Computational Number Theory

Katalin Gyarmati

katalin.gyarmati@ttk.elte.hu

Eötvös Loránd University

Lecture Notes

Institute of Mathematics - Faculty of Science - ELTE TTK

2023

Reviewed by Dr. Attila Pethő
member of the HAS
(2023.08.31)

Contents

1 Introduction

This lecture notes summarizes some of the most important results in a current number theory topic that is also connected to cryptography and computers. It analyzes the time required for fundamental operations, but it also studies speedy multiplication (FFT algorithm). It delves into some significant historical chapters in cryptography. It contains a thorough analysis of number theory techniques related to known and popular encrypting algorithms. Thus, it outlined some of the potential risks in the case of careless RSA implementations, for example (far beyond the fact that the two primes used in RSA may not be close to each other or beyond the relationship to factorization problems). It studies modern methods for solving the discrete logarithm problem (I remark here that these algorithms are quite slow). Basic primality tests and certain factorization techniques are also discussed in this lecture notes. It also provides insight into a modern approach to pseudo-random generation. It should be also emphasized that computational number theory is a considerably bigger topic than that covered in this lecture notes, which is categorized by MathSciNet with code 11Y; nonetheless, for reasons of breadth, I confined myself to the above.

The following two books serve as the foundation for the majority of the course: Neal Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994, Abhijit Das, *Computational Number Theory*, CRC Press, 2013. The chapter "Some standard cryptographic methods from the past" is mostly relied on Wikipedia articles. The part on incorrect applications of RSA was based on the writing of Tamás Dénes. The last chapter on elliptic curves relied mostly on Andrea Corbellini's internet notes and Lenstra's article on factorization. Besides from the abovementioned, this lecture notes is based on a variety of other literature. The complete bibliography can be found at the end of each chapter.

I wish the readers a pleasant time!

2 Some standard cryptographic methods from the past

2.1 Messages into space

Ivan Bell [1] planned to send a message into space in the 1960s so that foreign civilizations may hear about us. Deciphering Bell's original message is a fascinating challenge. Letters from A to Z represent different radio signals, while the dot and semicolon represent varying duration pauses between radio signals. Let's try to decode the message.

1. A. B. C. D. E. F. G. H. I. J. K. L. M. N. P. Q. R. S. T. U. V. W. Y. Z.

2. AA, B; AAA, C; AAAA, D; AAAAA, E; AAAAAA, F; AAAAAAA, G; AAAAAAAA, H; AAAAAAAAA, I; AAAAAAAAAA, J.

3. AKALB; AKAKALC; AKAKAKALD. AKALB; BKALC; CKALD; DKALE. BKELG; GLEKB. FKDLJ; JLFKD.

4. CMALB; DMALC; IMGLB.

5. CKNLC; HKNLH. DMDLN; EMELN.

6. JLAN; JKALAA; JKBLAB; AAKALAB. JKJLBN; JKJKJLCN. FNKGLFG.

7. BPCLF; EPBLJ; FPJLFN.

8. FQBLC; JQBLE; FNQFLJ.

9. CRBLI; BRELCB.

10. JPJLJRBLSLANN; JPJPJLJRCLTLANNN. JPSLT; JPTLJRD.

11. AQJLU; UQJLAQSLV.

12. ULWA; UPBLWB; AWDMALWDLDPU. VLWNA; VPCLWNC. VQJLWNNA; VQSLWNNNA. JPEWFGHLEFWGH; SPEWFGHLEFGWH.

13. GIWIHYHN; TKCYT. ZYCWADAF.

14. DPZPWNNIBRCQC.

The message was published by Bell on January 22, 1960 in “ The Japan Times ” [1].

I first heard Bell’s message at a math camp [2]. We asked the teacher if the message had been sent into space after decoding it. There was no one who knew the answer to that question. We were discussing whether it would be wise to send a message to alien civilizations, assuming that they might not be friendly. Now, more than 30 years later, I tried looking up the answer on the internet, but I could not find anything about it. In turn, I discovered a story claiming that American astronomers on the island of Puerto Rico used the radio transmitter of the Arecibo Observatory to send a message into space. Dr. Frank Drake and Carl Sagan wrote the message. It was divided into seven sections, each of which had the following terms:

The numbers one to ten

The atomic numbers of the elements hydrogen, carbon, nitrogen, oxygen, and phosphorus, which make up deoxyribonucleic acid (DNA).

The formulas for the chemical compounds that make up the nucleotides of DNA.

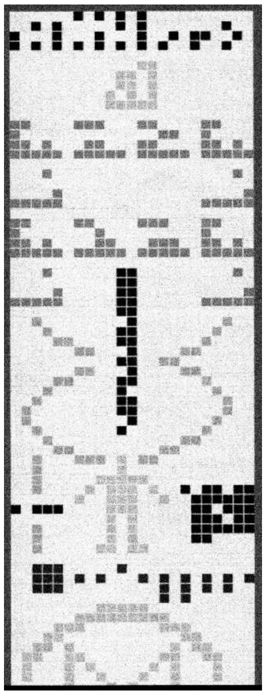
The estimated number of DNA nucleotides in the human genome, and a graphic of the double helix structure of DNA.

The dimension (physical height) of an average man, a graphic figure of a human being, and the human population of Earth.

A graphic of the Solar System, indicating which of the planets the message is coming from.

A graphic of the Arecibo radio telescope and the dimension (the physical diameter) of the transmitting antenna dish.

Arranging the message in an appropriate-sized rectangle, it looked like this:



Wikipedia has a more in-depth study of the message [3]. In reality, as I proceeded to surf the web, I discovered that on some pages, someone may have discovered the message since it came back in August 2001... It is believed that in the astronomical observatory in Chilbolton in the south of England, they discovered a crop circle with nearly the same figure contained as the above message. The "aliens" have taken up silicon among the earth's elements in the return message, indicating that they have this as the foundation of organic life. These findings, in my opinion, should be questioned.

The sad news is that the Arecibo observatory is being broken down because the building has become unsafe.

References

[1] M. Gardner, *Mathematical games*, Scientific American 213 (2) (1965), 96-101.

[2] L. Pósa, Math camp.

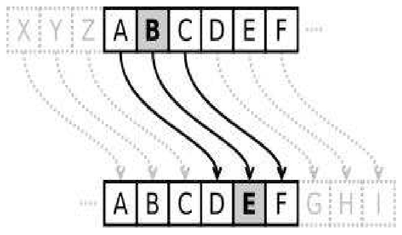
[3] Wikipedia, Arecibo message, https://en.wikipedia.org/wiki/Arecibo_message

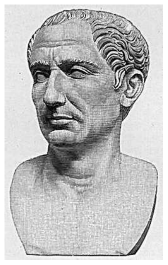
[4] Photo, Wikipedia, https://en.wikipedia.org/wiki/Arecibo_message

2.2 Caesar cipher

In this and the next few subsections, a few old ciphers will be briefly discussed, starting from antiquity. The basis of these chapters are the corresponding Wikipedia pages on the given ciphers, which I have shortened a bit. Examples to illustrate ciphers are also from Wikipedia.

The Caesar cipher was probably one of the simplest and most widely used encryption methods of its time, but it is also one of the best known encryption methods to this day. This is a substitution cipher in which each letter in the alphabet is replaced by a letter at a specified distance from it. So, for example, suppose we shift the alphabet by 3, and in the English alphabet, we replace A with D, B with E, C with F, and so on. The cipher was named after Julius Caesar, who used it to communicate with his generals.





Plaintext:	THE	QUICK	BROWN	FOX	JUMPS	OVER	THE	LAZY	DOG
Ciphertext:	QEB	NRFZH	YOLTK	CLU	GRJMP	LSBO	QEB	IXWV	ALD

However we do not know how effective this code was at the time, we think it must have been pretty reliable. This is confirmed by the fact that Caesar used it to encrypt crucial messages. It should be mentioned that the majority of Caesar's opponents were illiterate.

Deciphering the cipher is first mentioned in the 9th century, linked to frequency analysis by Al-Kindi [2]. We have no prior literature on cipher breaking, thus it is possible that Caesar's cipher was decrypted for the first time around this time.

References

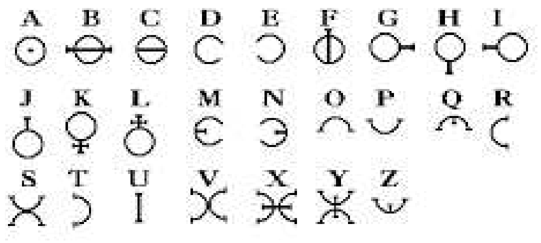
[1] Wikipedia, *Caesar cipher*, https://en.wikipedia.org/wiki/Caesar_cipher.

[2] S. Singh, *The Code Book*, Anchor, 2000, 14–20.

[3] Photos, Wikipedia, https://hu.wikipedia.org/wiki/Caesar-rejtjel.

2.3 Mono-alphabetic cipher

In monoalphabetic encryption, different symbols are assigned to the letters of the text, and the same symbol is always assigned to the same letter.


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

Példa az egyszerű subpozicio-ra:
Az alchimisták által használt titkosírás.

However, this encryption can be easily deciphered by frequency analysis: e.g., the most common letter in the English alphabet is the letter E, the most frequently occurring symbol in the encrypted text corresponds to the letter E. Similarly, the code for the second and third most frequent letters of the alphabet can be found, this is the letter A and then R in the English alphabet. After this, even short words can be examined, e.g. the word "THE" is very common in the English language, that is, the code for the letter E is often preceded by the code for the letter H. It is interesting reading perhaps the earliest and certainly the most famous literary appearance of cryptography, namely Edgar Allan Poe's short story, "The Gold Bug" [2], in which a monoalphabetic cipher is essentially deciphered. However, deciphering appears in a variety of other books including Jules Verne's novel,"Mathias Sandorf" [3] and Arthur Conan Doyle's short story, "The Adventure of the Dancing Man" [1].

References

[1] A. C. Doyle, *The Return of Sherlock Holmes*, *The Adventure of the Dancing Men*, https://sherlock-holm.es/stories/pdf/a4/1-sided/danc.pdf

[2] E. A. Poe, *The Gold Bug*, https://www.gutenberg.org/files/2147/2147-h/2147-h.htm#chap05

[3] J. Verne, *Mathias Sandorf*, https://www.rohpress.com/sandorf.html

[4] Wikipedia, *Substitution cipher*, https://en.wikipedia.org/wiki/Substitution_cipher

[5] https://kripto.blog.hu/2014/09/29/az_elso_kripto-thriller_szerzo_edgar_allan_poe

[6] Figure, *Example of simple subpositio: Ciphers used by alchemists*, https://tanarbazar.blogger.hu/2017/04/30/kodjatszma

2.4 Vigenère cipher

In this chapter, one of the most famous encryption methods in history will be discussed, the so-called Vigenère cipher based on [11].

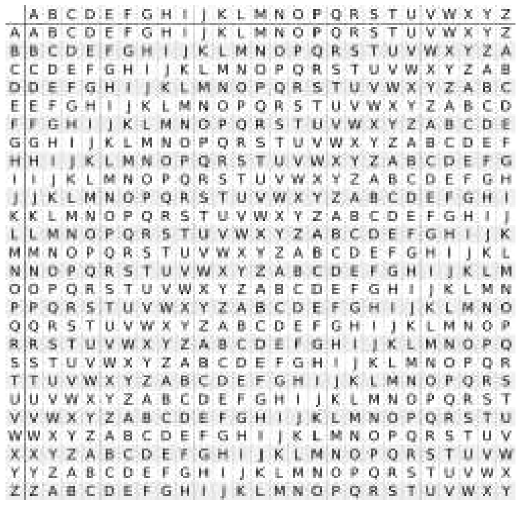
The Vigenère cipher is an encryption method that uses different Caesar ciphers, and which of these Caesar ciphers is used in the encryption depends on the letters of a particular keyword. So it is a polyalphabetic cipher.

The first polyalphabetic coding was thoroughly described and studied by Leon Battista Alberti around 1467. Alberti used different Caesar codes, where he replaced the given Caesar code with another Caesar code after a few words.

Later, John Trithemius created a cipher that already used the main component of the Viegnére code, the Viegnére table. However, Blaise de Vigenére later published his polyalphabetic cipher in 1586, where the key words were based on the original text (auto-keyed cipher). This encryption was later called Vigenére encryption.

In fact, however, what we call the Vigenére cipher today was invented more than 30 years before Vigenére's code. This code was first described by Giovan Battista Bellaso in 1553, in his book titled La cifra del. Sig. Giovan Batista.

The code is well-known because it is easy to understand and use, and it was believed to be unbreakable for a long time, several centuries; therefore, the French name "le chiffre indéchiffrable" ("uncrackable code") stuck to it.



	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
A	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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

Encoding and decoding

In addition to the text to be encoded, for encryption we will also need a secret key whose length is k. Then, the text to be encoded (in other words, plaintext) is divided into k long parts, and the secret key itself is signed under each part. We will create a table, this will be the Vigenére table, in which all the Caesar codes are listed, each line with the previous Caesar code is shifted by exactly one. The plaintext is then encrypted with the Caesar code, whose first letter in the table matches to the corresponding letter of the keyword beneath the plaintext.

This coding is easy to understand with an example, which can also be found on the Wikipedia page [11].

