigma^*$*. We define* $T_M^{\max}(w)$ *to be the* ***maximum number of steps*** *performed by* M *on input* w *among all possible computations on input* w*.*

Furthermore, we define $S_M^{\max}(w)$ *to be the* ***maximum number of all cells*** *on* M*'s work tapes visited by the read-write heads of* M *on input* w *among all its possible computations on input* w*.*

Both $T_M^{\max}$ and $S_M^{\max}$ are undefined for inputs on which the maximum does not exist, i.e., in particular if M does not stop.

Moreover, we set

$$\begin{aligned} T_M^{\max}(n) &= \max\{T_M^{\max}(w) \mid |w| = n\} \\ S_M^{\max}(n) &= \max\{S_M^{\max}(w) \mid |w| = n\} \end{aligned}$$

and define the resulting complexity classes $\mathit{NTIME}^{\max}(f(n))$ and $\mathit{NSPACE}^{\max}(f(n))$ analogously as above.

We continue with the following important lemma.

Lemma 15.10. *Let* $f\colon \mathbb{N} \to \mathbb{N}$ *be any* S*-constructible function. Then we have*

$$\mathit{NSPACE}^{\max}(f(n)) = \mathit{NSPACE}(f(n)) \ .$$

Proof. The inclusion $\mathit{NSPACE}^{\max}(f(n)) \subseteq \mathit{NSPACE}(f(n))$ is obvious by the definitions of the complexity classes $\mathit{NSPACE}^{\max}(f(n))$ and $\mathit{NSPACE}(f(n))$.

For showing the part $\mathit{NSPACE}(f(n)) \subseteq \mathit{NSPACE}^{\max}(f(n))$, let M be any given nondeterministic Turing machine such that $S_M(n) \leqslant f(n)$ for all $n \in \mathbb{N}$. We are going to construct a nondeterministic Turing machine M' such that $L(M') = L(M)$ and $S_{M'}^{\max}(n) \leqslant f(n)$.

Since f is S-constructible, there exists a deterministic Turing machine $\tilde{M}$ such that $S_{\tilde{M}}(w) = f(|w|)$ for all $w \in \Sigma^*$. Without loss of generality, we may assume that all cells visited by $\tilde{M}$ on the work tape are marked by a special symbol.

Thus, on input any $w \in \Sigma^*$, the desired machine M' first simulates machine $\tilde{M}$ until it stops. Next, it simulates machine M until it either stops or tries to use more space than marked by $\tilde{M}$. If M has stopped and accepted the input w, then M' accepts w, too. If M tries to use more space than marked by $\tilde{M}$, then M' rejects the input w and stops.

By assumption, there is an accepting computation path of M using at most space $f(|w|)$. Hence, M' will eventually simulate this computation part. Since in all other cases, M' is rejecting the input, we have $w \in L(M')$ if and only if $w \in L(M)$. Finally, by construction, the machine M' uses on all its computations space $f(n)$. Thus, the lemma follows. ∎

Next, we show a result analogous to Lemma 15.10 for time complexity. Before doing this, you should solve the following exercise.

Exercise 43. *Show that there are neither non-trivial T-constructible functions nor non-trivial weakly T-constructible functions below n, where trivial means constant functions.*

Lemma 15.11. *Let $f: \mathbb{N} \to \mathbb{N}$ be any T-constructible function. Then we have*

$$NTIME^{\max}(f(n)) = NTIME(f(n)) \ .$$

Proof. The inclusion $NTIME^{\max}(f(n)) \subseteq NTIME(f(n))$ is obvious by the definitions of the complexity classes $NTIME^{\max}(f(n))$ and $NTIME(f(n))$.

For showing the part $NTIME(f(n)) \subseteq NTIME^{\max}(f(n))$, let M be any given nondeterministic Turing machine such that $T_M(n) \leqslant f(n)$ for all $n \in \mathbb{N}$. We are going to construct a nondeterministic Turing machine M' such that $L(M') = L(M)$ and $T^{\max}_{M'}(n) \leqslant f(n)$.

The construction is performed in two steps. First, we construct a machine M'' such that $L(M'') = L(M)$ and $T^{\max}_{M''}(n) \leqslant 2 \cdot f(n)$. By Exercise 43 we additionally know that n is the smallest non-trivial T-constructible function. Thus, we can apply Theorem 6.3 to obtain M' from M''.

The nondeterministic Turing machine M'' is obtained as follows. Since f is T-constructible, there exists a deterministic Turing machine $\tilde{M}$ such that $T_{\tilde{M}}(w) = f(|w|)$ for all $w \in \Sigma^*$. So, M'' simulates simultaneously $\tilde{M}$ and M on input w by working one step as $\tilde{M}$ and then one step as M, and so on. If $\tilde{M}$ stops first, M'' does a last step for M. If M does not accept the input in this step, M'' rejects the input, otherwise it accepts it, too.

If $\tilde{M}$ has not stopped yet, but M does, then M'' accepts the input w if and only if M has accepted w. Thus, the theorem follows. ∎

Consequently, the complexity measures space and time for nondeterministic Turing machines are in a sense robust. Therefore, in the following, we shall mainly deal with $S_M(n)$ and $T_M(n)$, but only occasionally with $S^{\max}_M(n)$ and $T^{\max}_M(n)$.

We finish this part by shortly analyzing the minimum amount of space needed by nondeterministic Turing machines for accepting non-regular languages.

The following Theorem can proved analogously as Theorem 15.6. We therefore omit the proof here.

Theorem 15.12. *Let* $c \geqslant 0$ *be any constant and let* M *be a nondeterministic 2-tape Turing machine such that* $S_M(n) \leqslant c$ *for all* $n \in \mathbb{N}$. *Then* $L(M)$ *is regular.*

Theorem 15.13. *Let* M *be any nondeterministic 2-tape Turing machine such that* $S_M^{\max}(n) = o(\log \log n)$. *Then* $L(M)$ *is regular.*

Proof. By Theorem 15.12 it suffices to show that there is a constant $c > 0$ such that $S_M(n) < c$.

We continue indirectly. Suppose there is an infinite sequence $(v_i)_{i \in \mathbb{N}}$ of strings such that

$$S_M^{\max}(v_{i+1}) > S_M^{\max}(v_i) \quad \text{for all } i \in \mathbb{N} . \tag{15.3}$$

Then we can choose the strings v_i in a way such that for all $i \in \mathbb{N}$

$$S_M^{\max}(u) < S_M^{\max}(v_i) \quad \text{for all strings } u \text{ with } |u| < |v_i| .$$

Next, we fix any computation of M that needs the maximum amount of space. The rest can then be shown as in the proof of Theorem 15.7. ∎

Note that it remains open whether or not Theorem 15.13 does also hold for the complexity measure S_M.

Now, we look at the GAP problem and show how it can be used to prove a special case of Theorem 8.3.

15.4. More Properties of the GAP Problem

Next, we provide a concrete function f such that GAP $\in$ *SPACE*$(f(n))$. Unfortunately, we are not able to prove GAP $\in$ *SPACE*$(\log n)$, but only a slightly weaker result.

Theorem 15.14. GAP $\in$ *SPACE*$(\log^2 n)$.

Proof. Let $G = (V, E)$ be any directed graph with vertex set $\{1, \ldots, m\}$ and edge set E. Then G can be represented as a string of length $n = O(m^2 \log m)$. Without loss of generality we can assume that the distinguished start node and the distinguished end node is 1 and m, respectively. We define the Boolean function GAP(i, j, ℓ) as follows.

$$\text{GAP}(i, j, \ell) = \begin{cases} 1 , & \text{if there is a path in } G \text{ between nodes } i \text{ and } j \\ & \text{having length at most } \ell, \\ 0 , & \text{otherwise.} \end{cases}$$

For computing GAP(i, j, ℓ) we shall use the following procedure, where $(i = j)$ denotes the Boolean predicate that is 1 if $i = j$ and 0 otherwise. The other predicates used below are defined analogously.

PROCEDURE $GAP(\mathtt{i},\mathtt{j},\ell)$;

if $\ell = 0$ then $GAP := (\mathtt{i} = \mathtt{j})$ else

if $\ell = 1$ then $GAP := (((\mathtt{i},\mathtt{j}) \in \mathsf{E}) \vee (\mathtt{i} = \mathtt{j}))$ else

$$GAP := \bigvee_{\mathtt{k}=1}^{\mathtt{m}} \left[\left(GAP\left(\mathtt{i},\mathtt{k},\left\lceil \frac{\ell}{2} \right\rceil\right)\right) \wedge \left(GAP\left(\mathtt{k},\mathtt{j},\left\lfloor \frac{\ell}{2} \right\rfloor\right)\right)\right]$$

Now, for deciding GAP it suffices to call $GAP(1,\mathtt{m},\mathtt{m})$. The maximum number of recursive calls is the bounded by $\log \mathtt{m}$. At each level of recursion, one has to store the actual node number in binary. This needs space at most $1+\log \mathtt{m}$. Consequently, the total amount of space needed is $\mathsf{O}(\log^2 \mathtt{m}) = \mathsf{O}(\log^2 \mathtt{n})$. ∎

Some remarks are mandatory here. The procedure presented in the proof of Theorem 15.14 does not achieve the optimal running time, since it must be called super-polynomially often. On the other hand, since $\mathcal{NL} \subseteq \mathcal{P}$, we also know that there is an acceptor for GAP working in polynomial time. However, it remains open whether or not we can construct an acceptor for GAP achieving simultaneously polynomial time complexity and a space bound of $(\log \mathtt{n})^{\mathsf{O}(1)}$. Hence, it also remains *open* whether or not

$$\begin{aligned} \mathcal{NL} &\subseteq \mathcal{PLOPS} \\ \mathcal{P} &\subseteq \mathit{SPACE}(\log^2 \mathtt{n}) \quad \text{or} \\ \mathit{SPACE}(\log^2 \mathtt{n}) &\subseteq \mathcal{P}\ . \end{aligned}$$

Nevertheless, Theorem 15.14 proves a special case of Theorem 8.3. Next, we prove a very general theorem relating non-deterministic and deterministic space complexity. This is done by using the so-called ***padding method***. The idea of the padding method is easily explained. For a given language L one defines a language L_0 of lower complexity by "stretching the inputs" as long as necessary. Now, assuming we have a method for deciding L_0, we can transform this decision procedure back for obtaining a decision procedure for L. The following theorem is due to Savitch [3].

Theorem 15.15. *Let* $\mathsf{f}(\mathtt{n})$ *be an* S*-constructible function satisfying* $\mathsf{f}(\mathtt{n}) \geqslant \log \mathtt{n}$ *and let* $\varepsilon > 0$ *be such that* $\mathcal{NL} \subseteq \mathit{SPACE}(\log^{1+\varepsilon} \mathtt{n})$*. Then we also have*

$$\mathit{NSPACE}(\mathsf{f}(\mathtt{n})) \subseteq \mathit{SPACE}\left(\mathsf{f}^{1+\varepsilon}(\mathtt{n})\right)\ .$$

Proof. Let $\mathsf{L} \in \mathit{NSPACE}(\mathsf{f}(\mathtt{n}))$. By using the padding-function g defined as

$$\mathsf{g}(\mathtt{n}) = 2^{\mathsf{f}(\mathtt{n})} - \mathtt{n} - 1$$

we define the desired language L_0 as follows

$$\mathsf{L}_0 = \{w * 0^{\mathsf{g}(|w|)} \mid w \in \mathsf{L}\}\ .$$

We continue with the following claim.

Claim 1. *Provided* $L \in NSPACE(f(n))$ *we have* $L_0 \in \mathcal{NL}$.

We have to define a NDTM M' accepting L_0. Let M be any NDTM witnessing $L \in NSPACE(f(n))$.

Let v be any given string. For finding out if $v \in L_0$, the NDTM M' first checks whether or not $v = w*0^k$ for some $k \in \mathbb{N}$. If this is the case, M' uses a binary counter to test whether or not $|v| = 2^{f(|w|)}$. This can be done by using at most $f(|w|)+1$ work tape cells.

Now, if $|v| = 2^{f(|w|)}$, the NDTM M' uses the NDTM M for checking if $w \in L$. Provided $w \in L$, the NDTM M' will find a computation of M witnessing this. Note that the amount of space needed by M' in this simulation of M is bounded by $f(|w|)$.

Taking into account that indeed

$$\begin{aligned} |v| &= |w| + 1 + \left|0^{g(|w|)}\right| = |w| + 1 + g(|w|) \\ &= |w| + 1 + 2^{f(|w|)} - |w| - 1 \\ &= 2^{f(|w|)} \ , \end{aligned}$$

we directly get $L_0 \in \mathcal{NL}$. This proves Claim 1.

Next, by assumption we additionally know that $\mathcal{NL} \subseteq SPACE(\log^{1+\varepsilon} n)$. Hence, we can conclude $L_0 \in SPACE(\log^{1+\varepsilon} n)$. Thus, it remains to show any *deterministic* Turing machine obeying a space bound of $\log^{1+\varepsilon} n$ can be used for deciding L in space $f^{1+\varepsilon}(n)$. This is done by the following claim.

Claim 2. $L_0 \in SPACE(\log^{1+\varepsilon} n)$ *implies* $L \in SPACE(f^{1+\varepsilon}(n))$.

We have to construct a DTM M' deciding L. Let any string w over the underlying alphabet be given as input. For deciding whether or not $w \in L$, we use ideas similar to those exploited in the proof of Lemma 8.4. However, we are only allowed to use at most space $f^{1+\varepsilon}(n)$. Therefore, we *cannot* write $v = w * 0^{g(|w|)}$ on any of the work tapes of M', since $|v| = 2^{f(|w|)}$.

Thus, we construct a translator which, on input w and $bin(k)$ on the first work tape, where $0 \leqslant k \leqslant 2^{f(|w|)}$, will print the kth symbol of v on a special work tape of M' which is then used as input tape for the deterministic machine deciding L_0 which we like to simulate. Note that the translator uses space at most $f(|w|)$.