Coding example

Let the plaintext be "**attack at dawn**", and the key be "**lemon**". Then it is not necessary to list all the Caesar code alphabets, only those starting with the letters of the code word (and of course the original alphabet at the beginning of the table):

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

EFGHIJKLMNOPQRSTUVWXYZABCD

LMNOPQRSTUVWXYZABCDEFGHIJK

MNOPQRSTUVWXYZABCDEFGHIJKL

NOPQRSTUVWXYZABCDEFGHIJKLM

OPQRSTUVWXYZABCDEFGHIJKLMN
```

The coding then:

Plaintext:	ATTACKATDAWN
Key:	LEMONLEMONLE
Ciphertext:	LXFOPVEFRNHR

Decoding is done similarly, we find the used character in the row of the key and write the letter in the same column from the alphabet in the first row.

Cracking the code

The Vigenére code was thought to be unbreakable and absolutely safe for a long time due to the fact that the same letter can be coded in many different ways, and this can lead to many variations. However, in 1854, Charles Babbage succeeded in breaking the code, but never described his method. Friedreich Kaisiki first published a successful method of breaking the Vigenére code. He noticed that in the case of a text significantly longer than the key, there will be repetitions in the coded text. The length between two repetitions is usually a multiple of the length of the keyword, so by taking the greatest common divisor of these lengths, we get the length of the key: k, or its multiple. By dividing the text into pieces of this size, the decryption becomes simple, we can apply frequency analysis:

The coded text is divided into k groups according to the position of the letters. In the first group 1., $(k+1)$th, $(2k+1)$th, etc., in the second 2., $(k+2)$th, $2k$+2th etc. characters are added. After that, we perform a frequency analysis on the groups, thus obtaining the letters of the text and the key at the same time.

Roughly speaking, for a not too long key, it takes about 6-9 hours for an experienced code breaker to decipher the coded text.

References

[1] G. B. Bellaso, *La Cifra del Sig. Giovan Battista Belaso*, (in Italian), Venice, (Italy), 1553. Available at: Museo Galileo (Florence (Firenze), Italy)

[2] A. A. Bruen, M. A. Forcinito, *Cryptography, Information Theory, and Error-Correction: A Handbook for the 21st Century* John Wiley & Sons, 2011, p. 21.

[3] M. Gamer, *Die Polygraphia des Johannes Trithemius. Zwei Fassungen eines frühneuzeitlichen Handbuchs zur Geheimschrift*, in: Baier, Schultheiß, Jochen (eds.). *Würzburger Humanismus* (in German), Tübingen, Germany: Narr Verlag, 2015, 121–141.

[4] F. W. Kasiski, *Die Geheimschriften und die Dechiffrir-Kunst* (in German). Berlin, (Germany): E.S. Mittler und Sohn, 1863.

[5] Laurence D. Smith, *Cryptography: The Science of Secret Writing*, Courier Corporation, 1955, p. 81.

[6] Keith M. Martin, *Everyday Cryptography*, Oxford University Press, 2012, p. 142.

[7] Z. Megyesi, *Titkosírások*, Kisújszállás, Szalay könyvkiadó, 1999 (in Hungarian).

[8] D. Rodriguez-Clark, *Vigenère Cipher*, Crypto Corner, 2017, https://crypto.interactive-maths.com/vigenegravere-cipher.html

[9] J. Trithemius, *Liber quintus exordium capit* (Book 5, Ch. 1), Polygraphiae, libri sex (in Latin), Reichenau, (Germany), Johann Haselberg, 1518, p. 471. Available at: George Fabyan Collection (Library of Congress; Washington, D.C., U.S.A.).

[10] B. de Vigenère, *Traicté des Chiffres, ou Secretes Manieres d'Escrire* (in French), Paris, France, Abel l'Angelier, 1586.

[11] Wikipedia, Vigenére cipher, https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher.

[12] Photos, Wikipedia, https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher.

2.5 Vernam cipher

The most common cryptographic use of pseudorandom and random sequences is the so-called Vernam cipher.

Assume you want to encrypt some text. Each letter is then assigned a 0,1 sequence. For example:

A: 000001	B: 000010	C: 000011	D: 000101	E: 000110	F: 000111
G: 001000	H: 001001	I: 001010	J: 001011	K: 001100	L: 001101
M: 001110	N: 001111	O: 010000	P: 010001	Q: 010011	R: 010100
S: 010101	T: 010110	U: 010111	V: 011000	W: 011001	X: 011010
Y: 011011	Z: 011100				

With letter frequency analysis, text encoded in this manner is simple to decipher. (For example, the letter E is the most common in English, hence in the coded string 000110 will be the most common.)

As a result, the encoded text is further encoded by bit-by-bit addition to a pseudorandom sequence, where the addition is modulo 2. The resulting encryption method is called **Vernam cipher**:

$$\begin{aligned} \text{Message} : & (a_1, \ldots, a_N) \in \{0,1\}^N \\ \oplus \text{ Secret key} : & \underline{(e_1, \ldots, e_N)} \in \{0,1\}^N \\ \text{Encrypted message} : & (f_1, \ldots, f_N) \in \{0,1\}^N. \end{aligned}$$

Addition rule:

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

Vernam performed this technique at the early 1900s in the XX. century.

It is critical to remember that a key can only be used to encrypt a message once; otherwise, the technique can be cracked.

If the key is a true random sequence, the process is called one-time pad. In this situation, all bits of the message (independently) change or remain the same with the same probability during encryption.

As a result, this encryption approach guarantees complete security. This technology was utilized extensively during World War I and is still one of the most secure encryption methods in use today. The sole disadvantage of the Vernam cipher is that the secret key must be at least as long as the message. Delivering the secret key to the communicating parties may be difficult in this case. This will be discussed in more detail in the chapter on Diffie-Hellman key exchange.

This is now solved by using a smaller secret key, from which computers produce a random simulator (sufficiently) long sequence, referred to as a pseudorandom sequence.

In Chapter 4, I'll go into pseudorandom sequences in greater detail.

References

[1] Wikipedia, *Gilbert Vernam*, https://en.wikipedia.org/wiki/Gilbert_Vernam

[2] Photo, Wikipedia, *Gilbert Vernam*, https://en.wikipedia.org/wiki/File:Gilbert_Vernam.jpg

[3] Photo, *Computer*, http://azcoloriage.com/coloriage/31045

2.6 Public-key cryptography

In the case of Vernam's encryption method, we have seen that the persons performing the encoding and decoding uses the same secret key. Such encryptions are called symmetric cryptography. However, there is another type of encryption, the asymmetric cryptography. when the encoding and decoding parties use different keys. For example, we may want anyone to be able to write an encrypted letter to a person or institution. To do this, they publish a public key that anyone can know and and that anyone can use to send an encrypted message to the receiving party. However, the decoding is already done with another secret key, which of course is known only to the person or the institution to whom the message is intended, since it is important that only they be able to read the encrypted message. The RSA encryption method, which we can read more about in chapter 9, or the Diffie-Hellman key exchange, which will be the topic of our chapter 11, are excellent examples of this. But we can also think about the problem of digital signatures (see e.g. chapter 12.4), where it is very important that everyone can sign, at the same time, falsification must not occur. In these cases, the usual symmetric key encryption fails, but the above problems can be effectively solved with asymmetric cryptography.

References

[1] Wikipedia, *Public-key cryptography*, https://en.wikipedia.org/wiki/Public-key_cryptography.

3 Basics of Number Theory

In this chapter, I describe the basics of number theory required for the course. We do not prove the theorems in this chapter, but their proofs can be found in most books dealing with elementary number theory.

3.1 Congruences

Definition 3.1. *We say that a is congruent to b modulo m if the remainders of the division of the integers a and b by the positive integer m are the same. In other words: $m \mid a - b$. Notation:*

$$a \equiv b \pmod{m}.$$

For example: $15 \equiv 27 \pmod{12}$, but $3 \not\equiv 14 \pmod{12}$.

Theorem 3.2. *If $a \equiv x \pmod{m}$ and $b \equiv y \pmod{m}$, then*

$$a + b \equiv x + y \pmod{m} \quad \text{and} \quad ab \equiv xy \pmod{m}.$$

The following rule applies to the division:

Theorem 3.3. *Let a, b, c be integers and m be a positive integer. Then in case of*

$$ac \equiv bc \pmod{m},$$

dividing the congruence by c gives that

$$a \equiv b \pmod{\frac{m}{(c, m)}},$$

that is, we must also divide the modulus by the greatest common divisor of c and m.

One of the most famous theorems of elementary number theory is the Euler-Fermat theorem. Euler published the theorem in 1736, in which he presented his own proof of Fermat's theorem. Before I explain the theorem, the definition of Euler's φ-function follows.

Definition 3.4. *For each positive integer n, denote by $\varphi(n)$ the number of positive integers that are prime relative to n and not greater than n. Given by formula:*

$$\varphi(n) = |\{r : 1 \leq r \leq n \text{ and } (r, n) = 1\}| .$$

This function φ is multiplicative, i.e., if positive integers a and b are relative primes, then

$$\varphi(ab) = \varphi(a)\varphi(b).$$

Similarly to other multiplicative functions, φ also has the relation $\varphi(1) = 1$. If $n > 1$ and its prime factor decomposition is $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$, then

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_r}\right).$$

Knowing the definition of the function φ, we can state the famous Euler-Fermat theorem:

Theorem 3.5. (Euler-Fermat) *If a is an integer and m is a prime positive integer relative prime to a, then*

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Fermat's little theorem, which Fermat discovered 100 years before Euler, in 1636, and which he published without proof in 1640, follows very simply from the theorem. Fermat's little theorem [1] is a special case of the Euler-Fermat

[1] Fermat conjectured that if $3 \leq n \in \mathbb{N}$, the equation $x^n + y^n = z^n$ does not have a solution consisting only positive integers. This conjecture was open for centuries. It was finally proved by Andrew Wiles in 1994 using deep number theoretical tools. Since then, this theorem has been called the Fermat's Last Theorem.

theorem, when the modulus m is a prime number. The following two forms of the theorem are known:

Theorem 3.6. (Fermat's little theorem) *If p is prime and $(a,p)=1$ for the integer a, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Theorem 3.7. (Fermat's little theorem) *If p is prime then for all integer a we have:*

$$a^{p} \equiv a \pmod{p}.$$

The order is an important concept in number theory:

Definition 3.8. *Let m be a natural number and a be an integer for which $(a,m)=1$. The order of a modulo m is the smallest positive integer r for which*

$$a^{r} \equiv 1 \pmod{m}.$$

Notation: $o_m(a)$.

The basic properties of the order are the following:

Theorem 3.9. *Let m be a natural number, a be an integer for which $(a,m)=1$, and x,y be also natural numbers, then if*

$$a^{x} \equiv a^{y} \pmod{m},$$

we have

$$x \equiv y \pmod{o_m(a)}.$$

The consequence of this is the following (taking $y=0$):

Theorem 3.10. *Let m and x be natural numbers, a an integer for which $(a,m)=1$. Then*

$$a^{x} \equiv 1 \pmod{m},$$

holds if and only if

$$o_m(a) \mid x.$$

Due to this and the Euler-Fermat theorem:

Corollary 3.11. *Let m be a natural number, $(a, m) = 1$ integer, then*

$$o_m(a) \mid \varphi(m),$$

The primitive root is also an important notion, its definition is the following:

Definition 3.12. *Let m be a natural number, g an integer, then g is a primitive root modulo m, if*

$$o_m(g) = \varphi(m).$$

The following two theorems are frequently used in connection with primitive roots:

Theorem 3.13. *The integer g is a primitive root modulo m if and only if the set $1, g, g^2, \ldots, g^{\varphi(m)-1}$ gives only those mod m residue classes that are relative primes to m exactly once each.*

Theorem 3.14. *For the modulus m, there exists a primitive root if and only if $m = 2, 4$ or m is of the form $m = p^\alpha, 2p^\alpha$, where p is an odd prime, and α is a natural number.*

Finally, we describe one of the oldest number theory theorems, the more than 2000-year-old Chinese remainder theorem, whose simplest form is as follows:

Theorem 3.15. (Chinese remainder theorem) *If $k \in \mathbb{N}$ and $m_1, \ldots, m_k \in$*

$\mathbb{N}$ *are pairwise relative primes, then the simultaneous congruence system*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

is solvable, and the solutions form a unique residue class modulo $\bmod m_1 \dots m_k$.

References

[1] G. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 2008, sixth edition.

3.2 Legendre symbol

Exercise: Prove that there is no square number whose remainder is 2 modulo 3.

Solution: Let x^2 be the square number in question. We study 3 different cases according to the remainder of the integer x modulo 3:

$$\begin{aligned} x \equiv 0 \pmod 3 &\Rightarrow x^2 \equiv 0^2 = 0 \pmod 3 \\ x \equiv 1 \pmod 3 &\Rightarrow x^2 \equiv 1^2 = 1 \pmod 3 \\ x \equiv 2 \pmod 3 &\Rightarrow x^2 \equiv 2^2 = 4 \equiv 1 \pmod 3 \end{aligned}$$

That is, square numbers are congruent to 0 or 1 modulo 3. What about larger primes?

Let $p = 11$. Then
$$\begin{aligned} x \equiv 0 \pmod{11} &\Rightarrow x^2 \equiv 0^2 = 0 \pmod{11} \\ x \equiv 1 \pmod{11} &\Rightarrow x^2 \equiv 1^2 = 1 \pmod{11} \end{aligned}$$

$$\begin{aligned}
x \equiv 2 \pmod{11} &\Rightarrow x^2 \equiv 2^2 = 4 \pmod{11}\\
x \equiv 3 \pmod{11} &\Rightarrow x^2 \equiv 3^2 = 9 \pmod{11}\\
x \equiv 4 \pmod{11} &\Rightarrow x^2 \equiv 16 \equiv 5 \pmod{11}\\
x \equiv 5 \pmod{11} &\Rightarrow x^2 \equiv 25 \equiv 3 \pmod{11}\\
x \equiv 6 \pmod{11} &\Rightarrow x^2 \equiv 36 \equiv 3 \pmod{11}\\
x \equiv 7 \pmod{11} &\Rightarrow x^2 \equiv 49 \equiv 5 \pmod{11}\\
x \equiv 8 \pmod{11} &\Rightarrow x^2 \equiv 64 \equiv 9 \pmod{11}\\
x \equiv 9 \pmod{11} &\Rightarrow x^2 \equiv 81 \equiv 4 \pmod{11}\\
x \equiv 10 \pmod{11} &\Rightarrow x^2 \equiv 100 \equiv 1 \pmod{11}
\end{aligned}$$

That is x^2 may take the values $0, 1, 3, 4, 5$ and 9 modulo 11.

Then $1, 3, 4, 5$ and 9 are **quadratic residues** modulo 11.

While $2, 6, 7, 8$ and 10 are **quadratic non-residues** modulo 11.

More generally, let p be a prime and $(a, p) = 1$. Then a **is a quadratic residue modulo** p if the congruence

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

is solvable and a **is a quadratic non-residue modulo** p if the congruence

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

is not solvable.

Let's continue to assume that $(a, p) = 1$. The **Legendre symbol** $\left(\frac{a}{p}\right)$ is defined as follows:

$$\left(\frac{a}{p}\right) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } a \text{ is quadratic residue} \quad \Leftrightarrow \quad x^2 \equiv a \pmod{p} \\ & \text{modulo } p \qquad\qquad\qquad\qquad\;\; \text{is solvable} \\ & \\ -1 & \text{if } a \text{ is quadratic non-residue} \quad \Leftrightarrow \quad x^2 \equiv a \pmod{p} \\ & \text{modulo } p \qquad\qquad\qquad\qquad\;\; \text{is not solvable} \end{cases}$$

We illustrate our results for the case $p = 11$ with a table:

	1	2	3	4	5	6	7	8	9	10
Quadratic residues modulo 11?	yes	no	yes	yes	yes	no	no	no	yes	no

This distribution seems random...

Based on the concept of quadratic residues, we can define **pseudorandom binary sequences:**

	1	2	3	4	5	6	7	8	9	10
Quadratic residues modulo 11?	1	0	1	1	1	0	0	0	1	0

The number of quadratic residues $\frac{p-1}{2}$: Consider the following numbers $1^2, 2^2, 3^2, \ldots, (p-1)^2$ modulo p. In this sequence

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

holds if and only if

$$\begin{aligned}
&p \mid x^2 - y^2\\
&p \mid (x-y)(x+y)\\
&p \mid x-y \text{ or } p \mid x+y\\
&x \equiv \pm y \pmod{p}\\
&x = y \text{ or } p-y
\end{aligned}$$

That is, the sequence $1^2, 2^2, 3^2, \ldots, (p-1)^2$ contains $\frac{p-1}{2}$ different elements modulo p. **That is, the number of quadratic residues is $\frac{p-1}{2}$.** Hence **the number of quadratic non-residues is $\frac{p-1}{2}$**. According to the older concept, 0 does not belong to any of these sets, now we also follow this terminology. However, we note that 0 is also defined as a quadratic residue more recently.

This simple fact was the motivation of the pseudorandom constructions based on the Legendre symbol.

The value of the Legendre symbol can be quickly calculated (using its extension: the Jacobi symbol.)

Theorem 3.16. *The basic properties of Legendre symbol:*

(a) $(a,p) = (b,p) = 1,\ a \equiv b \pmod{p} \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

(b) $(a,p) = 1 \Rightarrow \left(\frac{a^2}{p}\right) = 1$, *spec.:* $\left(\frac{1}{p}\right) = 1$.

(c) $(a,p) = (b,p) = 1 \Rightarrow \left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$.

(d) *Euler-lemma:*

In case of $(a,p) = 1$ we have

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

(e) $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = \begin{cases} +1, & \textit{if } p \text{ is a prime of the form } p = 4k+1 \\ -1, & textif\, p \text{ is a prime of the form } p = 4k+3. \end{cases}$

(f) $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1, & \textit{if } p \text{ is a prime of the form } p = 8k \pm 1 \\ -1, & \textit{if } p \text{ is a prime of the form } p = 8k \pm 3. \end{cases}$

(g) *Gauss's law of quadratic reciprocity: If p, q are odd primes, then*

$$\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}} \left(\frac{q}{p}\right).$$

Based on the basic properties, the Legendre symbol can be easily calculated. Let's see an example of this:

$$\left(\frac{12345}{331}\right) = \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{823}{331}\right)$$

$$\begin{aligned}
&= \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{161}{331}\right)\\
&= \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{7}{331}\right)\left(\frac{23}{331}\right)\\
&= (-1)\left(\frac{331}{3}\right)\left(\frac{331}{5}\right)(-1)\left(\frac{331}{7}\right)(-1)\left(\frac{331}{23}\right)\\
&= -\left(\frac{1}{3}\right)\left(\frac{1}{5}\right)\left(\frac{2}{7}\right)\left(\frac{9}{23}\right)\\
&= -\left(\frac{1}{3}\right)\left(\frac{1}{5}\right)\left(\frac{2}{7}\right)\left(\frac{3^2}{23}\right)\\
&= -1\cdot 1\cdot 1\cdot 1\\
&= -1.
\end{aligned}$$

The number of steps in this algorithm is $O(\log p)$, yes, but you have to factorize during some steps, which is very time-consuming. Problem: Factorization is very time-consuming!

A quick note before we go any further. The big ordo notation from Edmund Landau is used in the following sense: $f(x) = O(g(x))$ means that $|f(x)|\ leqCg(x)$ for a real constant C in the places of the domain in question. An equivalent notation is: $f(x) \ll g(x)$. If $f(x)/g(x) \to 0$ also holds, then we use the notation $f(x) = o(g(x))$ (this is the little ordo notation).

Important question: How to determine $\left(\frac{a}{p}\right)$ quickly, preferably without factorization steps?

For this we introduce the so-called Jacobi symbol.

Definition 3.17. *If $n \in \mathbb{N}$, $n > 1$, n is <u>odd</u>, $a \in \mathbb{Z}$ $(a, n) = 1$, then the definition of the Jacobí symbol $\left(\frac{a}{n}\right)$ can be given by the factorization of n which is $n = {p_1}^{\alpha_1} \dots {p_r}^{\alpha_r}$. Then*

$$\left(\frac{a}{n}\right) \overset{\text{def}}{=} \left(\frac{a}{p_1}\right)^{\alpha_1} \dots \left(\frac{a}{p_r}\right)^{\alpha_r}.$$

In the above definition, the Jacobi symbol $\left(\frac{a}{n}\right)$ to be defined is on the left, while on the right there is the product of Legendre symbols.

Caution!!! If n is a composite number, then $\left(\dfrac{a}{n}\right)$ has no connection with the solvability of $x^2 \equiv a \pmod{n}$ (unlike the Legendre symbol). But if n is an odd prime, then the definitions of the Legendre and Jacobi symbols coincide.

The Jacobi symbol is fully multiplicative, similarly to the Legendre symbol, and the theorem concerning $\left(\frac{-1}{n}\right)$, $\left(\frac{2}{n}\right)$ and Gauss's quadratic reciprocity theorem can be transferred to the Jacobi symbol.

Theorem 3.18. *If $n \in \mathbb{N}$ is odd, then*

a)

$$\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2} = \begin{cases} 1, & \text{if } n \text{ is of the form} n = 4k+1, \\ -1, & \text{if } n \text{ is of the form} n = 4k+3. \end{cases}$$

b)

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8} = \begin{cases} 1, & \text{if } n \text{ is of the form} n = 8k \pm 1, \\ -1, & \text{if } n \text{ is of the form} n = 8k \pm 3. \end{cases}$$

Theorem 3.19. *If $m, n \in \mathbb{N}$ are odd numbers and $(m,n) = 1$, then*

$$\left(\frac{m}{n}\right) = (-1)^{\frac{m-1}{2}\cdot\frac{n-1}{2}} \left(\frac{n}{m}\right).$$

Proof of Theorem 3.18. Let $n = {p_1}^{\alpha_1} \dots {p_r}^{\alpha_r}$. Then

$$\begin{aligned} \left(\frac{-1}{n}\right) &= \left(\frac{-1}{p_1}\right)^{\alpha_1} \dots \left(\frac{-1}{p_r}\right)^{\alpha_r} \\ &= \left((-1)^{\frac{p_1-1}{2}}\right)^{\alpha_1} \dots \left((-1)^{\frac{p_r-1}{2}}\right)^{\alpha_r} \\ &= (-1)^{\alpha_1 \cdot \frac{p_1-1}{2} + \dots + \alpha_r \frac{p_r-1}{2}}. \end{aligned}$$

This is exactly $+1$ if $\sum_{p_i \equiv 3 \pmod 4} \alpha_i$ is even, which is equivalent to n being a natural number of the form $4k+1$. Moreover,

$$\begin{aligned}\left(\frac{2}{n}\right) &= \left(\frac{2}{p_1}\right)^{\alpha_1} \cdots \left(\frac{2}{p_r}\right)^{\alpha_r} \\ &= \left((-1)^{\frac{{p_1}^2-1}{8}}\right)^{\alpha_1} \cdots \left((-1)^{\frac{{p_r}^2-1}{8}}\right)^{\alpha_r} \\ &= (-1)^{\alpha_1 \cdot \frac{{p_1}^2-1}{8} + \ldots + \alpha_r \frac{{p_r}^2-1}{8}}.\end{aligned}$$

Whether this expression is -1 or $+1$ depends on whether $\sum \alpha_i \frac{{p_i}^2-1}{8}$ is even or odd. If we prove that

$$\sum \alpha_i \frac{{p_i}^2-1}{8} \equiv \frac{n^2-1}{8} \pmod 2, \tag{3.1}$$

then we complete the proof of the theorem. Then

$$\frac{p^2-1}{8} \equiv \begin{cases} 1 \pmod 2, & \text{if } p \equiv \pm 3 \ \ (8), \\ 0 \pmod 2, & \text{if } p \equiv \pm 1 \ \ (8). \end{cases}$$

That is

$$\begin{aligned}\sum \alpha_i \frac{{p_i}^2-1}{8} &\equiv \sum_{\alpha_i \text{ odd}} \frac{{p_i}^2-1}{8} \pmod 2, \\ &\equiv \sum_{p_i \equiv \pm 3 \ (8), \ \alpha_i \text{ odd}} 1 \pmod 2. \end{aligned} \tag{3.2}$$

On the other hand

$$\frac{n^2-1}{8} = \frac{{p_1}^{2\alpha_i} \ldots {p_r}^{2\alpha_r} - 1}{8}.$$

Then we have to examine the remainder of $p_1^{2\alpha_1} \cdots p_r^{2\alpha_r}$ modulo 16. We know:

$$p^2 \equiv \begin{cases} 1 \pmod{16}, & \text{if } p \equiv \pm 1 \ \ (8), \\ 9 \pmod{16}, & \text{if } p \equiv \pm 3 \ \ (8), \end{cases}$$

thus

$$ {p_i}^{2\alpha_i} \equiv \begin{cases} 1 \pmod{16}, & \text{if } \alpha_i \text{ is even or } p \equiv \pm 1 \ \ (8), \\ 9 \pmod{16} & \text{otherwise.} \end{cases}$$

$$p_1{}^{2\alpha_i} \dots p_r{}^{2\alpha_r} \equiv 9^{\sum\limits_{p_i \equiv \pm 3\ (8),\ \alpha_i \text{ odd}} 1}$$

$$\equiv \begin{cases} 1, & \text{if } \sum\limits_{p_i \equiv \pm 3\ (8),\ \alpha_i \text{ is odd}} 1 \text{ is even,} \\ 9, & \text{if } \sum\limits_{p_i \equiv \pm 3\ (8),\ \alpha_i \text{ is odd}} 1 \text{ is odd.} \end{cases} \pmod{16}$$

Based on these:

$$\frac{p_1{}^{2\alpha_1} \dots p_r{}^{2\alpha_r} - 1}{8} = \begin{cases} \text{is even,} & \text{if } \sum\limits_{p_i \equiv \pm 3\ (8),\ \alpha_i \text{ is odd}} 1 \text{ is even,} \\ \text{is odd,} & \text{if } \sum\limits_{p_i \equiv \pm 3\ (8),\ \alpha_i \text{ is odd}} 1 \text{ is odd.} \end{cases}$$

Comparing this with (3.2), we get (3.1), and the theorem follows from this.

Proof of Theorem 3.19. Let $n = p_1 p_2 \dots p_r$, where now among the primes p_i certains can be identical. Furthermore, let $m = q_1 q_2 \dots q_s$, where now among the primes q_i certains can be identical. It is important that $p_i \neq q_j$. By the multiplicity of the Jacobi symbol:

$$\left(\frac{m}{n}\right) = \prod_{1 \le i \le r,\ 1 \le j \le s} \left(\frac{q_j}{p_i}\right), \ \left(\frac{n}{m}\right) = \prod_{1 \le i \le r,\ 1 \le j \le s} \left(\frac{p_i}{q_j}\right).$$