Finally, we combine the translator with a simulator for the DTM M deciding L_0. Each symbol M wishes to read to read on its input tape is then produced by the translator and after having been read by M erased. Consequently, the total amount of space needed by M' is bounded by

$$(\log |v|)^{1+\varepsilon} = (f(|w|))^{1+\varepsilon} \ .$$

Now, by construction we have $w \in L$ if and only if $v \in L_0$, and thus M' accepts w if and only if M accepts v. ∎

Clearly, Theorem 15.15 directly implies Theorem 8.3, since for $\varepsilon = 1$ we already know that $\mathcal{NL} \subseteq \mathit{SPACE}(\log^2 n)$. Nevertheless, Theorem 15.15 is stronger than Theorem 8.3. In particular, Theorem 15.15 directly allows one to transform any improvement achieved at the log-level upwards. The converse, however, is not true, i.e., it is usually not possible to transform results achieved above the log-level downwards.

Finally, Theorem 15.15 allows the following corollary which we have already mentioned in Lecture 8.

Corollary 15.16. $\mathcal{PSPACE} = \mathcal{NPSPACE}$

Next, we shall present some more $\mathcal{NL}$-complete problems. This material should help you to get acquainted with proof techniques suitable for showing $\mathcal{NL}$-completeness results.

15.5. More $\mathcal{NL}$-complete Problems

While we had to prove the $\mathcal{NL}$-completeness of GAP from scratch, now the situation becomes easier, since from now on its suffices to reduce a problem already known to be $\mathcal{NL}$-complete to the problem on hand. We continue with the definition of two problems.

First, we define a variant of the well-known satisfiability problem.

SAT_2:

Input: A propositional expression e in conjunctive normal form such that each clause contains at most two literals.

Problem: Is e not satisfiable ?

Next, we define the ***associative generation problem*** (abbr. AGEN).

AGEN:

Input: An associative binary operation over a finite set A as well as subset $S \subseteq A$.

Problem: Does a fixed element $a \in A$ belong to the algebraic closure of S ?

Theorem 15.17. SAT_2 *and* AGEN *are* $\mathcal{NL}$*-complete.*

We leave it as an exercise to proof Theorem 15.17.

Next, we define particular classes of directed graphs that are commonly studied in the literature.

Definition 15.3. *Let* $G = (V, E)$ *be a directed graph with vertex set* $V = \{1, \ldots, n\}$.

(1) G *is called a* ***monotone graph*** *provided* $(i, j) \in E$ *implies* $i \leqslant j$ *for all edges* $(i, j) \in E$.

(2) G *is said to be a* ***binary graph****, if all vertices of* G *have input and output valence at most 2.*

(3) *Let* $f:\mathbb{N}\to\mathbb{N}$ *be any function.* G *is said to be a graph with* ***bandwidth*** $f(n)$ *provided* $(i,j)\in E$ *implies* $|i-j|\leqslant f(n)$ *for all edges* $(i,j)\in E$.

The restriction of GAP *to the types of graphs defined in* (1) *through* (3) *is denoted by* MGAP, GAP_2, *and* $\mathrm{GAP}(f(n))$.

First we show that bounding the bandwidth $f(n)$ has direct influence to the complexity of the language $\mathrm{GAP}(f(n))$.

Theorem 15.18. *Let* $f:\mathbb{N}\to\mathbb{N}$ *be any function such that* $2\leqslant f(n)\leqslant n$. *Then we have*

$$\mathrm{GAP}(f(n)\in \mathit{SPACE}(\log(f(n))\cdot\log n)\ .$$

Proof. For proving this theorem we use the procedure $\mathit{GAP}(i,j,\ell)$ defined within the proof of Theorem 15.14. However, when calling $\mathit{GAP}(1,n,n)$, we do not store the vertex numbers, but instead only the difference of the actual vertices to the number of their parent vertex. By the given band with, this will require only $\log(f(n))$ tape cells. The number of recursive calls remains $\log n$. Finally, taking into account that $f(n)\geqslant 2$, we see that $\log(f(n))\cdot\log n\geqslant\log n$. Hence, whenever necessary, we can reconstruct a true vertex number. Thus, the theorem follows. ∎

15.6. The Complexity of MGAP and GAP_2

In this part of our course we take a closer look at MGAP and GAP_2. First, we deal with MGAP.

Theorem 15.19. GAP *is log-space reducible to* MGAP.

Proof. Let $G=(V,E)$ be any given directed graph with vertex set $V=\{1,\ldots,n\}$. Let 1 be the distinguished start node, and let n be the distinguished end node. We are going to describe a log-space computable transformation mapping G to a monotone graph $G'=(V',E')$ with respect to the lexicographical order.

We set $V'=V\times V$ and $E'=E'_1\cup E'_G$. The edge sets E'_1 and E'_G are defined as follows. E'_1 is independent of E and connects vertices having the same second component. More precisely, we define

$$E'_1=\{((i,j),(i+1,j))\mid\ j=1,2,\ldots,n\ \wedge\ i=1,2,\ldots,n-1\}\ .$$

The edge set E'_G is obtained from E in the following way.

$$E'_G=\{((i,k),(i+1,\ell))\mid\ (k,\ell)\in E,\ i=1,2,\ldots,n-1\}\ .$$

Hence, by construction, every path in G corresponds to a path in G' such that the first component is always increased. Thus, there is a path in G between 1 and n if and only if there is a path in G' between $(1,1)$ and (n,n). We leave it as an exercise to verify that G' is monotone with respect to the lexicographical order

$$(i,j)\ \longrightarrow\ (i-1)\cdot n+j\ .$$

Since this transformation is clearly log-space computable, the theorem follows. ∎

For the sake of illustration, we exemplify the construction outlined in the proof of Theorem 15.19. Let $\mathsf{G} = (\{1,2,3,4\},\{(1,2),(2,3),(3,2),(4,3)\})$. One directly sees that there is no path from 1 to 4 in the graph G.

Next, we construct G'. Both graphs G and G' are displayed in Figure 15.2 below.

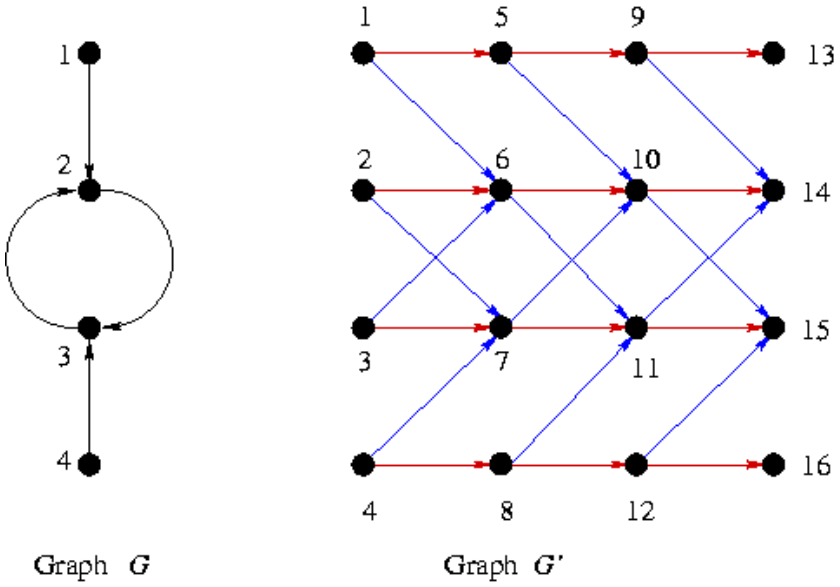

Figure 15.2: The graph G and its monotone transform G'

The vertex set V' is obtained as described above, i.e.,

$$\begin{aligned} \mathsf{V}' &= \{1,2,3,4\} \times \{1,2,3,4\} \\ &= \{(1,1),(1,2),(1,3),(1,4),(2,1),\ldots,(2,4),(3,1),\ldots,(3,4),(4,1),\ldots,(4,4)\} \end{aligned}$$

These vertices are mapped by the lexicographical order to the following numbers:

$$\begin{aligned} (1,1) &\longrightarrow (1-1)\cdot 4+1 = 1 \\ (1,2) &\longrightarrow (1-1)\cdot 4+2 = 2 \\ &\vdots \\ (1,4) &\longrightarrow (1-1)\cdot 4+4 = 4 \\ (2,1) &\longrightarrow (2-1)\cdot 4+1 = 5 \\ &\vdots \\ (4,4) &\longrightarrow (4-1)\cdot 4+4 = 16 \end{aligned}$$

The edge set E'_1 is easily computed, i.e.,

$$\begin{aligned} E'_1 &= \{((1,1),(2,1)),((1,2),(2,2)),\ldots,((3,4),(4,4))\} \\ &= \{(1,5),(2,6),(3,7),(4,8),(5,9),\ldots,(9,13),(10,14),(11,15),(12,16)\}\,. \end{aligned}$$

The edges from E'_1 are drawn in red in Figure 15.2. Finally, we compute E'_G, i.e.,

$$\begin{aligned} E'_G &= \{((1,1),(2,2)),((1,2),(2,3)),((1,3),(2,2)),\ldots,((3,4),(4,3))\} \\ &= \{(1,6),(2,7),(3,6),(4,7),\ldots,(9,14),\ldots,(12,15)\}\,. \end{aligned}$$

The edges from E'_G are drawn in blue in Figure 15.2.

The following corollary is left as an exercise.

Corollary 15.20. MGAP *is* $\mathcal{NL}$*-complete.*

Next, we turn our attention to GAP_2.

Theorem 15.21. GAP *is log-space reducible to* GAP_2*.*

Proof. Let $G = (V, E)$ be any given directed graph with vertex set $V = \{1,\ldots,n\}$. Let 1 be the distinguished start node, and let n be the distinguished end node. We are going to describe a log-space computable transformation mapping G to a binary graph $G' = (V', E')$. Again, we set $V' = V \times V$. Furthermore, we let $E' = E'_I \cup E'_G$, where the sets E'_I and E'_G are defined as follows.

$$E'_I = \{((1,j),(2,j)),(2,j),(3,j)),\ldots,((n,j),(1,j))\mid\ j = 1,\ldots,n\}\,,$$

and

$$E'_G = \{((j,i),(i,j))\mid\ (i,j) \in E,\ i \neq j\}\,.$$

Intuitively speaking, each vertex v in G corresponds to an n-gon in G' such that the valences of v are distributed to the corners of the n-gon.

By construction, G' is a binary graph. Moreover, the transformation given above is obviously log-space computable. Finally, it is easy to see that there is a path in G from 1 to n if and only if there is path in G' from $(1,1)$ to (n,n). We omit the details. ∎

Again, we are going to exemplify the transformation described in the proof of Theorem 15.21. Let $G = (\{1,2,3,4\},\{(1,2),(1,3),(1,4),(2,3),(4,3)\})$. So, there is a path from 1 to 4 as witnessed by the edge $(1,4)$.

Next, we construct G'. Both graphs G and G' are displayed in Figure 15.3 below.

The vertex set V' is obtained as described above, i.e.,

$$\begin{aligned} V' &= \{1,2,3,4\} \times \{1,2,3,4\} \\ &= \{(1,1),(1,2),(1,3),(1,4),(2,1),\ldots,(2,4),(3,1),\ldots,(3,4),(4,1),\ldots,(4,4)\} \end{aligned}$$

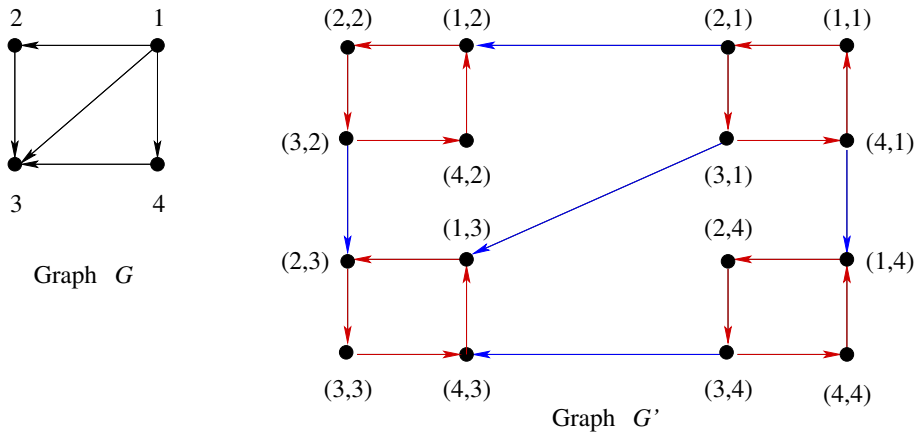

Figure 15.3: The graph G and its binary transform G'

The edge set E'_{I} is as follows:

$$\begin{aligned}\mathsf{E}'_{\mathrm{I}} &= \{((1,1),(2,1)),((2,1),(3,1)),((3,1),(4,1)),((4,1),(1,1)),\ldots,\\ &= ((1,4),(2,4)),((2,4),(3,4)),((3,4),(4,4)),((4,4),(1,4))\}\,.\end{aligned}$$

Finally, we calculate the edge set E'_{G}.

$$\mathsf{E}'_{\mathsf{G}} = \{((2,1),(1,2)),((3,1),(1,3)),((4,1),(1,4)),((3,2),(2,3)),((3,4),(4,3))\}\,.$$

The edges from E'_{I} are displayed red and the edges from E'_{G}, i.e., those corresponding to the original edges in G are drawn in blue in Figure 15.3.

Again, we leave it as an exercise to prove the following corollary.

Corollary 15.22. GAP_2 *is* $\mathcal{NL}$*-complete.*

We finish this lecture by pointing to an interesting open problem. If we switch from directed graphs to undirected ones, then we obtain the problem UGAP. Of course, undirected graphs can be considered as special cases of directed graphs, i.e., those ones having a symmetric edge relation. Thus, the complexity of UGAP is bounded by the complexity of GAP. However, a precise characterization of UGAP's complexity has not been obtained so far. In particular, it remains open whether or not UGAP $\in \mathcal{L}$ or whether or not UGAP is $\mathcal{NL}$-hard. We refer the interested reader to [1] and [2] for further details.