Let among the primes p_i be u pieces of form $4k+3$ and among the primes q_j be v pieces of form $4k+3$. Based on Gauss's quadratic reciprocity theorem, there are uv pieces of pairs $p_i, q_j$is then satisfied for p_i, q_j pairs that for which

$$\left(\frac{q_j}{p_i}\right) = -\left(\frac{p_i}{q_j}\right),$$

and the other pairs have the same Legendre symbol on the left and right hand-sides of the congruence. That is:

$$\left(\frac{m}{n}\right) = (-1)^{uv} \left(\frac{n}{m}\right).$$

However, here uv is odd exactly if u and v are also odd, which is equivalent to m and n having the form $4k+3$. This completes the proof of the theorem.

Example. Is solvable the following congruence

$$x^2 \equiv 7411 \pmod{9283}?$$

Check if 9283 is prime. Since it is, the definitions of Legendre and Jacobi symbol are the same. Henceforth, we can calculate with the Jacobi symbol. This is the following:

$$\begin{aligned}\left(\frac{7411}{9283}\right) &= (-1)^{\frac{7411-1}{2}\cdot\frac{9283-1}{2}}\left(\frac{9283}{7411}\right)\\ &= -\left(\frac{1872}{7411}\right) = -\left(\frac{2^4\cdot 117}{7411}\right)\\ &= -\left(\frac{2}{7411}\right)^4\left(\frac{117}{7411}\right)\end{aligned}$$

$\left(\frac{2}{117}\right)^4 = 1$ therefore remains the negative sign

$$\begin{aligned}&= -(-1)^{\frac{117-1}{2}\cdot\frac{7411-1}{2}}\left(\frac{7411}{117}\right)\\ &= -\left(\frac{40}{117}\right) = -\left(\frac{2}{117}\right)^3\left(\frac{5}{117}\right)\end{aligned}$$

since $\left(\frac{2}{117}\right) = -1$ we also have

$$= (-1)^{\frac{5-1}{2}\cdot\frac{117-1}{2}}\left(\frac{117}{5}\right) = +\left(\frac{2}{5}\right) = -1.$$

Based on the above, the congruence is not solvable.

References

[1] G. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 2008, sixth edition.

3.3 A few words about continued fractions

Let x be a real number. Then we would like to write x in the following form

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}, \tag{3.3}$$

where $a_0 \in \mathbb{Z}$, $a_1, a_2, \dots \in \mathbb{Z}^+$ and $a_i \geq 1$ if $i \geq 1$. This will be the continued fraction form of x. It is not obvious from the definition that all real numbers can be written as a continued fraction, but we will prove this after Theorem 3.22.. Instead of the above double fraction, the more space-saving notation $x = [a_0; a_1, a_2, \dots]$ is often used, but we will stick to the notation for double fractions for the sake of easier transparency.

By $a_1 \geq 1$ we get

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}} > 1,$$

thus

$$\cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}} < 1. \tag{3.4}$$

By (3.3), (3.4) and since a_0 is an integer we get:

$$a_0 = [x]$$

and

$$\cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}} = \{x\}.$$

Thus:

$$x = a_0 + x_0,$$

where

$$a_0 = [x], \qquad x_0 = \{x\} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}.$$

Then

$$\frac{1}{x_0} = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}.$$

The above algorithm can be continued in the same way. We show you one more step:

$$\frac{1}{x_0} = a_1 + x_1,$$

where

$$a_1 = \left[\frac{1}{x_0}\right], \qquad x_1 = \left\{\frac{1}{x_0}\right\} = \frac{1}{x_0} - a_1.$$

Our calculations show that the continued fraction digits a_i can also be given by an algorithm. This is as follows:

$$a_0 = [x], \qquad x_0 = \{x\},$$

and if $i \geq 1$, then

$$a_i = \left[\frac{1}{x_{i-1}}\right], \qquad x_i = \left\{\frac{1}{x_{i-1}}\right\} = \frac{1}{x_{i-1}} - a_i.$$

Now

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \dots \cfrac{1}{a_i + x_i}}}.$$

Furthermore, for the a_i's defined by this way, the relation (3.3) indeed holds. A small clarification is still necessary here. If $\frac{1}{x_{i-1}}$ is an integer, then the algorithm either stops, i.e., $a_i = \frac{1}{x_{i-1}}$, or stops at the next step, and $a_i = \frac{1}{x_{i-1}} - 1$, $a_{i+1} = 1$. This shows that rational numbers have two different forms of continued fractions. (We will return to this later.)

The following are two simple statements without proof. The first is that the above algorithm exactly ends in a finitely many steps if x is rational. Our second statement is Lagrange's theorem, which states:

Theorem 3.20. (Lagrange theorem) *If x is the root of an integer quadratic equation, it has a continued fraction form with periodic digits, and vice versa, if x has a continued fraction form with periodic digits, it is a root of a quadratic equation.*

An important definition in the study of continued fractions is:

Definition 3.21. *Let the continued fraction form or x be the following:*

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}.$$

Then we define the i-th convergent by

$$\frac{p_i}{q_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{i-1} + \cfrac{1}{a_i}}}}.$$

Our first theorem on convergents is the following:

Theorem 3.22. *Write x in a continued fraction form:*

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}$$

Now we define the numbers p_i and q_i by

$$\begin{aligned}
&p_0 = a_0, \quad q_0 = 1,\\
&p_1 = a_0 a_1 + 1, \quad q_1 = a_1,\\
&p_i = a_i p_{i-1} + p_{i-2}, \quad q_i = a_i q_{i-1} + q_{i-2}, \ \textit{if } i \geq 2.
\end{aligned}$$

The following is true in this case:

(a)

$$\frac{p_0}{q_0} = \frac{a_0}{1},$$

$$\frac{p_1}{q_1} = a_0 + \frac{1}{a_1},$$

$$\frac{p_i}{q_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{i-1} + \cfrac{1}{a_i}}}}.$$

(b) $p_i q_{i-1} - p_{i-1} q_i = (-1)^{i-1}$, *if* $i \geq 1$.

(c) $(p_i, q_i) = 1$.

In part (a) of the theorem we define the integers p_i, q_i a little differently, as in Definition 3.21.. However, part (c) shows that the two definitions coincide. It is further noted that parts (a) and (b) of the theorem also holds for any numbers a_i, it is not required that a_i's are integer. This fact will be used later in this chapter.

Proof of Theorem 3.22. Part (a) of the theorem is verified by induction for i. For $i = 0$ and $i = 1$ we have

$$\frac{p_0}{q_0} = \frac{a_0}{1} \quad \text{and} \quad \frac{p_1}{q_1} = \frac{a_0 a_1 + 1}{a_1} = a_0 + \frac{1}{a_1},$$

so the statement is obvious. Then we move on to the induction step: Suppose we have proved the statement for $i = k$. Then we will also prove that for $i = k + 1$, that is,

$$\frac{p_{k+1}}{q_{k+1}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_k + \cfrac{1}{a_{k+1}}}}}.$$

In the continued fraction above, there are $k + 1$ pieces of digits, but with a clever notation it can also be written using k pieces of digits. Let

$$a'_k = a_k + \frac{1}{a_{k+1}}.$$

Clearly a'_k is not an integer, but as mentioned this is not necessary to assume in part (a). Then

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_k + \cfrac{1}{a_{k+1}}}}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{k-1} + \cfrac{1}{a'_k}}}}.$$

However, on the right side of the equation, there are only k digits. Since we assumed that the statement is true for k pieces of digits, we have

$$\frac{p_0}{q_0} = \frac{a_1}{1},\ \frac{p_1}{q_1} = a_0 + \frac{1}{a_1}, \dots,\ \frac{p_{k-1}}{q_{k-1}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{k-2} + \cfrac{1}{a_{k-1}}}}}.$$

Then the k-th digit is a'_k (and not a_k), thus the k-th convergent is $\frac{p'_k}{q'_k}$. By the induction hypothesis we have

$$p'_k = a'_k p_{k-1} + p_{k-2},\ q'_k = a'_k q_{k-1} + q_{k-2}.$$

Then

$$\begin{aligned}
\frac{p'_k}{q'_k} &= \frac{a_k{}' \, p_{k-1} + p_{k-2}}{a_k{}' \, q_{k-1} + q_{k-2}} \\
&= \frac{\left(a_{k-1} + \frac{1}{a_k}\right) p_{k-1} + p_{k-2}}{\left(a_{k-1} + \frac{1}{a_k}\right) q_{k-1} + q_{k-2}} \\
&= \frac{(a_{k-1}a_k + 1)p_{k-1} + a_k p_{k-2}}{(a_{k-1}a_k + 1)q_{k-1} + a_k q_{k-2}} \\
&= \frac{a_k(a_{k-1}p_{k-1} + p_{k-2}) + p_{k-1}}{a_k(a_{k-1}q_{k-1} + q_{k-2}) + q_{k-1}} \\
&= \frac{a_k p_k + p_{k-1}}{a_k q_k + q_{k-1}} = \frac{p_{k+1}}{q_{k+1}},
\end{aligned}$$

which was to be proved.

Part (b) of the theorem can also be proved by induction. Let $i = 1$, then

$$p_i q_{i-1} - p_{i-1} q_i = p_1 q_0 - p_0 q_1 = (a_0 a_1 + 1) \cdot 1 - a_0 a_1 = 1.$$

We can then proceed to the induction step. Suppose that we have proved the statement for $i = k$. Let's see the proof for $i = k + 1$. To do this:

$$\begin{aligned} p_{k+1} q_k - p_k q_{k+1} &= (a_k p_k + p_{k-1}) q_k - p_k (a q_k + q_{k-1}) \\ &= p_{k-1} q_k - p_k q_{k-1} = -(-1)^{k-1} = (-1)^k. \end{aligned}$$

Finally, we also prove part (c) of the theorem: Let $d \overset{\text{def}}{=} (p_i, q_i)$,

$$d \mid \underbrace{p_i q_{i-1}}_{d|} - \underbrace{p_{i-1} q_i}_{d|} = (-1)^{i-1}$$

$$d \mid 1 \;\Rightarrow\; d = 1.$$

This completes the proof.

Dividing the equation in part (b) of the theorem by $q_{i-1} q_i$ we get:

$$\frac{p_i}{q_i} - \frac{p_{i-1}}{q_{i-1}} = \frac{(-1)^{i-1}}{q_i q_{i-1}}, \tag{3.5}$$

By definition

$$\frac{p_{i+1}}{q_{i+1}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_i + \cfrac{1}{a_{i+1}}}}}.$$

If a_{i+1} is replaced by $\frac{1}{x_i}$, the value of this fraction is x, thus:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_i + \cfrac{1}{\cfrac{1}{x_i}}}}} \tag{3.6}$$

When applying Theorem 3.22. part (a), it is not necessary to suppose that the digits are integers, so applying part (a) to (3.6) we get:

$$x = \frac{\frac{1}{x_i}p_i + p_{i-1}}{\frac{1}{x_i}q_i + q_{i-1}} = \frac{p_i + x_i p_{i-1}}{q_i + x_i q_{i-1}}$$

A simple calculation shows that the above fraction is between $\frac{p_i}{q_i}$ and $\frac{p_{i-1}}{q_{i-1}}$. The convergents tend to x since $\left|\frac{p_i}{q_i} - \frac{p_{i-1}}{q_{i-1}}\right| \longrightarrow 0$ and x is between $\frac{p_i}{q_i}$ and $\frac{p_{i-1}}{q_{i-1}}$. In the following, we will prove two theorems which will be much needed in the later chapters of this lecture notes. Theorem 3.23. will be used in a factorization algorithm, namely for the continued fraction algorithm, while a major attack against RSA was based on Theorem 3.24..

Theorem 3.23. *Let the i-th convergent of $x \geq \frac{1}{2}$ be $\frac{p_i}{q_i}$. Then:*

$$\left|{p_i}^2 - x^2{q_i}^2\right| < 2x.$$

Proof of Theorem 3.23. Then we have

$$\left|p_i^2 - x^2 q_i^2\right| = q_i^2 \left|x - \frac{p_i}{q_i}\right| \left|x + \frac{p_i}{q_i}\right|.$$

Since x is between the convergents $\frac{p_i}{q_i}$ and $\frac{p_{i+1}}{q_{i+1}}$ and by (3.5) we have:

$$\left|x - \frac{p_i}{q_i}\right| \leq \left|\frac{p_{i+1}}{q_{i+1}} - \frac{p_i}{q_i}\right| = \frac{1}{q_i q_{i+1}}.$$

Furthermore

$$\left|x + \frac{p_i}{q_i}\right| = \left|\frac{p_i}{q_i} - x + 2x\right| \leq \left|\frac{p_i}{q_i} - x\right| + 2x \leq 2x + \frac{1}{q_i q_{i+1}}.$$

Thus:

$$\begin{aligned}\left|p_i^2 - x^2 q_i^2\right| &= q_i^2 \left|x - \frac{p_i}{q_i}\right| \cdot \left|x + \frac{p_i}{q_i}\right| \\ &\leq q_i^2 \frac{1}{q_i q_{i+1}} \left(2x + \frac{1}{q_i q_{i+1}}\right) \\ &= 2x \frac{q_i}{q_{i+1}} + \frac{1}{q_{i+1}^2}.\end{aligned}$$

Rearranged:

$$\begin{aligned}\left|{p_i}^2 - x{q_i}^2\right| - 2x &< 2x\left(-1 + \frac{q_i}{q_{i+1}} + \frac{1}{2x{q_{i+1}}^2}\right)\\ &< 2x\left(-1 + \frac{q_i}{q_{i+1}} + \frac{1}{q_{i+1}}\right)\\ &< 2x\left(-1 + \frac{q_{i+1}}{q_{i+1}}\right) = 0.\end{aligned}$$

Thus

$$\left|{p_i}^2 - x{q_i}^2\right| < 2x,$$

which was to be proved.

Theorem 3.24. (Lagrange) *If*

$$\left|\frac{p}{q} - \alpha\right| < \frac{1}{2q^2}, \tag{3.7}$$

where p and q and relatively prime, then $\frac{p}{q}$ is a convergent of α.

Proof of Theorem 3.24.

The proof is based on the following lemma:

Lemma 3.25. *If*

$$x = \frac{P\zeta + R}{Q\zeta + S},$$

where $\zeta > 1$ and for integers P, Q, R, S we have

$$Q > S > 0, \quad PS - QR = \pm 1,$$

then $\frac{R}{S}$ and $\frac{P}{Q}$ are two consecutive convergents of x. If $\frac{R}{S}$ is the $n-1$-th convergent and $\frac{P}{Q}$ is the n-th, then ζ is the $n+1$-th complete quotient, namely

$$\zeta = a_{n+1} + \cfrac{1}{a_{n+2} + \cfrac{1}{\ddots}},$$

where a_i's are the digits of x.

Proof of the Lemma 3.25. Write $\frac{P}{Q}$-t continued fraction form:

$$\frac{P}{Q} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{n-1} + \cfrac{1}{a_n}}}} = \frac{p_n}{q_n}. \tag{3.8}$$

Here we can assume that n is even or odd as we please, since every rational number has exactly two continued fraction forms, in one the number of digits is even, and in the other is odd. This statement is based on the following remark: if $a_k \geq 2$, then:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_{k-1} + \cfrac{1}{a_k}}}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_{k-1} + \cfrac{1}{a_k - 1 + \cfrac{1}{1}}}}}.$$

So in (3.8) we can choose the parity of n such that

$$PS - QR = (-1)^{n-1}$$

holds. Then $(P, Q) = 1$, $Q > 0$ and $(p_n, q_n) = 1$. Thus by (3.8) we have $P = p_n$, $Q = q_n$. That is

$$p_n S - q_n R = PS - QR = (-1)^{n-1} = p_n q_{n-1} - p_{n-1} q_n.$$

Rearranged

$$p_n(S - q_{n-1}) = q_n(R - p_{n-1}).$$

By $(p_n, q_n) = 1$ we have

$$q_n \mid S - q_{n-1}. \tag{3.9}$$

But

$$q_n = Q > S > 0, \;\; q_n \geq q_{n-1} > 0,$$

and thus

$$|S - q_{n-1}| < q_n.$$

But by (3.9), this is possible only if $S - q_{n-1} = 0$. So

$$S = q_{n-1}, \quad R = p_{n-1}.$$

In summary, so far

$$x = \frac{p_n\zeta + p_{n-1}}{q_n\zeta + q_{n-1}}.$$

Now consider that continued fraction whose first n digits are identical with the first n digits of x, but the $n+1$-th digit is " ζ ", which is not an integer, but as we said in parts of (a) and (b) of Theorem 3.22. it is not needed. By Theorem 3.22., for the $n+1$-th convergent $\frac{p'_n}{q'_n}$ we have

$$\begin{aligned} p'_n &= p_n\zeta + p_{n-1}, \\ q'_n &= q_n\zeta + q_{n-1}. \end{aligned}$$

Write ζ in continued fraction form, where the digits are $a_{n+1}, a_{n+2}, \ldots$, that is

$$\zeta = a_{n+1} + \cfrac{1}{a_{n+2} + \cfrac{1}{\ddots}}.$$

By the conditions of the theorem we have $a_{n+1} = [\zeta] \geq 1$, so the continued fraction form of x is indeed

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}.$$

As a result, all statements of the lemma are proved.

Let's go back to the proof of Theorem 3.24.. Let

$$\frac{p}{q} - \alpha = \frac{\varepsilon\theta}{q^2},$$

where by (3.7) we may assume

$$\varepsilon = \pm 1, \quad 0 < \theta < \frac{1}{2}.$$

Write $\frac{p}{q}$ in continued fraction form

$$\frac{p}{q} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots \cfrac{1}{a_{n-1} + \cfrac{1}{a_n}}}} = \frac{p_n}{q_n}, \tag{3.10}$$

where we can choose the parity of n as we please. Let n be such that

$$\varepsilon = (-1)^{n-1}.$$

Define ζ by

$$\alpha = \frac{\zeta p_n + p_{n-1}}{\zeta q_n + q_{n-1}},$$

where p_n/q_n and p_{n-1}/q_{n-1} are the last and the last but one convergents to the continued fraction form of $\frac{p}{q}$ in (3.10). Then

$$\frac{\varepsilon\theta}{q_n^2} = \frac{p_n}{q_n} - \alpha = \frac{p_n q_{n-1} - p_{n-1} q_n}{q_n\left(\zeta q_n + q_{n-1}\right)} = \frac{(-1)^{n-1}}{q_n\left(\zeta q_n + q_{n-1}\right)},$$

thus

$$\theta = \frac{q_n}{\zeta q_n + q_{n.1}}.$$

Since $0 < \theta < \frac{1}{2}$

$$\zeta = \frac{1}{\theta} - \frac{q_{n-1}}{q_n} > 1.$$

By Lemma 3.25. the rationals $\frac{p_{n-1}}{q_{n-1}}$ are $\frac{p_n}{q_n}$ two consecutive convergents in the continued fraction form of α. Since $\frac{p}{q} = \frac{p_n}{q_n}$, the theorem is proved.

References

[1] G. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 2008, sixth edition.

4 Pseudorandomness

Pseudorandom sequences have many applications. It can be used to simulate natural phenomena, economic processes, or as a key in various encryption algorithms. Pseudorandom sequences are used in mathematics and physics, particularly in cryptography and numerical analysis.

In Chapter 2.5, for example, we saw a very important application of pseudorandom sequences, namely the Vernam cipher. The Vernam cipher is a well-known and widely used encryption algorithm that is still in use today. It is worthwhile to read a tutorial on the Vernam cipher to understand the significance of pseudorandom generation (e.g., Chapter 2.5 of this lecture notes).

Without claiming to be complete, we will study random and pseudorandom generation in this chapter.

But, exactly, what is random number generation? From ancient times to the present, random generation has always played an important role. It is questionable whether the methods we use produce truly random numbers. Many people believe that only physical methods can generate real random numbers, but are these approaches truly appropriate?

In this chapter, I study when a 0-1 sequence of a given length (e.g.,

1100100110)

can be characterized as pseudorandom. Obviously, the sequences

1111111111

or

1010101010

are not considered pseudorandom. They are both "too regular," with the former containing just ones and the latter alternating between ones and zeros. Pseudorandom number generators use mathematical formulas to generate a seemingly random sequence of numbers.

References

[1] G. S. Vernam, *Cipher printing telegraph systems for secret wire and radio telegraphic communications*, Transactions of the American Institute of Electrical Engineers 55 (1926), 109–115.

[2] photo, http://dibujosa.com/dibujosgratisapp.php?codigo=20886

4.1 Pseudorandomness - History

Many articles on the subject of pseudorandomness have been published in the previous 80 years, covering a wide range of goals, techniques, and mathematical methods. Perhaps a classic book should be highlighted here among many, D. Knuth, in volume 2 (Seminumerical Algorithms) of his book The Art of Computer Programming [3] devotes an entire chapter to generating pseudorandom sequences.

First, the concept of pseudorandomness was defined in terms of complexity theory. Goldwasser [2] has written an excellent review paper on this topic.

Definition 4.1. *A random bit generator is a device or algorithm that generates bits that are statistically independent and unbiased.*

Historically, hardware-based generators (e.g., diodes) were used, but software-based generators (machine time, memory size, and so on) were also available; when a method is used to construct a sequence of bits, and this sequence must then be evaluated with certain statistical tests, this testing is known as an a posteriori test.

This is a difficult, time-consuming, and now unsatisfactory technique. As a result, nowadays random bit generators have been substituted by pseudo-random bit generators.

We should remark that among the latter, even today, the most used method in programming is the linear congruence generator, see e.g., [5], but you can read more about this in Knuth's book [3]. In this lecture notes, however, we would rather describe some more modern methods, but first let's see an important definition.

Definition 4.2. *A* pseudorandom bit generator *is a deterministic algorithm that generates a seemingly random binary sequence of length ℓ from a considerably smaller secret sequence of length k (where ℓ is significantly greater than k), producing a pseudorandom binary sequence that appears to be random.*

The input sequence, an arbitrary secret binary sequence of length k, is called as the "seed", and the output sequence, which is much longer, is known as the pseudorandom binary sequence.

What defines a binary sequence as "random-looking" or "good" pseudo-random? Of course, the applications specify which random properties must be regulated.

Unpredictability is a requirement that is frequently used in cryptography:

Definition 4.3. *A pseudo-random generator is said to satisfy the next bit test if no polynomial-time algorithm can predict the $k+1$st element from the* *previous k elements with a considerably higher probability than the* $1/2$.

Although this test is often very important, we note that e.g., in the Monte-Carlo methods this is not required.

Criticism of the next bit test: So that only recursive structures can be classified in this way, but e.g., it is possible that a generator may pass through, but by knowing the initial and end bits, the entire sequence may be uniquely identified. Furthermore, only conditionally justified polynomial-time algorithms exist. Moreover, it is difficult to clarify the meaning of "significantly larger than 1/2" in the definition. What does "significantly larger" than 1/2 mean exactly?

The most important construction that satisfies the next bit test (but only under certain unproven hypothesis):

Definition 4.4. *The Blum-Blum-Shub [1] pseudorandom generator is as follows:*

1. *Consider two large primes p and q of the form $4k+3$, and let $n = pq$.*
2. *Consider a “random” integer a (this is the „seed”) with $0 < a < n$, $(a, n) = 1$. Let x_0 be a^2 modulo n, where $1 \leq x_0 < n$.*
3. *When $x_0, x_1, \ldots, x_k$ are known, then let x_{k+1} be ${x_k}^2$ modulo n, where $1 \leq x_{k+1} < n$.*
4. *Let z_i be the last binary digit of x_i.*

The sequence x_i in the construction will become periodic at some point, denoted by the lowest period length t. Obviously, only the first t elements of the sequence z_i are worth studying for pseudorandomness.

Then:

Theorem 4.5. (Blum, Blum, Shub, [1]) *The Blum-Blum-Shub binary sequence generator* $z_0, z_1, z_2, \ldots, z_{t-1}$ *satisfies the next bit test, assuming the unproven hypothesis that there is no polynomial-time test to factorise the modulus* n *in the construction.*

We will not prove it. The proof can be found in [1].

The complexity theory approach has recently heavily criticised. Namely:

1. This approach simply qualifies generators, however it is not possible to avoid a posteriori testing of each sequence using it.
2. One of the fundamental standard definitions, known as the "next bit test", allows testing purely based on unproven hypotheses.
3. It is typically used only for infinitely long sequences of numbers, however in practise, only finite long sequences are used.

References

[1] L. Blum, M. Blum, M.Shub, *A simple unpredictable pseudo-random number generator*, SIAM Journal on Computing. Society for Industrial & Applied Mathematics (SIAM), 15 (2) (1986), 364-383.

[2] S. Goldwasser, *Mathematical foundations of modern cryptography: computational complexity perspective*, ICM, 2002, vol. I, 245-272.

[3] D. E. Knuth, *The Art of Computer Programming (Volume 2) – The Seminumerical Algorithms*, Addison-Wesley, 1997.

[4] A. Sárközy, *Computational Number Theory*, university lecture.

[5] Wikipedia, *Linear congruential generator*, https://en.wikipedia.org/wiki/Linear_congruential_generator

4.2 Pseudorandomness - Quantitative approach

In practise, the above definition of pseudorandomness ("next bit test") is not very satisfactory. As a result, beginning in 1997, Mauduit and Sárközy developed a new theory of pseudorandomness that is much more useful in practise. They changed from bit sequences to ± 1 sequences for technical reasons of ease of calculation (since then the expected value of the most commonly used related random variables is 0).

The following new quantitative measures were introduced by Mauduit and Sárközy [7]:

Definition 4.6. *Let $E_N = (e_1, e_2, \ldots, e_N) \in \{-1, +1\}^N$ be a ± 1 sequence of length N. Then the* well-distribution measure *is defined by*

$$W(E_N) = \max_{a,b,t} \left| \sum_{j=0}^{t-1} e_{a+jb} \right|,$$

where the maximum is taken over all a, b, t such that $a, b, t \in \mathbb{N}$ and $1 \leq a \leq a + (t-1)b \leq N$. The correlation measure of order k *is defined by*

$$C_k(E_N) = \max_{M,D} \left| \sum_{n=1}^{M} e_{n+d_1} e_{n+d_2} \ldots e_{n+d_k} \right|,$$

where the maximum is taken over all $M, D = (d_1, d_2, \ldots, d_k)$ such that $1 \leq d_1 < d_2 < \cdots < d_k \leq M + d_k \leq N$.

Cassaigne, Mauduit and Sárközy [1] proved that the values of these measures are less than $cN^{1/2}(\log N)^{1/2}$ for almost all sequences of length N (i.e., for $(1-\varepsilon)2^N$ pieces of the all 2^N sequences). Thus, a sequence $E_N = (e_1, e_2, \ldots, e_N)$ is said to have strong pseudorandom properties if positive constants c_1 and c_k exist such that

$$W(E_N) \leq N^{1-c_1},$$
$$C_k(E_N) \leq N^{1-c_k} \text{ at least for small } k\text{'s}.$$

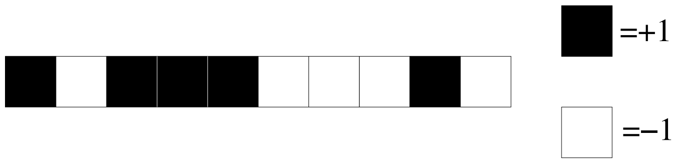