Moreover, many more interesting results concerning the complexity classes studied so far can be found in the literature.

References

[1] M. Ajtai and R. Fagin (1988), Reachability is harder for directed than for undirected finite graphs, *in* Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 358 – 367.

[2] H. LEWIS AND C. PAPADIMITRIOU (1982), Symmetric space-bounded computation, *Theoretical Computer Science* **19**, 161 – 187.

[3] W. J. SAVITCH (1970), Relationships between nondeterministic and deterministic tape complexities, *Journal of Computer and System Sciences*, 4(2):177-192.

Appendix for Cryptography

In this appendix, we provide a bit more material for further reading which had to be omitted due to the lack of time but which may be interesting.

First, we provide another example illustrating Theorem 10.1. We assume the following mapping of numbers to letters:

b	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
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	26

Figure 16.1: Mapping numbers to letters

Example 16.1. Suppose we have eavesdropped the cipher NUObT and, additionally, ...RY.... Moreover, we know that the message is related to a popular telecast in Germany. The title of it is *XY unresolved*. Thus, we conjecture $\mathsf{R} = \mathsf{f}(\mathsf{X})$ and $\mathsf{Y} = \mathsf{f}(\mathsf{Y})$. Since R, X and Y are the 18th, 24th, and 25th letter, respectively, we obtain the following system of linear congruences:

$$\begin{aligned} 18 &\equiv \mathsf{a} \cdot 24 + \mathsf{r} \bmod 27 \text{ and} \\ 25 &\equiv \mathsf{a} \cdot 25 + \mathsf{r} \bmod 27 \end{aligned}$$

Applying the algorithm described in the proof of Theorem 10.1, we directly obtain $\mathsf{a} \equiv 7 \bmod 27$ and $\mathsf{r} \equiv 12 \bmod 27$. Therefore, we can decipher the encrypted message NUObT as follows:

$$\begin{array}{rcrcrcl} \mathsf{f}^{-1}(\mathsf{N}) & = & \mathsf{f}^{-1}(14) & \equiv & 8 & = & \mathsf{H} \\ \mathsf{f}^{-1}(\mathsf{U}) & = & \mathsf{f}^{-1}(21) & \equiv & 9 & = & \mathsf{I} \\ \mathsf{f}^{-1}(\mathsf{O}) & = & \mathsf{f}^{-1}(15) & \equiv & 12 & = & \mathsf{L} \\ \mathsf{f}^{-1}(\mathsf{b}) & = & \mathsf{f}^{-1}(0) & \equiv & 6 & = & \mathsf{F} \\ \mathsf{f}^{-1}(\mathsf{T}) & = & \mathsf{f}^{-1}(20) & \equiv & 5 & = & \mathsf{E} \end{array}$$

That is, we have obtained the German word *HILFE* which means *HELP*. This example also shows another aspect. If one can correctly *guess at least part of the plaintext*, cryptanalysis becomes usually much easier. For example, when using cryptography in governmental applications, politeness may create enormous danger. Just assume someone writes all the time "Dear Mr. President." The same is true for sending the same message twice, but using different keys.

The cryptosystems studied so far a special cases of so-called affine cryptosystems. We continue with a closer look at them.

16.1. Affine Cryptosystems

The principal idea of affine cryptosystems is as follows. Let $\mathcal{A}$ be any fixed finite and non-empty alphabet with $|\mathcal{A}| = \mathsf{N}$. We assume that the messages to be sent are

spelled out using the alphabet $\mathcal{A}$. We define the set of all *plaintext message units* to be the set of all k-tuples of letters from $\mathcal{A}$, where $k \geqslant 1$ is any fixed natural number. In order to assign numbers to the plaintext message units, we proceed as follows. First, we assign the numbers $0, 1, \ldots, N-1$ to the letters in $\mathcal{A}$. For example, if $\mathcal{A} = \{b, A, B, C, \ldots, Z\}$, we may use the assignment $b \to 0$, $A \to 1$, $B \to 2$, $\ldots$, $Z \to 26$. Obviously, this mapping is bijective. Next, let $w = x_{k-1} \cdots x_0$ be any plaintext message unit, i.e., $x_i \in \mathcal{A}$, $0 \leqslant i \leqslant k-1$. Using the assignment defined above, we replace the letters x_i by the corresponding numbers w_i, and define the coding $c(w)$ of w by

$$c(w) = \sum_{j=0}^{k-1} w_j N^j \ .$$

That is, we canonically express w as an $N-$ary number in the number system having the basis N.

The enciphering of $c(w)$ is performed via the *affine transformation* t defined as follows:

$$t(c(w)) = (a \cdot c(w) + r) \bmod N^k \text{ where } a, r \in \{0, 1, \ldots N^k - 1\} \ .$$

Moreover, in order to ensure decipherability, we additionally require $\gcd(a, N^k) = 1$. The ordered pair (a, r) is referred to as *secret key* and has to be kept ***secretly*** by the sender and receiver.

Finally, we express $t(c(w))$ as a number to the basis N, i.e., we write

$$t(c(w)) = \sum_{j=0}^{k-1} y_j N^j \ \text{ where } y_j \in \{0, \ldots, N-1\} \ .$$

Using the inverse assignment of numbers to letters, we obtain the wanted *ciphertext units* over $\mathcal{A}^k$.

The deciphering is performed *mutatis mutandis*. The only difference is that the receiver has to compute $(\hat{a}, \hat{r})$ as follows: $\hat{a} = a^{-1} \bmod N^k$ (* this is possible, since we have assumed $\gcd(a, N^k) = 1$ *), and $\hat{r} = N^k - a^{-1} r \bmod N^k$. Instead of applying t, the receiver applies t^{-1} defined as $t^{-1}(y) = \hat{a}y + \hat{r} \bmod N^k$.

Remark: The number of possible keys is given by $N^k \cdot \varphi(N^k)$, where φ is *Euler's totient function*. Note that $\varphi(n) = n \prod_{p|n}(1 - \frac{1}{p})$, p prime. There are also some special cases we should remember, i.e.,

$\varphi(p) = p - 1$, provided p is prime, and

$\varphi(p^\alpha) = p^{\alpha-1}(p-1)$, if p is prime and $\alpha \geqslant 1$.

For example, if $N = 27$, $k = 2$, then $\varphi(27^2) = \varphi(3^6) = 3^5 \cdot 2 = 486$. Thus, there are 354294 many possible keys.

We continue by asking how secure are affine cryptosystems. First of all, Theorem 10.1 directly extends. Thus, we have to analyze the difficulty of finding or guessing two plaintext units for two ciphertext units. If we are in Case 2.2. or 2.3. of the cryptanalysis scenarios described in Lecture 10, then we are easily done. However, even if we are in Case 2.1. there is some hope to break the encryption. In order to see what the main weakness of affine cryptosystems is, let us consider the following example.

Example 16.2. Let $N = 27$, $k = 2$, and let $\mathcal{A} = \{b, A, B, \ldots, Z\}$. Moreover, we assign the numbers $0, 1, \ldots 26$ to $\{b, A, B, \ldots, Z\}$ in canonical order to the letters. Furthermore, let $(7, 26)$ be the key chosen. We compute the cipher of KL and ZL, respectively. Since $K \to 11$ and $L \to 12$ we obtain $c(KL) = 11 \cdot 27 + 12 = 309$. Thus, $t(c(KL)) = 7 \cdot 309 + 26 \equiv 2 \bmod 729$. Finally, $0 = 0 \cdot 27 + 2$, and therefore the cipher is bB. Analogously, using $Z \to 26$, we obtain $c(ZL) = 714$, and $t(c(ZL)) = 7 \cdot 714 + 26 \equiv 650$. Finally, $650 = 24 \cdot 27 + 2$, and therefore the cipher is XB. At this point we can make the following observation: The plaintext units KL and ZL have the common suffix L. Interestingly enough, the ciphertext units bB and XB have a common suffix, too, i.e., B. Is this a general phenomenon? The affirmative answer is provided by our next theorem.

Theorem 16.1. *Let $\mathcal{A}$ be any fixed non-empty finite alphabet with $|\mathcal{A}| = N$. Furthermore, let $k \geqslant 1$ be arbitrarily fixed, and let (a, r) be any key with $\gcd(a, N^k) = 1$. Let w and $\hat{w}$ be any two plaintext units from $\mathcal{A}^k$ having a common suffix of length $i \leqslant k$. Then the ciphers $t(c(w))$ and $t(c(\hat{w}))$ have a common suffix of length i, too.*

Proof. Let $c(w) = \sum_{j=0}^{k-1} w_j N^j$ and $c(\hat{w}) = \sum_{j=0}^{k-1} \hat{w}_j N^j$. By assumption, $w_j = \hat{w}_j$ for all $j = 0, \ldots, i-1$.

Furthermore, consider

$$y = t(c(w)) = aw + r \bmod N^k = \sum_{j=0}^{k-1} y_j N^j$$

$$\hat{y} = t(c(\hat{w})) = a\hat{w} + r \bmod N^k = \sum_{j=0}^{k-1} \hat{y}_j N^j$$

We have to show that $y_j = \hat{y}_j$ for all $j = 0, \ldots, i-1$. By construction, we additionally know

$$a \sum_{j=0}^{k-1} w_j N^j + r \equiv \sum_{j=0}^{k-1} y_j N^j \bmod N^k \tag{16.1}$$

$$a \sum_{j=0}^{k-1} \hat{w}_j N^j + r \equiv \sum_{j=0}^{k-1} \hat{y}_j N^j \bmod N^k \tag{16.2}$$

Now, observing that Equations (16.1) and (16.2) directly imply their validity modulo N^ℓ for every $\ell \leqslant k$, we get:

$$a \sum_{j=0}^{k-1} w_j N^j + r \equiv$$

$$a\sum_{j=0}^{i-1} w_j N^j + r \equiv \sum_{j=0}^{i-1} y_j N^j \bmod N^i \tag{16.3}$$

$$a\sum_{j=0}^{k-1} \hat{w}_j N^j + r \equiv$$

$$a\sum_{j=0}^{i-1} \hat{w}_j N^j + r \equiv \sum_{j=0}^{i-1} \hat{y}_j N^j \bmod N^i \tag{16.4}$$

Taking into account that the left hand sides of Equation (16.3) and (16.4) are equal since $w_j = \hat{w}_j$ for all $j = 0, \ldots, i-1$, we may conclude that the right hand sides are equal, too. By the uniqueness of number representation with respect to the basis N we thus have $y_j = \hat{y}_j$ for all $j = 0, \ldots, i-1$. ∎

But there is something more to be said concerning cryptanalysis. As already mentioned in Lecture 10, a very useful tool is *frequency analysis*. The basic idea behind this approach is to count the frequencies of the different letters in a huge and representative text. If there is no knowledge concerning the relevant subject, then one must use some general text. Applying this approach to English yields the following frequencies for general text:

E	12,31 %	O	7,94 %	S	6,59 %
T	9,59 %	N	7,19 %	R	6,03 %
A	8,05 %	I	7,18 %	H	5,14 %

The above table does not display the frequency of the blank symbol. This is due to the fact that this symbol has for sure the highest frequency. Now, all you have to do is to compute the frequencies of the different letters appearing in the ciphertext. If $k = 1$, then you may easily break the encryption scheme by trying to identify the most often appearing letter to be the cipher of E, the one with the 2nd highest frequency to be the cipher of T, and so on. Even if this fails (because the text was to short) you may just continue guessing the next probable combinations until the code breaks.

If $k > 1$, we may apply Theorem 16.1 by performing a frequency analysis of all letters at positions congruent 0 modulo k. This might result in providing us the key modulo N. Then we may proceed by using a 2-gram statistics to obtain the key modulo N^2. Clearly, we may considerably shrink the search space by using the information gained modulo N. And so on, we may continue until we have made the right guess. Note that the additional information gained modulo N, modulo N^2, ... a.s.o. considerably reduces the number of keys to be tried.

We continue by asking how to overcome the principal weakness of affine cryptosystems. This is done in the next subsection. We may be a bit surprised how simple the system presented below is. But we should keep in mind that in those days everything had to be done by hand.

16.2. The PLAYFAIR System

One of the first such systems that overcomes the weakness of affine cryptosystems pointed out by Theorem 16.1 has been proposed by the physicist Sir Charles Wheatstone in 1854. His friend Baron Playfair of St. Andrews encouraged the British government to put it into use.

Thus, the following cryptosystems are referred to as PLAYFAIR-systems. Take the letters $\mathsf{A}, \mathsf{B}, \ldots, \mathsf{Z}$ without J, and arrange them arbitrarily in a 5×5 square. Figure 16.2 displays a possible arrangement.

S	Y	D	W	Z
R	I	P	U	L
H	C	A	X	F
T	N	O	G	E
B	K	M	Q	V

Figure 16.2: A 5×5 square for PLAYFAIR.

The encryption is performed as follows.

(i) Partition the written plaintext in blocks of length 2 in a way such that no block consists of identical letters. Also make sure that the plaintext is of even length. This can be achieved by introducing irrelevant letters that can be easily recognized.

(ii) The enciphering of the blocks obtained is done as follows. If the two letters of the block are neither in the same row nor column of the square, then consider the spanned rectangle. Replace the first letter of the block by the letter in the same row of the resulting rectangle, and substitute the second letter by its corresponding letter in the other row. For example, EA spans the rectangle

A	X	F
O	G	E

Thus, E is replaced by O and A is substituted by F resulting in $\mathsf{EA} \to \mathsf{OF}$. Analogously, $\mathsf{SV} \to \mathsf{ZB}$, $\mathsf{RF} \to \mathsf{LH}$, and so on.