During their research, several authors generated various sequences with very strong pseudorandom properties, i.e.,:

$$W(E_N) \ll \sqrt{N} \log N,$$
$$C_k(E_N) \ll \sqrt{N}(\log N)^k.$$

Mauduit and Sárközy [7] gave the following construction in their first paper in the field, published in 1997:

$$E_{p-1} = \left(\left(\frac{1}{p}\right), \left(\frac{2}{p}\right), \dots, \left(\frac{p-1}{p}\right) \right).$$

For example, if $p = 11$, this sequence can be illustrated as follows:



Mauduit and Sárközy [7] proved:

$$W(E_{p-1}) \ll p^{1/2} \log p \text{ and } C_\ell(E_{p-1}) \ll p^{1/2} \log p.$$

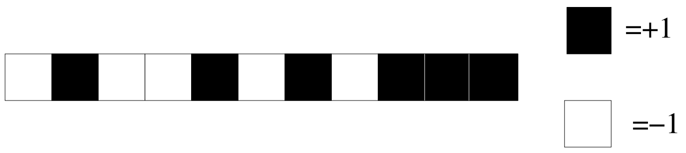
This construction, however, produces just one sequence for each prime p.

Hoffstein and Lieman [5] brilliantly extended this construct to generate many sequences for each prime p at a time, providing a large family of pseudorandom sequences:

$$E_p(f) = \left(\left(\frac{f(1)}{p}\right), \left(\frac{f(2)}{p}\right), \dots, \left(\frac{f(p)}{p}\right) \right) \quad \text{where } \left(\frac{0}{p}\right) \overset{\text{def}}{=} 1 \tag{4.1}$$

and f is a polynomial of degree at most $p - 2$ over $\mathbb{Z}_p$.

For example, if $p = 11$, $f(x) = x^2 + 1$ this sequence can be illustrated as follows:



Hoffstein and Lieman gave their construction based on numerical calculations and conjectured that it has strong pseudorandom properties and satisfies the "next bit test". However, they did not prove the pseudorandom properties of the sequence.

Goubin, Mauduit, and Sárközy [2] proved that the sequence in (4.1) has strong pseudorandom properties (assuming some not too restrictive conditions for the polynomial f):

$$W(E_p(f)),\ C_\ell(E_p(f)) \ll p^{1/2}\log p$$

More exactly,

Theorem 4.7. (Goubin, Mauduit, Sárközy, [2]) *Let p be a prime, $f(x) \in \mathbb{F}_p[x]$ be a polynomial of degree k, which is not of the form $cg(x)^2$, where $c \in \mathbb{F}_p$, $g(x) \in \mathbb{F}_p[x]$. Define $E_p(f) = (e_1, \dots, e_p)$ by:*

$$e_n = \begin{cases} \left(\frac{f(n)}{p}\right) & \text{for } (f(n), p) = 1, \\ +1 & \text{for } p \mid f(n). \end{cases} \tag{4.2}$$

Then

$$W(E_p(f)) \ll kp^{1/2}\log p.$$

Assume that one of the following three conditions for ℓ, which is the order of the correlation, holds true: (i) $\ell = 2$;
(ii) $\ell < p$ and 2 is a primitive root modulo p;
(iii) $(4k)^\ell < p$.
Then:

$$C_\ell(E_p(f)) \ll k\ell p^{1/2}\log p.$$

If $f(x)$ is of the form $f(x) = cg(x)^2$, our sequence, except for the zeros of f, contains only the same element, $\left(\frac{c}{p}\right)$, so the sequence as a sevret key is completely unsuitable in cryptography. Thus, the first condition in the theorem is very important. Fortunately, there are not many such polynomials: out of all k-degree polynomials (numbered p^{k+1} pieces), only $p^{[k/2]+1}$ pieces are of the form $cg(x)^2$...

Numerous additional constructions have developed since then, but in some senses, this is still the best: it has strong pseudorandom properties and the generation of sequences is fast. For the sake of completeness, we present some alternative powerful constructions.

These constructions are quick to generate, have "good" pseudorandom properties in the sense defined, which eliminates the need for a posteriori testing, and have been proved to have good cryptographic properties (not only conditionally, under certain unproven hypothesis).

Theorem 4.8. (Mauduit, Rivat, Sárközy [6]) *Let p be a prime, $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree k and $r_p(n)$ denote the least nonnegative residue of n modulo p. Define $E_p(f) = (e_1, \dots, e_p)$ by:*

$$e_n = \begin{cases} +1, & \text{if } 0 \le r_p(f(n)) < p/2, \\ -1, & \text{if } p/2 \le r_p(f(n)) < p. \end{cases}$$

Then for $2 \le \ell \le k-1$ we have

$$W(E_p(f)) \le kp^{1/2}(\log p)^2$$
$$C_\ell(E_p(f)) \le kp^{1/2}(\log p)^{\ell+1}.$$

The disadvantage of the construction is that correlation of high-order can be large.

Another construction:

Theorem 4.9. (Mauduit, Sárközy [8]) *Let p be a prime, $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree k and $r_p(n)$ denote the least nonnegative residue of n*

modulo p. Define $E_p(f) = (e_1, \dots, e_p)$ *by:*

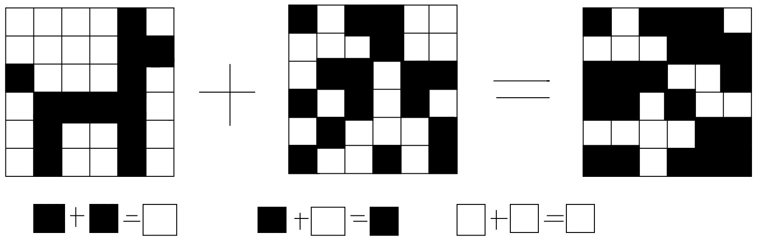
$$e_n = \begin{cases} +1 & \text{if } (f(n), p) = 1 \text{ and } 0 < r_p(f(n)^{-1}) < p/2, \\ -1 & \text{otherwise.} \end{cases}$$

Assume that one of the following three conditions for ℓ*, which is the order of the correlation, holds true: (i)* $\ell = 2$*;*
(ii) $(4k)^\ell < p$*;*
(iii) $k\ell < p/2$ *and* $f(x)$ *can be written of the form:* $f(x) = (x + a_1)(x + a_2)...(x + a_k)$ *where* $a_i \in \mathbb{F}_p$*. Ekkor:*

$$W(E_p(f)) \le kp^{1/2}(\log p)^2$$
$$C_\ell(E_p(f)) \le k\ell p^{1/2}(\log p)^{\ell+1}.$$

It is important to note that Rivat and Sárközy [9] tested numerous sequences in construction in Theorem 4.7. by a posteriori testing. These were the a posteriori tests included in the ″1.4-sts. package″ specified by the American "National Institute of Standards and Technology". Furthermore, Rivat and Sárközy [9] proved that if the pseudo-random measures are small, the sequences "almost" satisfy quite a few of the above tests, and some a posteriori testing can be avoided.

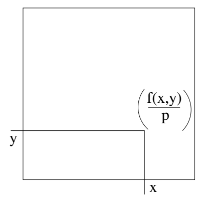
However, if you want to encrypt an image instead of text, you do not need a pseudorandom sequence, but instead a pseudorandom lattice.



My co-authors András Sárközy and Cameron L. Stewart and I [3], [4] constructed binary lattices with strong pseudorandom properties using the Legendre symbol.

Construction 4.10. (Gyarmati, Sárközy, Stewart, [3], [4]) *Let p be a prime and $f(x,y) \in \mathbb{Z}[x,y]$ be a polynomial in two variables. Define $\eta : \{0,1,\dots,p-1\} \times \{0,1,\dots,p-1\} \to \{-1,+1\}$ by*

$$\eta(x,y) = \begin{cases} \left(\frac{f(x,y)}{p}\right) & \text{if } p \nmid f(x,y), \\ +1 & \text{if } p \mid f(x,y). \end{cases}$$



The defined construction has strong multidimensional pseudorandom measures, according to our results [3], [4]. Another advantage of the construction is that the lattice elements can be generated quickly.

So far, the constructs discussed were based on the Legendre symbol and a polynomial f (one or more variables). For someone to be able to programme the sequence or lattice, they only need to know the value of the prime p and the coefficients of the polynomial f.

References

[1] J. Cassaigne, C. Mauduit, A. Sárközy, *On finite pseudorandom binary sequences VII: The measures of pseudorandomness*, Acta Arith. 103 (2) (2002), 97-118.

[2] L. Goubin, C. Mauduit, A. Sárközy, *Construction of large families of pseudorandom binary sequences*, Journal of Number Theory 106 (1) (2004), 56-69

[3] K. Gyarmati, A. Sárközy and C. L. Stewart, *On Legendre symbol lattices*, Unif. Distrib. Theory 4 (2009), 81-95.

[4] K. Gyarmati, A. Sárközy and C. L. Stewart, *On Legendre symbol lattices, II*, Unif. Distrib. Theory 8 (2013), 47-65.

[5] J. Hoffstein, D. Lieman, *The distribution of the quadratic symbol in function fields and a faster mathematical stream cipher*, Progress in Computer Science and Applied Logic, Vol. 20, Birkhäuser, Verlag, Basel, 2001, 59-68.

[6] C. Mauduit, J. Rivat and A. Sárközy, *Construction of pseudorandom binary sequence using additive characters*, Monatshefte Math. 141 (2004), 197-208.

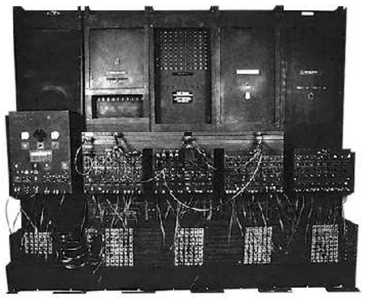
[7] C. Mauduit, A. Sárközy, *On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol*, Acta Arith. 82 (4) (1997), 365-377

[8] C. Mauduit, A. Sárközy, *Construction of pseudorandom binary sequences by using the multiplicative inverse*, Acta Math. Hungar. 108 (2005), 239-252.

[9] J. Rivat and A. Sárközy, *On pseudorandom sequences and their application*, Lecture Notes in Comput. Sci. 4123, General theory of information transfer and combinatorics, Springer, Berlin / Heidelberg, 2006, 343-361.

5 Principles of Neumann

In 1946, the first fully electronic computer, ENIAC, was built.



Based on the experience that János Neumann gained during the construction, he developed the basic principles that were essential for the operations of the computer at that time. Although these principles have been slightly (but only slightly) modified in practice today, they still best illustrate how computers work. The Neumann principles are as follows:

1. Fully electronic operation.
2. Using binary number systems.
3. Use of internal memory.
4. Stored program principle. The machine stores the data and program instructions required for calculations in the same way, both in the internal memory (operational memory).
5. Sequential instruction execution (the instructions should be executed sequentially in time).
6. Universal usability, Turing machine (programmability).
7. Structure: five functional units (arithmetic unit, central control unit, memories, input and output units).

It is important to note here that ENIAC was not the first computer in the world, for example, in 1840, Thomas Fowler produced a wooden computer based on the tenary number system, which has the size $1.8m \times 0.9m \times 0.3m$. The first electronic computer was built by Konrad Zuse around 1943 and called the Z3. Tenary computers were brought back into vogue by D. Knuth.

References

[1] H. H. Goldstine, *A számítógép Pascaltól Neumannig*, Budapest, Műszaki (2003) (in Hungarian).

[2] Kovács Győző, Szelezsán János, *Gondolatok Neumann János First Draft of a Report on the EDVAC című, 1945 júniusában megjelent tanulmányáról*, in: Ki volt igazából Neumann János, collection of papers (in Hungarian), Nemzeti Tankönyvkiadó, 2003..

[3] J. Neumann, *The Computer and the Brain*, Maple Press Compony, 1958.

[4] J. Neumann, *First Draft of a Report on the EDVAC*, University of Pennsylvania 1945.

[5] Szelezsán János, *Neumann János az első, számítógépet alkalmazó »fizikus«*, Fizikai Szemle 2003/12., p. 425 (in Hungarian).

[6] Photo, *ENIAC*, http://www.feltalaloink.hu/tudosok/neumannjanos/html/neujantal1.htm

6 Elementary arithmetic operations

The chapter is based on the books by Das [1] and Koeblitz [3].

Today, the most common computers are 32-bit or 64-bit. But what does it mean for a computer to be 64-bit? Oddly enough, the number 64 here is related to the base of the number system used by computers. A 64-bit computer basically uses B-based numbers where $B = 2^{64}$. This is indeed quite a large number, the base of the number system itself is more than a 21-digit number in the decimal system.

For the sake of illustration, let's take a large number in the decimal number system, which we want to write in the 256-based number system. This:

$$n = 12345678987654321.$$

Then

$$n = 43 \times B^6 + 220 \times B^5 + \cdots + 84 \times B^4 + 98 \times B^3 + 145 \times B^2 + 244 \times B + 177,$$

that is

$$n = (43, 220, 84, 98, 145, 244, 177)_B.$$

When we count in our notebooks at home, most of us still use the decimal number system. Converting from decimal system to the B-based system is easy, and vice versa. When calculating in our notebook, we mostly use the following standard arithmetic operations: ADDITION, SUBTRACTION, MULTIPLICATION and DIVISION WITH REMAINDER.

We usually provide algorithms for more complex operations based on these standard arithmetic operations. To mention just two simple examples, we can thus specify the algorithm of root extraction up to a certain decimal precision or an inverse calculation modulo m. One of the primary considerations in the usability of an algorithm is the running time of the algorithm, i.e., how fast the given algorithm is. But how do we measure the time required for an algorithm? In this lecture notes, the running time is defined as the number of bit operations required to perform the algorithm. Thus, it requires 1 unit of time, e.g., add, subtract, or modulo 2 add bits. Let's take as an example a simple addition in the binary number system:

$$\begin{array}{rr} & 1\;1\;0\;1\;0\;1\;1\;0\;0 \\ + & 1\;0\;0\;1\;1\;0\;1\;1\;0 \;. \\ \hline = & 1\;0\;1\;1\;1\;0\;0\;0\;1\;0 \end{array}$$

Let's look at this addition in a little more detail! We start the operation at the last bit, i.e., $0+0=0$. The next bit is also not a problem, i.e., $0+1=1$. However, after this $1+1=2$, and 2 written in binary number system is 10, i.e., a "carry" of "1" is created, which we have to carry over in the next column. Of course, with bit operations, we also have to take care of carries! The following 8 types of bit operations for addition may occur:

	Bit	Was there a carry in the previous column?
		No
1st number:	0	
2nd number:	0	
Result:	0	
Is there a new carry?	No	

	Bit	Was there a carry in the previous column?
		Yes
1st number:	0	
2nd number:	0	
Result:	1	
Is there a new carry?	No	

	Bit	Was there a carry in the previous column?
		No
1st number:	0	
2nd number:	1	
Result:	1	
Is there a new carry?	No	

	Bit	Was there a carry in the previous column?
		Yes
1st number:	0	
2nd number:	1	
Result:	0	
Is there a new carry?	Yes	

	Bit	Was there a carry in the previous column?
		No
1st number:	1	
2nd number:	0	
Result:	1	
Is there a new carry?	No	

	Bit	Was there a carry in the previous column?
		Yes
1st number:	1	
2nd number:	0	
Result:	0	
Is there a new carry?	Yes	

	Bit	Was there a carry in the previous column?
		No
1st number:	1	
2nd number:	1	
Result:	0	
Is there a new carry?	Yes	

	Bit	Was there a carry in the previous column?
		Yes
1st number:	1	
2nd number:	1	
Result:	1	
Is there a new carry?	Yes	

We do the same for subtraction, only instead of “carry”, “loan” is generated. Bit operations also include AND and OR operations, as well as MODULO 2 ADDITION, i.e.,

AND				
1st number:	0	0	1	1
2nd number:	0	1	0	1
Result:	0	0	0	1

OR				
1st number:	0	0	1	1
2nd number:	0	1	0	1
Result:	0	1	1	1

MOD 2 ADDITION				
1st number:	0	0	1	1
2nd number:	0	1	0	1
Result:	0	1	1	0

Of course, other operations that cannot be derived from the previous ones may also occur during an algorithm, however, their total time requirement is mostly negligible compared to the total time requirements of the bit operations specified so far. Thus, the time required for an algorithm is usually given by the number of bit operations performed during the algorithm. If we extend the definition of bit operations from a binary number system to larger-based number systems, say B-based number systems, then the time requirement changes by at most a constant factor depending on B. Furthermore, the time required for an operation can be different for different algorithms. Therefore, it is advisable to provide an estimate using $O()$ for the time required. (Giving the *exact time requirement* of an operation is usually a very difficult task. Thus we usually only limit ourselves to give upper estimates in this field.)

Obviously, the time required to add two numbers consisting of n digits is $O(n)$. Similarly, the time required to subtract two n digit numbers is also $O(n)$.

Definition 6.1. *The time required for an operation is measured by the number of bit operations required to perform it. Notation:*

$$T(\dots) = \dots.$$

Thus, e.g., memory access and so on are not counted.

Theorem 6.2.

$$T(k \text{ digits } + \ell \text{ digits }) \le \max\{k, \ell\}$$

or

$$T(k \text{ digits } + \ell \text{ digits }) = O(\max\{k, \ell\}).$$

We do not write "$= \max\{k, \ell\}$" because we only give an upper estimate; in many cases (e.g., as we will see with multiplication), there is a significantly better estimate than the first one.

Corollary 6.3.

$$T(m+n) \le \frac{\log \max\{m, n\}}{\log 2} + 1$$

In the case of subtraction, the situation is similar to that of addition, except that instead of “1 carry”, “1 loan” is generated. Therefore, subtraction must be treated in the same way as addition, we will not deal with this further here.

Next, we turn to multiplication. We illustrate the multiplication learned in primary school with an example:

$$\begin{array}{ll}
\underline{11101}\times 1101 & \\
11101 & \to \text{copy} \\
\;11101 & \to \text{copy} \\
\underline{11101} & \to \text{slide it with 2} \\
101111001 &
\end{array}$$

Let’s analyze the figure. The first number to be multiplied has k digits, the second number has ℓ digits. Let ℓ' be ones in the second number. Then obviously $\ell' \leq \ell$.

The time required for sliding and copying is negligible. In other words, we have ℓ' rows, each containing a number with k digits, we have to add them. We do this by adding the second to the first row, the third to this sum, then the fourth to the sum, and so on. After calculating the time required, we get $k\ell'$. Since $\ell' \leq \ell$, the time required for this algorithm is:

Theorem 6.4.

$$T(k \text{ digits } \times \ell \text{ digits }) \leq k\ell$$

Corollary 6.5.

$$T(k \text{ digits } \times k \text{ digits }) \leq k^2.$$

Corollary 6.6.

$$T(m \times n) \leq \left(\frac{\log m}{\log 2} + 1\right)\left(\frac{\log n}{\log 2} + 1\right).$$

The algorithm presented below has the simplest syntax, but it is far from the fastest. In the case of larger numbers, in Chapter 8, we will see considerably more complicated methods, but in terms of the time required for multiplication, much faster.

The complexity of the basic operations is also discussed in detail by D. Knuth in [2].

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] D. E. Knuth, *The Art of Computer Programming (Volume 2) – The Seminumerical Algorithms*, Addison-Wesley 1997

[3] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

6.1 Time required for simple elementary operations

In the previous chapters, we saw how important number systems are for computers. One of the first basic theorems about number systems is the following:

Theorem 6.7. *For $b \in \mathbb{N}$, $b > 1$, every $n \in \mathbb{N}$ can be uniquely written in the form*

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0,$$

where $a_i \in \{0, 1, \ldots, b-1\}$ for all i and $a_k > 0$. This form is called the form in the b-based number system, $a_k, a_{k-1}, \ldots, a_1, a_0$ are the digits of the number n (in the b-based number system).

We will mostly use the binary number system.

Corollary 6.8. *Every $n \in \mathbb{N}$ can be uniquely written in the form*

$$n = \varepsilon_k 2^k + \varepsilon_{k-1} 2^{k-1} + \cdots + \varepsilon_1 2 + \varepsilon_0,$$

where $\varepsilon_i \in \{0, 1\}$ for all i and $\varepsilon_k = 1$. This form is called the dyadic or binary form of n. Here the digits $\varepsilon_k, \varepsilon_{k-1}, \ldots, \varepsilon_1, \varepsilon_0$ are called bits.

In the following, we prove Theorem 6.7..

Proof of Theorem 6.7. Let's first look at the case where $1 \leq n < b$. Then obviously the number of digits of n is 1, i.e., $k = 0$ and $a_0 = n$. Let us further assume that $b \leq n$. We prove the theorem by induction. The first step of the induction is the case $n = b$, when $k = 1$ and $a_1 = 1,\ a_0 = 0$. Let's look at the induction step. Assume that we have already proved the statement for $n = 1, 2, \ldots, m-1$ and we would like to prove it for $n = m$. We are looking for digits $a_k, a_{k-1}, \ldots, a_1, a_0$ for which

$$m = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0.$$

Then

$$m = \left(a_k b^{k-1} + a_{k-1} b^{k-2} + \cdots + a_1\right) b + a_0,$$

where $0 \leq a_k < b$. In this way we write m in the form $m = qb + r$ uniquely by the division algorithm. That is, a_0 is clearly defined: $a_0 = r$. Since $m \geq b$, $q \geq 1$, $q < qb \leq m$, we may use the induction assumption to q:

$$q = a_k b^{k-1} + a_{k-1} b^{k-2} + \cdots + a_2 b + a_1,$$

where the digits $a_k, a_{k-1}, \ldots, a_2, a_1$ uniquely exist. It follows that the statement is also true for $m = qb + r$.

The number of digits in the number $n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$ is $k + 1 = \left[\frac{\log n}{log b}\right] + 1$. Thus, to find the form $n = (a_k a_{k-1} \ldots a_1 a_0)_b$, we need $O(\log n)$ pieces of divisions. From this it is possible to estimate the time required to switch from the binary system to the b-based number system and vice-versa.

We have seen that the switch to the b-based number system can be done using divisions with remainder. However, we have not yet determined the time required for one division. So let's look at the division with remainder. Let $m \geq n$. Then

$$m = qn + r,$$

where $q, r \in \mathbb{Z}$ and $0 \leq r < n$. What will be the result of the division? We would like to determine q and r. We illustrate this with an example: 98 : 5

$$\begin{array}{l}
1100010 : 101 = \underbrace{10011}_{19} \\
\underline{101} \\
\ \ 01001 \\
\underline{\ \ \ \ \ \ 101} \\
\ \ \ \ \ \ 1000 \\
\ \ \ \ \ \underline{\ \ 101} \\
\ \ \ \ \ \ \ \ \underbrace{\underline{011}}_{3}
\end{array}$$

At each step, the number of digits decreases by at least 1 until it runs out:

1st step:	k digits,
2nd step:	$k-1$ digits,
$\vdots$	
$k-\ell+1$th step:	ℓ digits.

In each step, subtraction with ℓ or $\ell+1$ bit operations, but in the latter case the leftmost digit is not informative, since 0 will definitely be there. This is a $(k-\ell+1) \times \ell \leq k\ell$ bit operation.

We finished studying the basic operations. Some additional theorems that can be derived to these, we present them without proofs:

Theorem 6.9. $T\Big((k \ \textit{digits}\ )^n\Big) \leq k^2 \dfrac{(n-1)n}{2}$.

Corollary 6.10.

$$\begin{aligned} T(a^n) &< \left(\frac{\log a}{\log 2} + 1\right)^2 \frac{(n-1)n}{2} \\ &= O(n^2(\log a)^2). \end{aligned}$$

Theorem 6.11. $T(n!) = O(n^2(\log n)^2)$.

Theorem 6.12. $T\left(\text{computing } \binom{n}{m}\right) = O\Big(m^2(\log n)^2\Big)$.

Before we go any further, a basic definition:

Definition 6.13. *A (computational) algorithm is said to have polynomial time (P) if, starting from numbers with $k_1, \dots, k_r$ digits, it gives the result by $O\Big({k_1}^{d_1} \dots {k_r}^{d_r}\Big)$ steps (where $d_1, \dots, d_r$ are given non-negative integers).*

Some examples:

$$\begin{array}{lll} T(k \text{ digits } + \ell \text{ digits}) \leq \max(k, \ell) & & P, \\ T(k \text{ digits } \times \ell \text{ digits}) \leq k\ell & & P, \\ T\Big((k \text{ digits})^n\Big) & = O(k^2 n^2) & noP, \\ T(n!) & = O\Big(n^2(\log n)^2\Big) & noP. \end{array}$$

Theorem 6.14. *Let*

$$\begin{aligned} f(x) &= \sum_{i=0}^{u} a_i x^i, \\ g(x) &= \sum_{j=0}^{v} b_j x^j \end{aligned}$$

be polynomials with integer coefficients, where

$$\max_{i,j}\{|a_i|, |b_j|\} \leq m, \quad v \leq u.$$

Then

$$T(f(x)g(x)) = O\Big(uv((\log m)^2 + \log v)\Big).$$

Proof of Theorem 6.14.

$$f(x)g(x)=\sum_{k=0}^{u+v}\underbrace{\left(\sum_{i+j=k}a_ib_j\right)}_{\text{time required}}x^k$$

$$\underbrace{O\Big(v((\log m)^2+\log v)\Big)}$$

$$O\Big((u+v+1)v\Big((\log m)^2+\log v\Big)\Big)$$

$$=O(uv(\log m)^2+\log v).$$

Many algorithms are known for root extraction. For the fastest of these:

Theorem 6.15. $T([\sqrt{a}])=O((\log a)^3)$.

We do not prove this theorem here.

In the following, we show the time required to switch from one number system to another, so e.g., the time required if n is given in binary form and we want to switch to b based:

Theorem 6.16.

$T(\text{Conversion of } n \text{ from binary form to } b \text{ based number system})=O((\log n)^2)$.

Proof of Theorem 6.16.. The theorem is a consequence of the proof of Theorem 6.7., where we need to add the time required of the used divisions with remainder.

Finally: Problem. How much time does it take to decide whether n is a prime or not? Classic; more recently also because of cryptography. One of the most important problems of computer number theory!

If we wish to determine not only a given number, but all the primes up to a certain limit, then the Eratosthenes' sieve is the most economical. One of the first methods learned in elementary or high school to determine whether a given number n is prime is as follows: Write the primes up to $\sqrt{n}$:

$$p_1 = 2,\ p_2 = 3,\ \ldots,\ p_k \leq \sqrt{n} < p_{k+1},$$

and now n is divided in sequence by $p_1, p_2, \ldots, p_k$, if it is divisible by one of them, it is composite; we stop. If with neither: prime. (We assume that the primes smaller than or equal to $\sqrt{n}$ are given.)

Theorem 6.17. *T(n is a prime? By Eratosthenesian sieve) $= O(\sqrt{n}\log n)$ noP.*