If both letters are in the same row or column then go one step right and down, respectively. This is done cyclically. For example, $\mathsf{HA} \to \mathsf{CX}$, $\mathsf{WX} \to \mathsf{UG}$, $\mathsf{CA} \to \mathsf{AX}$, $\mathsf{DM} \to \mathsf{PD}$, $\mathsf{RL} \to \mathsf{IR}$.

Decryption is then obviously performed by reversing the rules given above. Moreover, suffixes are no longer necessarily preserved. For instance, look at EA and CA which have the common suffix A. They are enciphered as OF and AX, respectively, which do not have a common suffix. On the other hand, the plaintext pairs HA and CA also do share the suffix A and so do their ciphers CX and AX.

Remark: The rules described above in (i) and (ii) allow several variations of PLAYFAIR-systems. Instead of using a 5×5 square one can alternatively design 3×9 or 9×3 rectangles. The latter systems again allow to include all 27 letters including the blank. Those variations were proposed by Wheatstone, too.

Furthermore, PLAYFAIR-systems can be easily memorized by using a *key word* in which no letter occurs twice. For example, let *MAGIC* be the key word chosen. Then we first write the key word into the square and rectangle, respectively, and the all remaining letters in lexicographical order (cf. Figure 16.3).

M	A	G	I	C
B	D	E	F	H
K	L	N	O	P
Q	R	S	T	U
V	W	X	Y	Z

M	A	G
I	C	B
D	E	F
H	J	K
L	N	O
P	Q	R
S	T	U
V	W	X
Y	Z	b

Figure 16.3: Two PLAYFAIR systems using the key word MAGIC.

Though the PLAYFAIR system has been really used by the Foreign Office in Great Britain starting with the Krim war and ending in World War I, it is by no means secure. To make things worse, the Foreign Office quite quickly forgot to use the variations mentioned above. In particular, the statistics of 2-grams is preserved, only the statistics of single letters is disturbed. When the 5×5 square is used, there are 625 possible 2-grams. But not all of them occur in practice. Even worse, the 18 most frequent 2-grams in German contribute already 92.93 % to all 2-grams that appear in a usual text.

Moreover, there is another peculiarity of the PLAYFAIR systems. For discovering it, let us have a look at the following 5×5 squares.

P	A	L	M	E
R	S	T	O	N
B	C	D	F	G
H	I	K	Q	U
V	W	X	Y	Z

E	P	A	L	M
N	R	S	T	O
G	B	C	D	F
U	H	I	K	Q
Z	V	W	X	Y

G	B	C	D	F
U	H	I	K	Q
Z	V	W	X	Y
E	P	A	L	M
N	R	S	T	O

Figure 16.4: Three PLAYFAIR squares giving the same encryption.

At first glance, these three squares look different. Now, let us encipher

CR YP TO GR AP HY

In all three cases we get

BS VM ON BN AL QV

The reason for this phenomenon is easily explained. As long as we cyclically shift rows and/or columns, we get the same rule for encryption/decryption. Clearly, this property severely reduces the combinatorial complexity.

Exercise 44. *The club of* kryptomanic *students organized a competition in poetry. The following encrypted poem has been submitted. Decipher it.*

OVJKIKJVOVJKIKJVKLMVKVJVVYNY
GKLUNWOFPJCTLXZHNVPMVLNYJV
PDZNFYTVVNJFVJNYAELUNWJCHJ
KLTIZNUJHKAMPJLJJNAEJVLYNS

Exercise 45. *What can be said about the complexity of breaking PLAYFAIR-systems provided the sender has added any text of odd length at the beginning of the plaintext?*

17. Using Probability Theory

In Lecture 10 we have seen that redundancy in any natural language provides useful information to possibly break Vigenère like cryptosystems. Below we take a look at natural languages from the perspective of probability theory.

17.1. Friedman's Test

First, we ask for the probability p that two randomly and independently chosen letters of a given plaintext are equal. Let $t = s_0 s_1 \dots s_{n-1}$ be any text of length n in some natural language. For the sake of presentation, we assume that t is written in English using the usual alphabet $\mathcal{A} = \{A, B, C, \dots, Z\}$. We again exclude the blank symbol from our considerations. Thus, we assume $s_i \in \mathcal{A}$ for all $i = 0, \dots n-1$. Furthermore, by $n_0, n_1, \dots,$ and n_{25} we denote the number of occurrences of As, Bs, ..., and Zs, respectively, in t. Clearly, $n = \sum_{i=0}^{25} n_i$. Furthermore, we have n possibilities to choose the first letter and n possibilities for the second one yielding an overall number of n^2 possible choices for pairs of letters chosen in the plaintext. Note that we allow to choose two times the same position. Analogously, there are n_0^2, n_1^2, ..., n_{25}^2 many possible choices for pairs containing two times the letter A, B, ..., and Z, respectively. Hence, the total number of pairs consisting of two identical letters is given by $n = \sum_{i=0}^{25} n_i^2$. Consequently, the wanted probability p can be expressed as

$$p = \frac{\sum_{i=0}^{25} n_i^2}{n^2} \,. \tag{17.1}$$

This probability is often referred to as *coincidence index*. Clearly, all our arguments used above remain valid, if we use any text instead of a plaintext, e.g. a ciphertext.

Additionally, we may use statistical information concerning text written in English. Let p_0, p_1, ..., and p_{25} denote the probability for the occurrence of letter A, B, ..., and Z, respectively, in a text written in English (cf. Figure 17.1).

letter	p_i	letter	p_i	letter	p_i	letter	p_i
A	0.0856	H	0.0528	O	0.0797	V	0.0092
B	0.0139	I	0.0627	P	0.0199	W	0.0149
C	0.0279	J	0.0013	Q	0.0012	X	0.0017
D	0.0378	K	0.0042	R	0.0677	Y	0.0199
E	0.1304	L	0.0339	S	0.0607	Z	0.0008
F	0.0289	M	0.0249	T	0.1045		
G	0.0199	N	0.0707	U	0.0249		

Figure 17.1: Probabilities for the occurrence of all letters in English.

Hence, the probability for two randomly chosen letters to be A, B, ..., and Z, is p_0^2, p_1^2, ..., and p_{25}^2, respectively. Consequently, the overall probability for two randomly chosen letters to be equal is given by

$$\tilde{p} = \sum_{i=0}^{25} p_i^2 \ . \tag{17.2}$$

Using the probabilities displayed in Figure 17.1, we obtain: $\tilde{p} = 0.06873314$, i.e., 6.87% of all possible choices yield a pair containing two times the same letter. On the other hand, if we generate a text randomly such that each letter has probability $\hat{p}_i = 1/26$, then

$$\hat{p} = \sum_{i=0}^{25} \hat{p}_i^2 = 26 \cdot \frac{1}{26^2} = \frac{1}{26} \approx 0.03846 \ . \tag{17.3}$$

Comparing the numerical values obtained from (17.2) and (17.3), we see that $\tilde{p}$ is roughly 1.8 times bigger than $\hat{p}$. At this point, we can make the following observation.

Observation 17.1. *In case of a monoalphabetical enciphering the coincidence index remains unchanged.*

This is obvious, since a monoalphabetical enciphering is nothing else than a permutation.

Next, we ask what happens in case of polyalphabetical encipherings. After a bit of reflection, it is easy to see that the coincidence index shrinks. Hence, we may use the following simple test to decide with high probability whether or not a given ciphertext results from a monoalphabetical or polyalphabetical substitution.

Substitution test $\mathcal{T}$: Count the number of occurrences of each letter in the ciphertext. Compute the coincidence index p using Formula (17.1).

If $p \approx \tilde{p}$, output "monoalphabetical substitution."

If $p << \tilde{p}$, output "polyalphabetical substitution."

Furthermore, we are interested whether or not one can gain additional information using the coincidence index. For answering this question we look at Vigenère substitutions. By Lemma 10.2 we already know that a Vigenère substitution performed by using a key word of length d can be decomposed into d monoalphabetical substitutions. If we write the plaintext in blocks of length d we obtain d columns as displayed in the proof of Lemma 10.2. Assuming a sufficiently long text, we may directly conclude from Observation 17.1 that the coincidence index remains unchanged *inside* each column. However, if we consider pairs of letters drawn from different columns, the probability to obtain pairs of equal letters decreases. Optimally, it should be close to 1/26. We take this values as a working hypothesis. Thus, we may continue as follows. Each column contains n/d many letters. For choosing the first letter, we have again n possibilities. After having chosen it, the column containing it is determined, too. Inside this column, we have again n/d many choices resulting in the overall number of n^2/d many possibilities. Outside the column containing the first letter are $n - n/d$ many remaining letters resulting in the total number of $n(n - n/d) = n^2(d-1)/d$ many such choices. Thus, the expected number of of pairs containing two equal letters can be approximated by

$$A = \frac{n^2}{d} \cdot 0.06873 + \frac{n^2(d-1)}{d} \cdot \frac{1}{26} \; . \tag{17.4}$$

Consequently, the probability to obtain a pair of equal letters is approximated by

$$\frac{A}{n^2} = \frac{1}{d} \cdot 0.06873 + \frac{(d-1)}{d} \cdot \frac{1}{26} \; . \tag{17.5}$$

Finally, since the coincidence index is an approximation of this probability, we obtain from (17.5)

$$p \approx \frac{1}{d} \cdot 0.06873 + \frac{(d-1)}{d} \cdot \frac{1}{26}$$

and therefore,

$$\begin{aligned} d &\approx \frac{0.06873 - \frac{1}{26}}{p - \frac{1}{26}} \\ &\approx \frac{0.0303}{p - 0.03846} \; . \end{aligned} \tag{17.6}$$

The latter estimate can be successfully used to obtain good estimates for the key word length, and thus to reduce the calculations when applying Kasiski's algorithm.

Formula (17.6) was discovered by William Friedman (one of the most successful cryptanalysts of all times) in 1925.

This is a good place to discuss some special cases of Formula (17.6). First, if the ciphertext to be analyzed has been enciphered using a monoalphabetical substitution, then its coincidence index p should be $\mathsf{p} \approx 0.06873$. Thus, $\mathsf{d} \approx 1$ and this nicely reflects reality. Second, if the key word is a truly randomly generated string then p should be close to 1/26. Obviously, in this case Formula (17.6) provides evidence for a huge key size. Finally, it should be remarked that Formula (17.6) is really an estimate and not an algorithm to compute the true key size. However, the estimate obtained may help to significantly reduce the number of cases to be considered during application of Kasiski's algorithm.

Exercise 46. *You have received the following cipher (for the sake of readability, it is displayed here in blocks of length 5, however, this choice was arbitrary):*

TPOGD JRJFS UBSFC SQLGP COFUQ NFDSF CLVIF OTGNW GT.

Try to decipher it.

We finish this part by applying the Substitution Test and Formula (17.6) to our example from Lecture 10.

Using the data displayed in Figure 10.7, we obtain the following frequencies for the letters A through Z in the ciphertext given in Lecture 10, Figure 10.5 (cf. Figure 17.2 below):

letter	n_i	letter	n_i	letter	n_i	letter	n_i
A	26	H	40	O	12	V	27
B	25	I	11	P	17	W	11
C	13	J	4	Q	2	X	13
D	2	K	10	R	9	Y	5
E	5	L	41	S	18	Z	14
F	13	M	21	T	23	all	400
G	23	N	4	U	11		

Figure 17.2: Frequencies for the letters A through Z in the ciphertext.

Thus, applying (17.1) we get: $\mathsf{p} = 0.05565$ which is significantly smaller than 0.06873. Thus, we conclude to have a cipher resulting from a polyalphabetical substitution. Furthermore, formula (17.6) yields the value $\mathsf{d} = 1.8$. Therefore, we have obtained high evidence for a short key word which matches nicely with the length 3 found in Lecture 10.

17.2. Security

Next, we want to deal with the security of cryptosystems from a higher point of view. Instead of looking at a particular cryptosystem, we are interested in general properties

a cryptosystems must possess to be secure. We shall distinguish between ***unconditionally secure***, ***computationally secure***, ***provably secure***, and, of course, ***insecure*** cryptosystems. While it is easy to define these notions on an intuitive level, it requires some work to provide mathematically sound definitions for these notions. Intuitively, a cryptosystem is said to be ***unconditionally secure*** if the probability p, $p < 1$, of breaking it, is independent of the computing resources available and the time an adversary is willing to spend. In contrast, we call a cryptosystem ***computationally secure*** if breaking it is possible in principle but all *known* methods of executing the computation necessary require an infeasible amount of time and/or hardware. Furthermore, a cryptosystem is said to be ***provably secure*** if it can be shown that breaking it for any *significant* number of cases implies that some other problem – such as computing the factorization of large composite integers – could be solved with comparable effort.

The distinction between computationally secure and provably secure is that while in either case the security of the system would be impeached if the underlying computationally difficult problem could be solved, the converse need not hold for computationally secure systems but *does hold* for provable secure systems.

Clearly, none of the above descriptions is a precise mathematical definition. Therefore, we continue by providing the necessary framework for making them precise. We start with the notion of unconditionally secure cryptosystems.

We model cryptosystems as follows. By $\mathbb{Z}_m$ we denote any fixed alphabet having m letters. As an example we have already considered $\mathbb{Z}_{26}$. Furthermore, let $n \in \mathbb{N}$; then we use $(x_0, \ldots, x_{n-1})$, $x_i \in \mathbb{Z}_m$, $i = 0, \ldots n-1$, to denote strings of length n over $\mathbb{Z}_m$. We refer to such strings as to ***n-grams***. By $\mathbb{Z}_{m,n}$ we denote the set of all n-grams over $\mathbb{Z}_m$. Moreover, we use Pt to denote the set of all possible plaintexts, i.e., $\mathsf{Pt} = \bigcup_{n\in\mathbb{N}} \mathbb{Z}_{m,n}$.

Definition 17.1. *A **cryptographic transformation** T is a sequence of bijective transformations $(\mathsf{T}^{(n)})_{n\in\mathbb{N}}$ with*

$$\mathsf{T}^{(n)} \colon \mathbb{Z}_{m,n} \to \mathbb{Z}_{m,n} \ ,$$

where $\mathbb{Z}_m$ is arbitrarily fixed.

*A cryptosystem $\mathcal{T}$ is a family of cryptographic transformations, i.e., $\mathcal{T} = \{\mathsf{T}_k \mid k \in \mathsf{K}\}$. K is referred to as set of all admissible **keys** (admissible means admissible for $\mathcal{T}$), and $k \in \mathsf{K}$ is called key.*

In accordance with Lecture 10, we generally assume that $\mathcal{T}$ is known to the cryptanalyst but she does not know which key k (i.e., which transformation $\mathsf{T}_k^{(n)}$) has been used. Furthermore, we assume that the generation of plaintext and the choice of a key are independent probabilistic processes.

Now, the task of the cryptanalyst can be formalized as follows: Determine the plaintext and the key, respectively, by using the available information derivable from

- the ciphertext Y received,
- the cryptosystem $\mathcal{T} = \{\mathsf{T}_\mathsf{k} \mid \mathsf{k} \in \mathsf{K}\}$ used to compute Y,
- the probability distribution Pr_{plain} over the set of all plaintexts,
- the probability distribution Pr_{key} over the set K of all keys admissible for $\mathcal{T}$,
- all possible ciphers $\mathsf{y} = \mathsf{T}_\mathsf{k}(\mathsf{x})$, where $\mathsf{k} \in \mathsf{K}$, $\mathsf{x} \in \mathsf{Pt}$.

The cryptanalyst has to make *a priori* assumptions about Pr_{plain} and Pr_{key}. Below we shall describe some possibilities how to arrive at reasonable assumptions about Pr_{plain} and Pr_{key}. Right now, we assume that these probability distributions are given. Obviously, these probability distributions induce the probability distribution Pr_{cipher} over the set of all ciphertexts Ct. Furthermore, we are interested in the following probability distributions and probabilities, respectively:

$\mathsf{Pr}_{plain,\,key}(\mathsf{x},\mathsf{k}) =_{\mathtt{df}}$ the product distribution over $\mathsf{Pt} \times \mathsf{K}$,

$\mathsf{Pr}_{plain,\,cipher}(\mathsf{x},\mathsf{y}) =_{\mathtt{df}}$ the joint distribution over Pt and Ct,

$\mathsf{Pr}_{cipher,\,key}(\mathsf{y},\mathsf{k}) =_{\mathtt{df}}$ the joint distribution over Ct and K,

$\mathsf{Pr}_{cipher,\,plain}(\mathsf{y},\mathsf{x}) =_{\mathtt{df}}$ the joint distribution over Ct and Pt,

$\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{Y}) =_{\mathtt{df}}$ the conditional probability of the plaintext x under the observed ciphertext Y,

$\mathsf{Pr}_{key|cipher}(\mathsf{k}|\mathsf{Y}) =_{\mathtt{df}}$ the conditional probability of the key k under the observed ciphertext Y,

$\mathsf{Pr}_{cipher|plain}(\mathsf{Y}|\mathsf{x}) =_{\mathtt{df}}$ the conditional probability of the cipher Y under the plaintext x,

which are defined as follows:

$$\mathsf{Pr}_{plain,\,key}(\mathsf{x},\mathsf{k}) = \mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{key}(\mathsf{k}) \tag{17.7}$$

$$\mathsf{Pr}_{plain,\,cipher}(\mathsf{x},\mathsf{y}) = \sum_{\{\mathsf{k}\in\mathsf{K} \mid \mathsf{T}_\mathsf{k}(\mathsf{x})=\mathsf{y}\}} \mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{key}(\mathsf{k}) \tag{17.8}$$

$$\mathsf{Pr}_{cipher,\,key}(\mathsf{y},\mathsf{k}) = \sum_{\{\mathsf{x}\in\mathsf{Pt} \mid \mathsf{T}_\mathsf{k}(\mathsf{x})=\mathsf{y}\}} \mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{key}(\mathsf{k}) \tag{17.9}$$

$$\mathsf{Pr}_{cipher}(\mathsf{Y}) = \sum_{\{(\mathsf{x},\mathsf{k}) \mid \mathsf{T}_\mathsf{k}(\mathsf{x})=\mathsf{Y}\}} \mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{key}(\mathsf{k}) \tag{17.10}$$

$$\mathsf{Pr}_{cipher,\,plain}(\mathsf{y},\mathsf{x}) = \sum_{\{\mathsf{k}\in\mathsf{K} \mid \mathsf{T}_\mathsf{k}^{-1}(\mathsf{y})=\mathsf{x}\}} \mathsf{Pr}_{cipher}(\mathsf{y}) \cdot \mathsf{Pr}_{key}(\mathsf{k}) \tag{17.11}$$

$$\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{Y}) = \frac{\mathsf{Pr}_{plain,\,cipher}(\mathsf{x},\mathsf{Y})}{\mathsf{Pr}_{cipher}(\mathsf{Y})} \tag{17.12}$$

$$\mathsf{Pr}_{key|cipher}(\mathsf{k}|\mathsf{Y}) = \frac{\mathsf{Pr}_{cipher,\,key}(\mathsf{Y},\mathsf{k})}{\mathsf{Pr}_{cipher}(\mathsf{Y})} \tag{17.13}$$

$$\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{X}) = \frac{\mathsf{Pr}_{cipher,\,plain}(\mathsf{y},\mathsf{X})}{\mathsf{Pr}_{plain}(\mathsf{X})}, \tag{17.14}$$

where the conditional probabilities in (17.12) and (17.13) are defined if and only if $\mathsf{Pr}_{cipher}(\mathsf{Y}) > 0$, and $\mathsf{Pr}_{cipher|plain}$ is defined if and only if $\mathsf{Pr}_{plain}(\mathsf{x}) > 0$.

Now we are ready to formalize the notion of unconditional security. The *a priori* information concerning plaintexts is provided by Pr_{plain}. So, *a priori* the most probable plaintexts of length $\mathfrak{n}$ are the $\mathfrak{n}$-grams $(\mathsf{x}_0,\ldots,\mathsf{x}_{\mathfrak{n}-1})$ for which $\mathsf{Pr}_{plain}(\mathsf{x}_0,\ldots,\mathsf{x}_{\mathfrak{n}-1})$ takes its maximum. Note that there might be more than one plaintext for which the maximum is taken. *A posteriori*, that is after having eavesdropped the ciphertext Y, the most probable plaintexts x are those ones for which $\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{Y})$ is maximized. These observations directly yield the following definition.

Definition 17.2. *A cryptosystem $\mathcal{T}$ is said to be* ***unconditionally secure*** *if*

$$\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{Y}) = \mathsf{Pr}_{plain}(\mathsf{x})$$

for all plaintexts x and all ciphertexts Y fulfilling $\mathsf{Pr}_{cipher}(\mathsf{Y}) > 0$.

Thus, for an unconditionally secure cryptosystem the probability distributions Pr_{plain} and $\mathsf{Pr}_{plain|cipher}(\cdot\,|\mathsf{Y})$ are identical for all ciphers Y provided the cipher can be generated at all by the cryptosystem $\mathcal{T}$. In other words, whatever the eavesdropped cipher Y is, for the cryptanalyst the probability to break it is the same as having not seen it. Note that this definition does not make any assumptions concerning the computing power available to the cryptanalyst nor does it limit the time she may spend.

The following theorem provides a first characterization of unconditionally secure cryptosystems.

Theorem 17.1. *A cryptosystem $\mathcal{T}$ is unconditionally secure if and only if $\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) = \mathsf{Pr}_{cipher}(\mathsf{y})$ for all plaintexts x with $\mathsf{Pr}_{plain}(\mathsf{x}) > 0$.*

Proof. First of all, consider the case $\mathsf{Pr}_{cipher}(\mathsf{y}) = 0$. Consequently, by (17.11) we get $\mathsf{Pr}_{cipher,\,plain}(\mathsf{y},\mathsf{x}) = 0$, and therefore $\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) = 0$, too, in accordance with (17.14). Hence, the theorem follows. Therefore, it suffices to consider the case $\mathsf{Pr}_{cipher}(\mathsf{y}) \neq 0$.

Necessity: Assume $\mathcal{T}$ is unconditionally secure. Thus $\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) = \mathsf{Pr}_{plain}(\mathsf{x})$ for all plaintexts x and all ciphertexts y fulfilling $\mathsf{Pr}_{cipher}(\mathsf{y}) > 0$ by Definition 17.2. Furthermore, by (17.14) we know that

$$\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) = \frac{\mathsf{Pr}_{cipher,\,plain}(\mathsf{y},\mathsf{x})}{\mathsf{Pr}_{plain}(\mathsf{x})} \text{ provided } \mathsf{Pr}_{plain}(\mathsf{x}) > 0\ .$$

Next, we apply Bayes' formula, i.e.,

$$\mathsf{Pr}(\mathsf{A}|\mathsf{B}) = \frac{\mathsf{Pr}(\mathsf{A}) \cdot \mathsf{Pr}(\mathsf{B}|\mathsf{A})}{\mathsf{Pr}(\mathsf{B})} \ , \tag{17.15}$$

for all events A and B with $\mathsf{Pr}(\mathsf{B}) > 0$. Thus, applying (17.15) to $\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y})$ directly yields

$$\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) = \frac{\mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x})}{\mathsf{Pr}_{cipher}(\mathsf{y})} \tag{17.16}$$

and hence,

$$\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) \cdot \mathsf{Pr}_{cipher}(\mathsf{y}) = \mathsf{Pr}_{plain}(\mathsf{x}) \cdot \mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) \ . \tag{17.17}$$

Since $\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) = \mathsf{Pr}_{plain}(\mathsf{x})$, Formula (17.17) immediately implies the assertion of the theorem and the necessity is shown.

Sufficiency: Now, we have the assumption $\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) = \mathsf{Pr}_{cipher}(\mathsf{y})$ for all $\mathsf{x} \in \mathsf{Pt}$ satisfying $\mathsf{Pr}_{plain}(\mathsf{x}) > 0$. We have to show $\mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) = \mathsf{Pr}_{plain}(\mathsf{x})$. Using again (17.15) (this time for $\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x})$), we get

$$\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) = \frac{\mathsf{Pr}_{plain}(\mathsf{y}) \cdot \mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y})}{\mathsf{Pr}_{cipher}(\mathsf{x})} \ . \tag{17.18}$$

Hence,

$$\mathsf{Pr}_{cipher|plain}(\mathsf{y}|\mathsf{x}) \cdot \mathsf{Pr}_{cipher}(\mathsf{x}) = \mathsf{Pr}_{plain}(\mathsf{y}) \cdot \mathsf{Pr}_{plain|cipher}(\mathsf{x}|\mathsf{y}) \ ,$$

and the theorem follows. ∎

The following theorem provides a necessary condition for unconditionally secure cryptosystems by relating the number of keys necessary to the number of plaintexts having non-zero probability.

Theorem 17.2. *Let $\mathcal{T}$ be any cryptosystem that is unconditionally secure for all n-grams x, y with $\mathsf{Pr}_{plain}(\mathsf{x}) > 0$ and $\mathsf{Pr}_{cipher}(\mathsf{y}) > 0$. Then*

$$|\mathsf{K}| \geqslant |\{\mathsf{x} \in \mathbb{Z}_{m,n} \mid \mathsf{Pr}_{plain}(\mathsf{x}) > 0\}| \ .$$

Proof. For the sake of presentation we introduce the following notations:
$\mathbb{Z}_{m,n,+,plain} =_{\mathtt{df}} \{\mathsf{x} \in \mathbb{Z}_{m,n} \mid \mathsf{Pr}_{plain}(\mathsf{x}) > 0\}$ and
$\mathbb{Z}_{m,n,+,cipher} =_{\mathtt{df}} \{\mathsf{y} \in \mathbb{Z}_{m,n} \mid \mathsf{Pr}_{cipher}(\mathsf{y}) > 0\}$.

Furthermore, without loss of generality we may assume $\mathsf{Pr}_{key}(\mathsf{k}) > 0$ for all $\mathsf{k} \in \mathsf{K}$, since otherwise we may replace K by $\hat{\mathsf{K}}$, where $\hat{\mathsf{K}}$ is just the set of all keys having non-zero probability. Now, we make the following observations:

Observation 17.2. *The transformation $\mathsf{T}_\mathsf{k}: \mathbb{Z}_{m,n,+,plain} \to \mathbb{Z}_{m,n,+,cipher}$ is injective for all $\mathsf{k} \in \mathsf{K}$.*

By definition $\mathsf{T}_k: \mathbb{Z}_{m,n} \to \mathbb{Z}_{m,n}$ is bijective. Furthermore, in accordance with (17.10) we know that

$$\mathsf{Pr}_{cipher}(y) = \sum_{\{(x,k) \,|\, \mathsf{T}_k(x)=y\}} \mathsf{Pr}_{plain}(x) \cdot \mathsf{Pr}_{key}(k) \ .$$

Thus, $\mathsf{Pr}_{cipher}(y) > 0$ iff there is at least one pair (x,k) with $\mathsf{T}_k(x) = y$ satisfying $\mathsf{Pr}_{plain}(x) > 0$ and $\mathsf{Pr}_{key}(k) > 0$. Since $\mathsf{Pr}_{key}(k) > 0$ for all $k \in K$, we directly obtain $\mathsf{T}_k(\mathbb{Z}_{m,n,+,plain}) \subseteq \mathbb{Z}_{m,n,+,cipher}$ for all $k \in K$. Hence, the restriction of T_k to $\mathbb{Z}_{m,n,+,plain}$ defines an injective mapping into $\mathbb{Z}_{m,n,+,cipher}$.

Observation 17.3. *Let $y \in \mathbb{Z}_{m,n,+,cipher}$ be arbitrarily fixed. Then, for every $x \in \mathbb{Z}_{m,n,+,plain}$ there must be a key $k \in K$ such that $\mathsf{T}_k(x) = y$.*

Suppose the converse, i.e., there exists an $\hat{x} \in \mathbb{Z}_{m,n,+,plain}$ such that $\mathsf{T}_k(\hat{x}) \neq y$ for all $k \in K$. Hence $\{k \in K \,|\, \mathsf{T}_k(\hat{x}) = y\} = \emptyset$. Since the cryptosystem is unconditionally secure we additionally know that

$$\begin{aligned} \mathsf{Pr}_{plain}(\hat{x}) &= \mathsf{Pr}_{plain|cipher}(\hat{x}|y) = \frac{\mathsf{Pr}_{plain,\,cipher}(\hat{x}, y)}{\mathsf{Pr}_{cipher}(y)} \\ &= \frac{\sum_{\{k \in K \,|\, \mathsf{T}_k(\hat{x})=y\}} \mathsf{Pr}_{plain}(\hat{x}) \cdot \mathsf{Pr}_{key}(k)}{\mathsf{Pr}_{cipher}(y)} = 0 \ , \end{aligned}$$

a contradiction to $\hat{x} \in \mathbb{Z}_{m,n,+,plain}$.

Finally, for all $x_1, x_2 \in \mathbb{Z}_{m,n,+,plain}$ with $x_1 \neq x_2$ and all $k_1, k_2 \in K$ satisfying $\mathsf{T}_{k_1}(x_1) = y = \mathsf{T}_{k_2}(x_2)$ we can conclude $k_1 \neq k_2$. In order to see this, suppose the converse, i.e., there are $x_1, x_2 \in \mathbb{Z}_{m,n,+,plain}$ with $x_1 \neq x_2$ and a key k such that $\mathsf{T}_k(x_1) = y = \mathsf{T}_k(x_2)$. However, T_k is injective, and therefore this would imply $x_1 = x_2$, a contradiction.

Consequently, the set K must contain at least as many keys as there are elements in $\mathbb{Z}_{m,n,+,plain}$. ∎

Next, we shall deal with the existence of ***unconditionally secure cryptosystems***. First we define a cryptosystem–the so-called ***one-time-pads***–and provide a proof for its unconditional security. One-time-pads have been introduced by the American engineer G.S. Vernam in 1918. Vernam proposed to introduce uncertainty at the same rate at which is was removed by redundancy among symbols of the message. His intuition was absolutely right as would be proved more than two decades later by C. Shannon [3]. However, as our Theorem 17.2 already established, this ideal requires exchanging an amount of key in advance of communication that is in most cases impractical if not totally infeasible. Nevertheless, one-time-pads *prove* the existence of unconditionally secure cryptosystems, and are therefore presented here. For the sake of convenience, we identify the possible keys of the cryptosystem to be defined with random variables.

Let $\{k_i \mid 0 \leqslant i < n\}$ be independently and identically distributed random variables taking all the values from $\mathbb{Z}_m$ equally likely. That is $\Pr(k_i = x) = 1/m$ for all $x \in \mathbb{Z}_m$, and all $i = 0, \ldots, n-1$. Now, one-time-pads are specified as follows: The desired bijections

$$T^{(n)}: X = (x_0, \ldots, x_{n-1}) \to Y = (y_0, \ldots, y_{n-1})$$

are defined by using a randomly generated key $(k_0, \ldots, k_{n-1})$ in accordance with the above assumption.

$$y_i = T_{k_i}^{(n)}(x_i) =_{df} (k_i + x_i) \bmod m \tag{17.19}$$

for all $i = 0, \ldots, n-1$. Thus, we have $\Pr_{key}((k_0, \ldots, k_{n-1})) = 1/m^n$. Moreover, the set of all keys satisfies $K = \mathbb{Z}_{m,n}$, and therefore also $|K| = m^n$. Consequently, there at least as many keys for enciphering the n-grams from $\mathbb{Z}_{m,n}$ as there elements $X \in \mathbb{Z}_{m,n}$ with $\Pr_{plain}(x) > 0$. Therefore, the condition of Theorem 17.2 is fulfilled.

Theorem 17.3. *For every plaintext source S the random variables $y_0, \ldots, y_{n-1}$ defined by (17.19) are independent and identically distributed, and every y_i , $i = 0, \ldots, n-1$, is equally distributed over $\mathbb{Z}_m$, i.e.,*

$$\Pr(y_0, \ldots, y_{n-1}) = \left(\frac{1}{m}\right)^n \quad \text{for all } y = (y_0, \ldots, y_{n-1}) \in \mathbb{Z}_{m,n} \ .$$

Proof. x_i and y_i uniquely determine k_i by $y_i - x_i \equiv k_i \bmod m$. Furthermore, all k_i are independent and identically distributed, and each k_i is equally likely chosen from $\mathbb{Z}_m$. Thus

$$\begin{aligned} \Pr_{plain,\, cipher}\{X = x, Y = y\} &= \sum_{\{k \in K \mid T_k^{(n)}(x) = y\}} \Pr_{plain}\{X = x\} \Pr_{key}(k) \\ &= \frac{1}{m^n} \Pr_{plain}\{X = x\} \ . \end{aligned}$$

Finally, by (17.10) we have

$$\begin{aligned} \Pr_{cipher}\{Y = y\} &= \sum_{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{m,n}} \Pr_{plain,\, cipher}\{X = x, Y = y\} \\ &= \frac{1}{m^n} \sum_{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{m,n}} \Pr_{plain}\{X = x\} \\ &= \frac{1}{m^n} \ . \end{aligned}$$

Thus, the y_i are independent and identically distributed, and

$$\Pr(y_0, \ldots, y_{n-1}) = \left(\frac{1}{m}\right)^n \quad \text{for all } y = (y_0, \ldots, y_{n-1}) \in \mathbb{Z}_{m,n} \ .$$

∎

Corollary 17.4. *One-time pads defined by (17.19) are unconditionally secure for all plaintexts of length* n.

Proof. We compute

$$\begin{aligned}
\Pr_{plain|cipher}\{X = x|Y = y\} &= \frac{\Pr_{plain,\,cipher}\{X = x, Y = y\}}{\Pr_{cipher}\{Y = y\}} = \\
\frac{\sum\limits_{\{k\in K \,|\, T_k^{(n)}(x)=y\}} \Pr_{plain}\{X = x\}\Pr_{key}(k)}{\Pr_{cipher}\{Y = y\}} &= \frac{\Pr_{key}(k_0,\ldots,k_{n-1})\Pr_{plain}\{X = x\}}{\Pr_{cipher}\{Y = y\}} \\
&= \frac{\frac{1}{m^n}\cdot \Pr_{plain}\{X = x\}}{\frac{1}{m^n}} \\
&= \Pr_{plain}\{X = x\}\ ,
\end{aligned}$$

where the equality in the second line directly results from the definition of the probabilities involved and the equality from the second to the third line is obtained by Theorem 17.3. ∎

17.3. Making A Priori Assumptions

Until now, we left it open in which way *a priori* assumptions concerning $\Pr_{plain}$ and $\Pr_{key}$ are made. Usually, the keys are uniformly distributed, i.e., each key is equally likely. The harder part is to find appropriate models for natural languages allowing reasonable assumptions over $\Pr_{plain}$. In the following, we describe some of the possible models, and outline generalizations.

First of all, we generally assume the cryptanalyst to know in which language the plaintext is written, e.g., in English, in Japanese, in French, in FORTRAN. Obviously, one could try to list all possible n-grams in the relevant language, where n corresponds to the length of the ciphertext eavesdropped. However, this approach would require a too huge amount of data to be processes, and moreover, the resulting probabilities would be hard to handle numerically. Therefore, it is common to model languages as probabilistic processes. The resulting model should fulfill the following properties:

- The model should reflect characteristic properties of the language modeled with "sufficient" precision. For example, in German, English and French, the letter Q is always followed by a U (e.g., Quark, question, informatique), while any other combination like QE, QP, QR, never appears (except in names like QANTAS).
- It must be possible to perform a great amount of computations within the model using a reasonable amount of time and hardware.

In principle, each language can be modeled with any desired precision. However, the complexity of the resulting models rapidly increases with the degree of precision obtained. So, for ensuring the applicability of the models, one has to compromise.

The *basic* idea of modeling languages is as follows. A plaintext source for text over $\mathbb{Z}_m$ is formalized as probabilistic process, i.e., as a finite or infinite sequence of random variables $X_0, X_1, \ldots, X_{n-1}, X_n, \ldots$. That is, the source models the generation of plaintext by a random experiment resulting in a sequence of letters $x_0, x_1, \ldots, x_{n-1}, \ldots$. A source is defined by determining the probabilities

$$\mathsf{Pr}_{plain}\{X_j = x_0, X_{j+1} = x_1, \ldots, X_{j+n-1} = x_{n-1}\}$$

for every n-gram $(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{m,n}$ and all $j, n \in \mathbb{N}$.

For arriving at mathematically sound models, the entity of all defined n-gram probabilities $\mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1})$ must fulfill the following conditions.

(1) $\mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1}) \geqslant 0$ for all $n \in \mathbb{N}$ and $(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{m,n}$

(2) $$\sum_{(x_0,\ldots,x_{n-1})\in\mathbb{Z}_{m,n}} \mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1}) = 1,$$

(3) $$\mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1}) = \sum_{(x_n,\ldots,x_{s-1})\in\mathbb{Z}_{m,s-n}} \mathsf{Pr}_{plain}(x_0, \ldots, x_{s-1}) \text{ for all } s > n.$$

Condition (1) and (2) are the classical axioms of non-negativity and normalization, respectively. Property (3) is a special case of Kolmogoroff's consistency requirement. It establishes the connection between the probability of a prefix $(x_0, \ldots, x_{n-1})$ and the set of all s-grams, $s > n$, extending it. For understanding its importance, it may be helpful to reflect about composed words in English, e.g. furthermore, moreover, therein, prefer, preferring, and the like. Its importance may become even clearer if you think of composed words in German, e.g. Bügeleisen, Waschmaschinenmonteur, Bildschirmarbeitsplatzbelehrungshandbuch, a.s.o

Next, we consider different possibilities for modeling differing from one another with respect to the degree of accuracy achieved.

Variant 1: 1-gram Source

Definition 17.3. *A plaintext source generates* **1-*grams*** *over* $\mathbb{Z}_m$ *by identical and independent random experiments if* $\mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1}) = \prod_{i=0}^{n-1} p(x_i)$, *where* $p(x_i)$ *denotes the probability to obtain* x_i.

Hence, we have to define p over $\mathbb{Z}_m$ such that $p(t) \geqslant 0$ for all $t \in \mathbb{Z}_m$, and $\sum_{t\in\mathbb{Z}_m} p(t) = 1$. The desired probabilities $p(t)$, $t \in \mathbb{Z}_m$ are empirically obtained by frequency analysis. But this is something we have already done for English when attacking the Vigenère cryptosystem (cf. Figure 17.1, pp. 192).

Now, it is easy to verify that Conditions (1) and (2) above are fulfilled for the distribution Pr_{plain} given in Definition 17.3.

Exercise 47. *Prove that the distribution* Pr_{plain} *given in Definition 17.3 does satisfy Condition (3) above.*

However, modeling languages by 1-gram plaintext source is still rough. For example, $\mathsf{Pr}_{plain}(\mathsf{HELP}) = \mathsf{Pr}_{plain}(\mathsf{LHEP})$. Also, the characteristic mentioned above that Q must be followed by U is not reflected, since $\mathsf{Pr}_{plain}(\mathsf{QE}) = 0.000156 > 0$. It may be, nevertheless, successfully applied when trying to break messages enciphered by simple cryptosystems as outlined in Lecture 4. Next, we refine the approach presented.

Variant 2: 2-gram Source

Definition 17.4. *A plaintext source generates* ***2-grams*** *over* $\mathbb{Z}_m$ *by identical and independent random experiments if* $\mathsf{Pr}_{plain}(x_0, \ldots, x_{n-1}) = \prod_{i=0}^{n-1} p(x_{2i}, x_{2i+1})$, *where* $p(x_i, x_j)$ *denotes the probability to obtain* $x_i x_j$.

Hence, we have to define $p\colon \mathbb{Z}_m \times \mathbb{Z}_m \to [0,1]$ such that $p(t,s) \geqslant 0$ for all $(t,s) \in \mathbb{Z}_m \times \mathbb{Z}_m$, and $\sum_{(t,s)\in\mathbb{Z}_m\times\mathbb{Z}_m} p(t,s) = 1$.

Again, the probability measure p is obtained by performing a frequency analysis with respect to the language in which the expected plaintexts are written. For arriving at reasonably precise estimates for the desired probabilities one has, however, to analyze much larger samples.

	A	B	C	D	E	F	G	H	I	J	K	L	M
A	7	125	251	304	13	65	151	13	311	13	67	681	182
B	114	7	2	1	394	0	0	0	74	7	0	152	6
C	319	0	52	1	453	0	0	339	202	0	86	98	4
D	158	3	4	33	572	1	20	1	273	5	0	19	27
E	492	27	323	890	326	106	93	16	118	4	27	340	253
F	98	0	0	0	150	108	0	0	188	0	0	35	1
G	122	0	0	2	271	0	20	145	95	0	0	23	3
H	646	2	5	3	2053	0	0	2	426	0	0	6	6
I	236	51	476	285	271	80	174	1	10	0	31	352	184
J	18	0	0	0	26	0	0	0	5	0	0	0	0
K	14	1	0	1	187	1	0	7	56	0	4	7	1
L	359	5	6	197	513	28	29	0	407	0	21	378	22
M	351	65	5	0	573	2	0	0	259	0	0	2	126
N	249	2	281	761	549	46	630	6	301	4	30	33	47
O	48	57	91	130	21	731	46	14	52	8	44	234	397
P	241	0	1	0	310	0	0	42	75	0	0	144	13
Q	0	0	0	0	0	0	0	0	0	0	0	0	0
R	470	15	79	129	1280	14	80	8	541	0	94	75	139
S	200	4	94	9	595	8	0	186	390	0	30	48	37
T	381	2	22	1	872	4	1	2161	865	0	0	62	27
U	72	87	103	51	91	11	80	2	54	0	3	230	69
V	65	0	0	2	522	0	0	0	223	0	0	0	1
W	282	1	0	4	239	0	0	175	259	0	0	5	0
X	9	0	15	0	17	0	0	1	15	0	0	0	1
Y	17	1	3	2	84	0	0	0	20	0	1	5	11
Z	18	0	0	0	36	0	0	0	17	0	0	1	0

Figure 17.3: Frequency of 2-grams, Part 1

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	1216	5	144	0	764	648	1019	89	137	37	17	202	15
B	0	118	0	0	81	28	6	89	2	0	0	143	0
C	3	606	0	1	113	23	237	92	0	0	0	25	0
D	8	111	0	1	49	75	2	91	15	6	0	40	0
E	1029	30	143	25	1436	917	301	36	160	153	113	90	3
F	1	326	0	0	142	3	58	54	0	0	0	5	0
G	51	129	0	0	150	29	28	58	0	0	0	6	0
H	14	287	0	0	56	10	85	31	0	4	0	15	0
I	1550	554	62	5	212	741	704	7	155	0	14	1	49
J	0	45	0	0	1	0	0	48	0	0	0	0	0
K	20	7	0	0	3	39	1	1	0	0	0	4	0
L	1	208	11	0	9	104	68	72	15	3	0	219	0
M	8	240	139	0	5	47	1	65	1	0	0	37	0
N	88	239	2	3	5	340	743	56	31	8	1	71	2
O	1232	125	164	0	861	201	223	533	188	194	7	23	2
P	1	268	103	0	409	32	51	81	0	0	0	3	0
Q	0	0	0	0	0	0	0	73	0	0	0	0	0
R	149	510	25	0	97	300	273	88	65	8	1	140	0
S	7	234	128	3	9	277	823	192	0	13	0	27	0
T	9	756	2	0	295	257	131	120	3	54	0	125	3
U	318	4	81	0	306	256	263	6	3	0	2	3	1
V	0	46	0	0	0	2	0	1	1	0	0	5	0
W	44	159	0	0	13	45	2	0	0	0	0	3	0
X	0	1	47	0	0	0	23	0	0	0	5	0	0
Y	5	64	9	0	9	44	5	4	0	3	0	2	1
Z	0	4	0	0	0	0	0	1	0	0	0	0	2

Figure 17.4: Frequency of 2-grams, Part 2

For example, Figures 17.3 and 17.4 display the relative frequencies of all 676 possible 2-grams over $\mathbb{Z}_{26} \times \mathbb{Z}_{26}$ obtained by analyzing a sample of size 67320 of English 2-grams. The entry in the row i and column j stands for the number $\mathsf{N}(\mathsf{i},\mathsf{j})$ of occurrences of the 2-gram (i,j) in the sample.

The desired probabilities are then obtained by computing

$$\mathsf{p}(\mathsf{t},\mathsf{s}) = \mathsf{N}(\mathsf{t},\mathsf{s})/67320 \ .$$

We leave it to the reader to perform this calculation. Nevertheless, a closer look to Figures 17.3 and 17.4 impressively shows that the characteristics of English are much better reflected by 2-gram plaintext sources than by 1-gram ones. For example, all entries that are 0 result in zero probability, too. Thus, $\mathsf{Pr}_{\mathit{plain}}(\mathsf{QA}) = \mathsf{Pr}_{\mathit{plain}}(\mathsf{QB}) = \ldots = \mathsf{Pr}_{\mathit{plain}}(\mathsf{QT}) = 0$. Moreover, $\mathsf{Pr}_{\mathit{plain}}(\mathsf{HELP}) = 0.0000061 > 0 = \mathsf{Pr}_{\mathit{plain}}(\mathsf{LHEP})$. On the other hand, $\mathsf{Pr}_{\mathit{plain}}(\mathsf{HELP}) = 0.0000061 < 0.0000064 = \mathsf{Pr}_{\mathit{plain}}(\mathsf{HEPL})$, despite the fact that HEPL is not an English word.

Thus, the 2-gram source model is not perfect. Alternatively, one could consider ℓ-gram sources for $\ell > 2$. This approach would require to analyze the relative frequencies of all 26^ℓ possible ℓ-grams. For arriving at reasonable counts much larger samples are

necessary.

Exercise 48. *Prove or disprove that the probability distribution defined in Definition 17.4 satisfies Kolmogoroff's consistency condition.*

A further possibility for modeling languages is provided by the theory of Markov processes which we want to present next.

Variant 3: Markov Chains

Definition 17.5. *A plaintext source generates* **1-*grams*** *over* $\mathbb{Z}_m$ *by a Markov chain with transition probabilities* $P = (p(s|t))_{s,t\in\mathbb{Z}_m}$ *and initial probability distribution* $\pi = (\pi_0, \ldots, \pi_{m-1})$ *if* $Pr_{plain}(x_0, \ldots, x_{n-1}) = \pi(x_0)p(x_1|x_0)p(x_2|x_1)\ldots p(x_{n-1}|x_{n-2})$ *for all* $n \in \mathbb{N}$ *and every* n*-gram* $(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{m,n}$.

Thereby, the following properties must be fulfilled:

(α) $p(s|t) \geqslant 0$ *for all* $0 \leqslant s, t < m$,

(β) $\sum\limits_{0\leqslant s<m} p(s|t) = 1$ *for all* $0 \leqslant t < m$,

(γ) $\pi(t) \geqslant 0$ *for all* $t = 0, \ldots, m-1$, *and* $\sum\limits_{0\leqslant s<m} \pi(s) = 1$,

(δ) $\pi(s) = \sum\limits_{0\leqslant t<m} \pi(t)p(s|t)$ *for all* $s = 0, \ldots, m-1$.

This looks much more complicated than our previous definitions. Hence, some additional remarks are in order. The general idea behind the Markov chain model is to describe a language by a *probabilistic* automaton having $|\mathbb{Z}_m|$ states. Thus, each state stands for a letter. Intuitively , the initial probability distribution π describes the probability of a plaintext to start with a particular letter. It could be obtained by counting the number of sentences (paragraphs) in a sufficiently long text starting with the letter A, B, ..., and Z and dividing these numbers by the number of all sentences (paragraphs) in this text. There is, however, a better method for computing it which we describe below. Figure 17.5 displays these probabilities.

letter	π	letter	π	letter	π	letter	π
A	0.0723	H	0.0402	O	0.0716	V	0.0117
B	0.0060	I	0.0787	P	0.0161	W	0.0078
C	0.0282	J	0.0006	Q	0.0007	X	0.0030
D	0.0483	K	0.0064	R	0.0751	Y	0.0168
E	0.1566	L	0.0396	S	0.0715	Z	0.0010
F	0.0167	M	0.0236	T	0.0773		
G	0.0216	N	0.0814	U	0.0272		

Figure 17.5: The initial probabilities $\boldsymbol{\pi}$

Do not mix these probabilities with those ones displayed in Figure 17.1 at pp. 192.

The matrix $\mathsf{P} = (\mathsf{p}(\mathsf{s}|\mathsf{t}))_{\mathsf{s},\mathsf{t}\in\mathbb{Z}_\mathsf{m}}$ describes the transition probabilities, i.e., entry $\mathsf{p}(\mathsf{s}|\mathsf{t})$ is the *conditional* probability for the event to obtain letter s under the condition that the previously obtained letter was t. Thus, P can also be computed by an appropriate frequency analysis. We may use the sample text exploited for obtaining Figures 17.3 and 17.4. Let $\mathsf{N}(\mathsf{i},\mathsf{j})$ denote the number of occurrences of the 2-gram (i,j) in the sample text. We set $\sum_{\mathsf{j}=0}^{\mathsf{m}-1} \mathsf{N}(\mathsf{i},\mathsf{j})$ and compute $\mathsf{p}(\mathsf{j}|\mathsf{i}) = \mathsf{N}(\mathsf{i},\mathsf{j})/\mathsf{N}(\mathsf{i})$. For example, $\mathsf{N}(\mathsf{A},\mathsf{G}) = 151$ as displayed in Figure 17.3. Moreover, $\mathsf{N}(\mathsf{A}) = \mathsf{N}(\mathsf{A},\mathsf{A}) + \mathsf{N}(\mathsf{A},\mathsf{B}) + \ldots + \mathsf{N}(\mathsf{A},\mathsf{Z}) = 6476$. Thus, $\mathsf{p}(\mathsf{G}|\mathsf{A}) = 151/6476 = 0.0233$. We leave it to the reader to compute the whole matrix. Now, it is immediately clear that Properties (α), (β), and (γ) are fulfilled.

Now, we sketch the announced method for computing π. The key property applied here is (δ). In matrix notation, (δ) reads as $\pi = \pi\mathsf{P}$. Now, we define $\pi^{(\ell)}(\mathsf{j})$ to be the probability that the Markov chain is in state j at the ℓth step, i.e., $\pi^{(\ell)}(\mathsf{j}) = \mathsf{Pr}[\mathsf{X}_\mathsf{n} = \mathsf{j}]$. Obviously, $\pi^{(0)}(\mathsf{j}) = \mathsf{Pr}[\mathsf{X}_0 = \mathsf{j}] = \pi_\mathsf{j}$. Assuming (δ), one can prove the remarkable result that $\pi^{(\ell)}(\mathsf{j}) = \pi_\mathsf{j}$, too. That is, the probabilities $\pi^{(\ell)}(\mathsf{j})$, $\ell = 0, 1, 2, \ldots$, *do not change* with time, but are stationary. Thus, π can be computed by solving the matrix equation $\pi = \pi\mathsf{P}$. It is beyond the scope of this lecture to prove this result here. Instead, the interested reader is encouraged to consult Feller [2].

Finally, it is not hard to see that the Markov Chain Model is not perfect either. As an easy calculation shows, $\mathsf{Pr}_{\mathit{plain}}(\mathsf{HELP}) < \mathsf{Pr}_{\mathit{plain}}(\mathsf{HEPL})$, since $\mathsf{p}(\mathsf{P}|\mathsf{L}) = 0.0041$ and $\mathsf{p}(\mathsf{L}|\mathsf{P}) = 0.0812$. On the other hand, it is sometimes better than the two gram source model. For example, consider the English word "gaga," and the non-English string "agag." In the two gram source model, we have $\mathsf{Pr}_{\mathit{plain}}(\mathsf{GAGA}) = 0.00000324 < 0.00000484 = \mathsf{Pr}_{\mathit{plain}}(\mathsf{AGAG})$, while in the Markov Chain Model $\mathsf{Pr}_{\mathit{plain}}(\mathsf{GAGA}) = \pi(\mathsf{G})\mathsf{p}(\mathsf{A}|\mathsf{G})\mathsf{p}(\mathsf{G}|\mathsf{A})\mathsf{p}(\mathsf{A}|\mathsf{G}) = 0.0216 \cdot 0.1078^2 \cdot 0.0233 = 0.000005848 > 0.000004231 = \pi(\mathsf{A}) \cdot \mathsf{p}(\mathsf{G}|\mathsf{A}) \cdot \mathsf{p}(\mathsf{A}|\mathsf{G}) \cdot \mathsf{p}(\mathsf{G}|\mathsf{A}) = \mathsf{Pr}_{\mathit{plain}}(\mathsf{AGAG})$. The following exercise points two further generalizations of the 1-gram Markov Chain Model.

Exercise 49. *Generalize Definition 17.5 to the case that* X_i *depends on* $(\mathsf{X}_{\mathsf{i}-1}, \mathsf{X}_{\mathsf{i}-2}, \ldots, \mathsf{X}_{\mathsf{i}-\mathsf{k}+1})$*, i.e., on the previous* $\mathsf{k}-1$ *letters.*

18. The Bayesian Approach to Cryptanalysis

Next, we shortly describe the Bayesian approach to cryptanalysis. Within this setting, the task of the cryptanalyst is described by ***decision functions*** δ, which model the decision

"Chiffre y originates from the deciphered plaintext $\delta(y)$."

Now, we can visualize the Bayesian formulation of cryptanalysis by the following picture (cf. Figure 18.1).

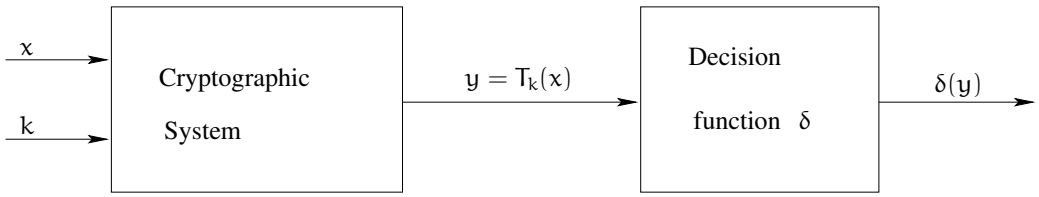

Figure 18.1: Bayesian Formulation of Cryptanalysis

Before continuing in a more formal way, please note that we make extensive use of the notations introduced above.

18.1. Decision Functions

Formally, we define decision functions as follows.

Definition 18.1. *A **deterministic decision function** $\delta = \{\delta^{(n)}\}_{n\in\mathbb{N}}$ is a sequence of transformations*

$$\begin{aligned} \delta^{(n)} &: \mathbb{Z}_{m,n} \to \mathbb{Z}_{m,n} \qquad n = 1, 2, \ldots \\ \delta^{(n)} &: y \to \delta^{(n)}(y) \end{aligned}$$

We denote the family of all deterministic decision functions by Δ_D.

Remark: Note that the cryptanalyst's decision $\delta(y)$ is the same for (x, k) for which $T_k(x) = y$. Therefore, a deterministic decision function produces an incorrect guess about the plaintext x to be deciphered for all pairs (x, k) satisfying $\delta(T_k(x)) \neq x$. Therefore, $\{(x, k) |\ \delta(T_k(x)) \neq x\}$ is the set of all incorrect guesses.

The generality of Definition 18.1 allows "many" decision functions. However, obviously not all of them are good in the sense that they deliver the right plaintext. In order to investigate which decision functions are best for the task of cryptanalysis, we have to compare decision functions qualitatively. For that purpose, next we introduce the notion of ***loss function***.

Definition 18.2. *Let* $\delta \in \Delta_D$. *We define the* ***loss function*** L_δ *for* δ *by*

$$L_\delta(x,y) = \begin{cases} 1, & \text{if} \quad \delta(y) \neq x, \\ 0, & \text{if} \quad \delta(y) = x \end{cases}$$

for all $x \in Pt$ *and* $y \in Ct$.

Looking at Definition 18.2 we see that $L_\delta(x,y)$ assigns loss 0 to a correct decision, i.e., if and only if $\delta(T_k(x)) = x$. Otherwise, the assigned loss is 1.

The mean loss of a deterministic decision function δ on n-grams is then nothing else than the expected value of the random variable $L_\delta(X,Y)$ taken with respect to the probability distribution $Pr_{plain,\,cipher}$ over the set of all ordered pairs (x,y) of n-grams, where $x \in Pt$ and $y \in Ct$. In order to simplify notation, we denote this expected value by $\mathbb{L}_{n,\delta}$. Thus, more formally we arrive at the following definition

$$\mathbb{L}_{n,\delta} =_{df} E[L_{\delta^{(n)}}] = \sum_{\{(x,y)\in\mathbb{Z}_{m,n}\}} Pr_{plain,\,cipher}(x,y) L_\delta(x,y) \ .$$

Finally, we define the mean loss of a deterministic decision function δ (taken over all n-grams, $n = 1, 2, \ldots$) to be the sequence

$$\mathbb{L}_\delta =_{df} \{\mathbb{L}_{n,\delta} \mid \ n \in \mathbb{N}\} \ .$$

Now, the deterministic Bayesian strategy consists in finding a function $\delta \in \Delta_D$ such that the expected loss $\mathbb{L}_{n,\delta}$ minimizes for every $n \in \mathbb{N}$. Here the minimum has to be taken with respect to all functions from Δ_D. Thus, we naturally arrive at the following definition.

Definition 18.3. *A deterministic decision function* δ^* *is said to be* ***optimal*** *if*

(1) $\delta^* \in \Delta_D$

(2) $\mathbb{L}_{n,\delta^*} \leqslant \mathbb{L}_{n,\delta}$ *for all* $\delta \in \Delta_D$ *and all* $n \in \mathbb{N}$.

At this point we have to ask whether or not there do exist optimal decision functions. The affirmative answer will be provided by our first theorem in this lecture. But before presenting it, we need one more definition which we provide below.

Definition 18.4. *A* ***Bayesian decision function*** *is a deterministic decision function* δ_B *such that*

$$Pr_{plain|cipher}(\delta_B(y)|Y) = \max_x Pr_{plain|cipher}(x|Y) \ .$$

By its definition $\delta_B(y) = x$ if and only if

$$Pr_{plain|cipher}(x|Y) = \max_\nu Pr_{plain|cipher}(\nu|Y) \ .$$

Moreover, $\delta_B(y) = x$ is defined for all ciphers y such that $Pr_{cipher}(y) > 0$ and it is not defined if $Pr_{cipher}(y) = 0$.

Furthermore, it is well possible that there are several $x \in \mathsf{Pt}$ for which the conditional *a posteriori* probability $\mathsf{Pr}_{plain|cipher}(x|Y)$ is taking its maximum. Consequently, there may be several Bayesian decision functions. The importance of Bayesian decision functions is pointed out by our next theorem which also provides the promised affirmative answer concerning the existence of optimal decision functions.

Theorem 18.1. *A deterministic decision function δ is optimal if and only if δ is Bayesian decision function.*

Proof. 1. Necessity:

$$\begin{aligned}
\mathbb{L}_{n,\delta} &= \mathrm{E}[\mathrm{L}_{\delta^{(n)}}] \\
&= \sum_{\{(x,y)\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{plain,\,cipher}(x,y)\mathrm{L}_\delta(x,y) \\
&= \sum_{\{(x,y)\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y)\mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) \qquad \text{(by (17.12)} \\
&= \sum_{\{x\in\mathbb{Z}_{m,n}\}}\sum_{\{y\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y)\mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) \\
&= \sum_{\{y\in\mathbb{Z}_{m,n}\}}\sum_{\{x\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y)\mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) \\
&= \sum_{\{y\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y) \cdot \sum_{\{x\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) \\
&= (*) \ .
\end{aligned}$$

Now, for any fixed $y \in \mathbb{Z}_{m,n}$ we know that $\mathrm{L}_\delta(x,y) = 0$ provided $\delta(y) = x$. Thus, those summands vanish in $(*)$. Consequently, for $y \in \mathbb{Z}_{m,n}$ we can write

$$\begin{aligned}
\sum_{\{x\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) &= \sum_{\{x\neq\delta(y)\}} \mathsf{Pr}_{plain|cipher}(x|y)\mathrm{L}_\delta(x,y) \\
&= \sum_{\{x\neq\delta(y)\}} \mathsf{Pr}_{plain|cipher}(x|y) \\
&= (**) \ ,
\end{aligned}$$

since for the remaining summands we know that $\mathrm{L}_\delta(x,y) = 1$.

Furthermore, by using the complement set, we can rewrite $(**)$ as follows

$$\begin{aligned}
(**) &= 1 - \sum_{\{\delta(y)\}} \mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y) \\
&= 1 - \mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y) \ ,
\end{aligned}$$

since there is only one summand.

Thus, we can put it all together and obtain

$$(*) = \sum_{\{y\in\mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y)\left(1 - \mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y)\right) .$$

At this point it is helpful to recall that we have started with the expected loss $\mathbb{L}_{n,\delta}$. For optimal decision functions, this expectation should be minimal. Consequently, we have to investigate under what conditions the term

$$\sum_{\{y \in \mathbb{Z}_{m,n}\}} \mathsf{Pr}_{cipher}(y)\left(1 - \mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y)\right)$$

minimizes. Now, it is easy to see that $\mathsf{Pr}_{cipher}(y)$ does not depend on $\delta^{(n)}$ at all, and thus it suffices to minimize

$$1 - \mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y) \ .$$

But the latter condition is equivalent to *maximize*

$$\mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y)$$

for every n-gram y with $\mathsf{Pr}_{cipher}(y) > 0$. Hence, $\delta^{(n)}(y)$ has to satisfy the condition

$$\mathsf{Pr}_{plain|cipher}(\delta^{(n)}(y)|y) = \max_{\{x \in \mathbb{Z}_{m,n}\}} \mathsf{Pr}_{plain|cipher}(x|y) \ .$$

Thus, the latter line and Definition 18.4 together imply that $\delta = \{\delta^{(n)} \mid \ n \in \mathbb{N}\}$ has to be a Bayesian decision function.

The sufficiency part is proved *mutatis mutandis*. We omit details. ∎

Clearly, for applying Theorem 18.1 successfully, one has to be able to compute the probabilities involved. This has to be done in a way such that reality is sufficiently good approximated. The most crucial part is to find good estimates for Pr_{plain}. Now, assuming that the cryptanalyst does know in what language the plaintext has been written, we have already provided a couple of estimates for Pr_{plain} in Lecture 5. It should be noted here that there are also good probabilistic tests for just determining the plaintext language with high probability. We refer the interested reader to Bauer (1994) for further information.

18.2. An Example

We finish this appendix by presenting an easy application of Theorem 18.1. To encipher plaintexts we use the transformation

$$t(i) \equiv i + 7 \bmod 26$$

for all $i \in \{0, 1, \ldots, 25\}$, i.e., a cyclic shift.

Thus, we have 26 keys and we assume these keys to be equally likely. That is, every key has probability 1/26.

Our example plaintext is SENDHELP. Thus, using the transformation t above, we obtain the ciphertext ZLUKOLSW (cf. Figure 18.2).

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
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

Figure 18.2: The Transformation $\mathsf{t}(\mathsf{i}) \equiv \mathsf{i} + 7 \bmod 26$

By Theorem 18.1 we know that Bayesian decision functions are optimal. Thus, we continue by computing the following conditional probabilities

$$\mathsf{Pr}_{plain|cipher}(\mathsf{X}|\text{ZLUKOLSW})$$

for X varying over all $\mathfrak{n}$-grams for which

$$\mathsf{Pr}_{plain|cipher}(\mathsf{X}|\text{ZLUKOLSW}) > 0 \ .$$

By the definition of $\mathsf{Pr}_{plain|cipher}$ it is immediately clear that we have to consider all 8-grams only. For further reducing the number of X to be considered we replace the cipher ZLUKOLSW by its numerical representation

$$\mathsf{Y} = (\mathsf{y}_0, \ldots, \mathsf{y}_7) = (25, 11, 20, 10, 14, 11, 18, 22) \ .$$

Since we always assume that the cryptanalyst knows the cryptosystem used but not the actual key, the cryptanalyst may conclude that all plaintexts to be considered must be of the form

$$\mathsf{Y} - \mathsf{k} = (\mathsf{y}_0 - \mathsf{k}, \ldots, \mathsf{y}_7 - \mathsf{k}) \bmod 26 \ , \quad \mathsf{k} \in \{0, 1, \ldots, 25\} \ .$$

By the definition of $\mathsf{Pr}_{plain,\,cipher}$ (cf. 17.8) we thus obtain

$$\mathsf{Pr}_{plain,\,cipher}(\mathsf{X}, \mathsf{Y}) = \begin{cases} \frac{1}{26} \cdot \mathsf{Pr}_{plain}(\mathsf{Y} - \mathsf{k}), & \text{if} \quad \mathsf{X} = \mathsf{Y} - \mathsf{k}, \ \text{for a } \mathsf{k} \in \{0, \ldots, 25\} \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, using its definition (cf. 17.10) we can write

$$\mathsf{Pr}_{cipher}(\mathsf{Y}) = \sum_{\mathsf{i}=0}^{25} \frac{1}{26} \cdot \mathsf{Pr}_{plain}(\mathsf{Y} - \mathsf{i}) \ .$$

Now, incorporating the last two equations into the definition of $\mathsf{Pr}_{plain|cipher}$ (cf. 17.12) yields

$$\mathsf{Pr}_{plain|cipher}(\mathsf{Y} - \mathsf{k}|\mathsf{Y}) = \frac{\mathsf{Pr}_{plain}(\mathsf{Y} - \mathsf{k})}{\sum_{\mathsf{i}=0}^{25} \mathsf{Pr}_{plain}(\mathsf{Y} - \mathsf{i})} \ .$$

The following table is listing all these probabilities for all 8-grams that have to be considered. Here, the probabilities Pr_{plain} have been computed in accordance with Definitions 17.3, 17.4, and 17.5.

k	$Y - k$	Pr_{plain} by Def. 17.3	Pr_{plain} by Def. 17.4	Pr_{plain} by Def. 17.5
0	ZLUKOLSW	0.00002353	0.00000089	0.00000309
1	YKTJNKRV	0.00000736	0.00000000	0.00000000
2	XJSIMJQU	0.00000003	0.00000000	0.00000000
3	WIRHLIPT	0.05269719	0.00418837	0.00033815
4	VHQGKHOS	0.00004373	0.00000000	0.00000000
5	UGPFJGNR	0.00012522	0.00000000	0.00000000
6	TFOEIFMQ	0.00601645	0.00000000	0.00000000
7	SENDHELP	0.35013155	**0.99580491**	**0.99965766**
8	RDMCGDKO	0.00158882	0.00000088	0.00000000
9	QCLBFCJN	0.00000412	0.00000000	0.00000000
10	PBKAEBIM	0.00100489	0.00000000	0.00000000
11	OAJZDAHL	0.00013951	0.00000000	0.00000000
12	NZIYCZGK	0.00000043	0.00000000	0.00000000
12	NZIYCZGK	0.00000043	0.00000000	0.00000000
13	MYHXBYFJ	0.00000164	0.00000000	0.00000000
14	LXGWAXEI	0.00007197	0.00000000	0.00000000
15	KWFVZWDH	0.00000134	0.00000000	0.00000000
16	JVEUYVCG	0.00001388	0.00000000	0.00000000
17	IUDTXUBF	0.00037299	0.00000000	0.00000000
18	HTCSWTAE	**0.58014350**	0.00000495	0.00000111
19	GSBRVSZD	0.00006560	0.00000000	0.00000000
20	FRAQURYC	0.00066643	0.00000000	0.00000000
21	EQZPTQXB	0.00000002	0.00000000	0.00000000
22	DPYOSPWA	0.00658287	0.00000000	0.00000000
23	COXNROVZ	0.00003584	0.00000000	0.00000000
24	BNWMQNUY	0.00005462	0.00000000	0.00000000
25	AMVLPMTX	0.00020646	0.00000000	0.00000000

Figure 18.3: Probabilities for all considered 8-grams

In all cases, the maximum is uniquely determined. Thus, δ_B would produce the following outputs: HTCSWTAE (by Def. 17.3) and $k = 18$, SENDHELP and $k = 7$ for by Definitions 17.4 and 17.5. The outcome also supports our remarks concerning the weakness of the one-gram source model.

References

[1] Bauer, F.L. (1994), *Kryptologie*, Springer-Verlag, Berlin.

[2] Feller, W. (1968), *An Introduction to Probability Theory and its Applications*, Vol. 1, 3rd ed., Wiley, New York.

[3] Shannon, C.E. (1949), Communication theory of of secrecy systems, *Bell Syst. Tech. J.* **28**, 379 – 423.

Subject Index

List of Symbols

List of Figures