The proof is homework.

There are much faster methods, e.g., Miller–Rabin probability test or Agrawal–Kayal–Saxena algorithm. These will be discussed in more detail later in Chapter 9.

Primality testing is actually polynomial time.

References

[1] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

6.2 Divisibility, Euclidean algorithm

The basics of number theory include the following: divisibility, greatest common divisor, prime and irreducible elements, the basic theorem of number theory, i.e., parts of the material in number theory from the first year of a BSc in mathematics at university.

Let's start with the time required for the Euclidean algorithm.

Theorem 6.18. *For $a \geq b$ we have $T((a, b) =?$ by Eucledian algorithm$) = O((\log a)^3)$.*

Proof of Theorem 6.18.

First, let's write the steps of the Euclidean algorithm for the numbers a and b:

$$\begin{aligned}
a &= bq_1 + r_1, && \text{where } 0 \le r_1 < |b| \\
b &= r_1q_2 + r_2, && \text{where } 0 \le r_2 < r_1 \\
r_1 &= r_2q_3 + r_3, && \text{where } 0 \le r_3 < r_2 \\
&\vdots \\
r_{k-2} &= r_{k-1}q_k + r_k, && \text{where } 0 \le r_k < r_{k-1} \\
r_{k-1} &= r_kq_{k+1}.
\end{aligned}$$

During the algorithm, we always divide the previous remainder by the current remainder. The last non-zero remainder, in this case r_k, will be the greatest common divisor of a and b. Then the following holds for the r_i residues:

Lemma 6.19. *Assume that $a > b$ are natural numbers. Let $r_{-1} = a$, $r_0 = b$. Then for $i = -1, 0, 1, 2, \dots$*

$$r_{i+2} < \frac{r_i}{2}.$$

Proof of Lemma 6.19. Two cases: If $r_{i+1} \le \dfrac{r_i}{2} \;\Rightarrow$

$$r_{i+2} < r_{i+1} \le \frac{r_i}{2}.$$

If, on the other hand, $r_{i+1} > \dfrac{r_i}{2}$, then we consider the $i+2$th division with remainder:

$$\begin{aligned}
r_i &= r_{i+1}q_{i+2} + r_{i+2}, \\
r_{i+2} &= r_i - r_{i+1}\underbrace{q_{i+2}}_{\ge 1} < r_i - \frac{r_i}{2} = \frac{r_i}{2}.
\end{aligned}$$

Applying Lemma 6.19. repeatedly, we get:

Lemma 6.20. $r_i \leq \dfrac{a}{2^{i/2}}$.

Now we can estimate $T\Big((a,b) =?$ by Euclidean algorithm$\Big)$.

Define t by

$$\begin{aligned}
2^{(t-1)/2} &\leq a < 2^{t/2} \\
2^{t-1} &\leq a^2 < 2^t, \\
t-1 &\leq \frac{\log a^2}{\log 2} < t, \qquad t-1 = \left[\frac{\log a^2}{\log 2}\right].
\end{aligned}$$

The number of divisions in the Euclidean algorithm is $k+1$, where the last non-zero remainder is r_k, that is

$$r_k \geq 1.$$

By Lemma 6.20. we get

$$\begin{aligned}
1 \leq r_k &\leq \frac{a}{2^{k/2}}, \\
2^k &\leq a^2 < 2^t, \\
k &< t, \\
k &\leq t-1.
\end{aligned}$$

Number of steps: $k+1 \leq t$. Time required for one step

$$\begin{aligned}
&T(i\text{th division}) = T(r_{i-2} = r_{i-1}q_i + r_i) \\
&\leq \underbrace{r_{i-2}}_{\leq a} \text{ number of digits} \times \underbrace{r_{i-1}}_{\leq a} \text{ number of digits} \\
&\leq (\text{number of digits of } a)^2 \leq \left(\frac{t}{2}+1\right)^2.
\end{aligned}$$

Finally:

$$\begin{aligned}
T\Big((a,b) \overset{?}{=} \text{ by Eucledian algorithm}\Big) &\leq \text{number of steps} \times \max_i T(i\text{th division}) \\
&\leq t\left(\frac{t}{2}+1\right)^2 = O(t^3) = O((\log a)^3).
\end{aligned}$$

Remark: $T\Big((a,b) \overset{?}{=} \text{ by Eucledian algorithm}\Big) = O((\log a)^2)$ is also true, but the proof is much more complicated.

By induction, it is easy to see that for each i the remainder r_i can be written in the form $ax_i + by_i$, where x_i and y_i are integers (one is positive, the other is negative). Applying this to the last non-zero remainder gives:

Theorem 6.21. *For $a \geq b$ we have*

$$\begin{aligned} &T\Big(\text{writing } (a,b) \text{ as a linear combination}\Big) \\ =&T\Big(\text{Finding in the equaion } (a,b) = ax + by \text{ the solution } x, y \in \mathbb{Z}\Big) \\ =&O((\log a)^3). \end{aligned}$$

We leave it to the reader to work out the details of the calculation.

Notation: The ring with unit-element formed by $\bmod m$ residue classes is denoted by Z_m.

Theorem 6.22. *The element $a \in \mathbb{Z}_m$ has a multiplicative inverse if and only if $(a,m) = 1$. If this is satisfied, then the multiplicative inverse can be found with $O((\log m)^3)$ bit operations.*

Proof of Theorem 6.22. If $(a,m) > 1$, then the multiples of a are all divisible by (a,m), so the equation $ax = km + 1$ has no solution (because then $(a,m) \mid ax - km = 1$, which is a contradiction). That is, $ax \equiv 1 \pmod m$ congruence also has no solution, in other words, there is no inverse. Next, we consider the case where $(a,m) = 1$.
Next the estimation of $T(a^{-1} \equiv ? \pmod m)$ follows. We assume $0 < a < m$ and $(a,m) = 1$. Then, applying Theorem 6.21., we get that $\exists\ x, y$

$$ax + my = (a,m) = 1$$

and x, y can be found by $O((\log m)^3)$ bit operations. Then

$$ax \equiv 1 \pmod m,$$

$$x \equiv a^{-1} \pmod{m}.$$

Theorem 6.23. *If p is prime, then all remainder class $\neq 0$ has a multiplicative inverse, and it can be determined by $O((\log p)^3)$ bit operations.*

Corollary 6.24. *The ring $\mathbb{Z}_p$ with the usual addition and multiplication is a field, and we denote it by $\mathbb{F}_p$.*

Next a theorem for solving linear congruences from elementary number theory follows:

Theorem 6.25. *The linear congruence*

$$ax \equiv b \pmod{m}$$

can be solved $\Leftrightarrow (a, m) \mid b$. *If this holds, then this linear congruence equivalent with the following linear congruence*

$$a'x \equiv b' \qquad \pmod{m'},$$

with

$$a' = \frac{a}{(a,m)}, \quad m' = \frac{m}{(a,m)}, \quad b' = \frac{b}{(a,m)} \quad (\Rightarrow \ (a', m') = 1).$$

The solutions form a fixed residue class $\bmod m'$. *This solution can be found with $O((\log m')^3)$ bit operations.*

References

[1] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

6.3 Modular exponentiation

It is known that $T(a^n) = O\Big(n^2(\log a)^2\Big)$.

Mod m the same can be calculated much faster:

Theorem 6.26. *$T(a^n \mod m) = O\Big((\log n)(\log m)^2\Big)$.*

Proof of Theorem 6.26. Let's write n in binary form

$$n = \varepsilon_0 + \varepsilon_1 \cdot 2 + \varepsilon_2 \cdot 2^2 + \ldots + \varepsilon_k 2^k.$$

where $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{k-1} \in \{0, 1\}$ and $\varepsilon_k = 1$. Then $k = O(\log n)$. For determining the digits ε_i's:

$$T(\text{binary form}) = O((\log n)^2).$$

For $i = 0, 1, \ldots, k$ let a_i be the least non-negative residue of a^{2^i} modulo m:

$$a_i \equiv a^{2^i} \pmod{m} \quad 0 \leq a_i < m.$$

Determine these numbers in order:

$$a_0 = a^{2^0} = a^1 = a.$$

If a_i is given, then for a_{i+1} we have

$$a_{i+1} \equiv a^{2^{i+1}} \equiv a^{2^i 2} \equiv \left(a^{2^i}\right)^2 \equiv {a_i}^2 \pmod{m_i}$$

By $0 \leq a_i < m$ we get

$$T({a_i}^2) = O((\log m)^2).$$

Then for the division with the remainder m we have:

$$T({a_i}^2 \pmod{m}) = O(\log(m^2) \log m) = O((\log m)^2),$$
$$T(a_0, a_1, \ldots, a_k) = O(k(\log m)^2) = O((\log n)(\log m)^2).$$

Furthermore:

$$a^n = a^{\sum\limits_{i=1}^{k} \varepsilon_i 2^i} = \prod_{i=1}^{k} a^{\varepsilon_i 2^i} = \underbrace{\prod_{i=1}^{k} {a_i}^{\varepsilon_i}}$$

The product of those a_i's for which $\varepsilon_i = 1$.

Multiplied by these in a row, in a maximum of $k = O(\log n)$ steps, where the product of two numbers $\leq m$ has to be calculated in each step. For the time required for a multiplication, we know that

$$T(\text{multiplication}) = O((\log m)^2).$$

Thus $T(a^n \bmod m) = O\Big((\log n)(\log m)^2\Big)$.

References

[1] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

7 Modular square root

In Chapter 3, in the section on the Legendre symbol, we discussed the solvability of $x^2 \equiv a \pmod p$, but if it is solvable, how to find a solution of x? In this chapter, we study two polynomial time algorithms for the above problem.

7.1 Tonelli-Shanks's algorithm

The first algorithm in question is the so-called Tonelli–Shanks algorithm. The version discussed here was developed by Daniel Shanks [3] in 1973, who explained:

"My tardiness in learning of these historical references was because I had lent Volume 1 of Dickson's History to a friend and it was never returned."

So, according to Dickson's book, a more redundant version of the algorithm already existed in 1891 and is associated with the name of Tonelli [1]. In this lecture notes the description of the algorithm is based on Koeblitz's book [2].

So the *problem:* For a given prime $p > 2$, $a \in \mathbb{Z}_p$, for which $(a, p) = 1$ and $\left(\dfrac{a}{p}\right) = +1$, we would like to find the solution of $x^2 \equiv a \pmod p$.

Let $p - 1 = 2^\alpha s$, $\alpha \in \mathbb{N}$, s be odd. We give an algorithm for determining x of length α (so if $\alpha = 1$, i.e., $p - 1 = 2(2k + 1) = 4k + 2 \Leftrightarrow p = 4k + 3$, we get the result in one step).

Consider a quadratic non-residue n (if the Riemann conjecture holds, such n can be found in polynomial time, we will return to this later). Let's compute $b \overset{\text{def}}{\equiv} n^s = n^{(p-1)/2^\alpha} \pmod p$ and $r \overset{\text{def}}{\equiv} a^{(s+1)/2} \pmod p$.

Lemma 7.1. *The number r^2a^{-1} is the $2^{\alpha-1}$th root of unity modulo p.*

Proof of Lemma 7.1. We want to extract a root from the number a, so a is a quadratic residue modulo p, and then due to Euler's lemma $a^{(p-1)/2} \equiv 1$

(mod p). Using this and the definition of the number r:

$$(r^2a^{-1})^{2^{\alpha-1}} = \left(a^{\frac{s+1}{2}\cdot 2}a^{-1}\right)^{2^{\alpha-1}} = (a^s)^{2^{\alpha-1}} = a^{\frac{p-1}{2}} \equiv 1 \pmod{p}.$$

Remark. If $\alpha = 1$, then $2^{\alpha-1} = 2^0 = 1$, so $r^2a^{-1} \equiv 1 \pmod{p}$, that is, the root of the number a is r.

But what if $\alpha > 1$?

We know that r^2a^{-1} is the $2^{\alpha-1}$th root of unity. Thus

$$r^2a^{-1} \equiv \omega \pmod{p},$$
$$\omega^{-1}r^2 \equiv a \pmod{p}.$$

Which shows that the solution of

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

must be sought in the form $r\varepsilon$, where ε is a 2^α-th root of unity. Now we will need the number b, whose definition was $b \overset{\text{def}}{=} n^s$. Then:

Lemma 7.2. *The number b is the 2^αth primitive root of unity modulo p.*

Proof of Lemma 7.2. a) The number b is the 2^α-th root of unity:

$$b^{2^\alpha} = (n^s)^{2^\alpha} = n^{2^\alpha s} = n^{p-1} \equiv 1 \pmod{p}.$$

b) The number b is a $\underline{\text{primitive}}$ 2^α-th root of unity: Indirectly, assume that it is not primitive, i.e., its order is $< 2^\alpha$. Since this order is a divisor of 2^α and $< 2^\alpha$, it is also a divisor of $2^{\alpha-1}$. That is, due to the basic properties of the order, $b^{2^{\alpha-1}} \equiv 1 \pmod{p}$ is also satisfied. However, due to Euler's lemma:

$$b^{2^{\alpha-1}} \equiv (n^s)^{2^{\alpha-1}} \equiv n^{\frac{p-1}{2}} \equiv \left(\frac{n}{p}\right) \equiv -1 \pmod{p},$$

which is contradiction.

Returning, we look for a solution x in the form $r\varepsilon$, where ε is a suitable 2^α-th root of unity. That is, it is of the form $\varepsilon = b^j$, where $0 \leq j < 2^\alpha$. Then $x = rb^j$, so we are looking for j such that

$$x^2a^{-1} \equiv (rb^j)^2a^{-1} \equiv 1 \pmod{p}.$$

Write this j in binary form:

$$j = j_0 + 2j_1 + 2^2j_2 + \dots,$$

here, due to $j < 2^\alpha$, $2^{\alpha-1}j_{\alpha-1}$ should be the last term.

But b is the 2^α primitive root of unity, so

$$\left(b^{2^{\alpha-1}}\right)^2 \equiv 1 \pmod{p}$$
$$b^{2^{\alpha-1}} \equiv \begin{cases} -1 \\ +1 \end{cases}$$

Because of Lemma 7.2.

$$b^{2^{\alpha-1}} \equiv -1 \pmod{p}. \tag{7.1}$$

Thus, if the binary form of j ends with $2^{\alpha-1}$, i.e., $j_{\alpha-1} = 1 \Rightarrow$ discarding this from j: $j' = j - 2^{\alpha-1}$ is such that

$$b^j \equiv b^{j'}\underbrace{b^{2^{\alpha-1}}}_{-1} \equiv -b^{j'} \pmod{p},$$

so if $x = rb^j$ is a solution, so is $x = rb^{j'}$. Therefore, it can be assumed that we only go up to $2^{\alpha-2}$. Now, j will be defined recursively starting from j_0.

1st step: determining j_0. By Lemma 7.1.:

$$(r^2a^{-1})^{2^{\alpha-1}} \equiv 1 \pmod{p},$$
$$\left((r^2a^{-1})^{2^{\alpha-2}}\right)^2 \equiv 1 \pmod{p},$$
$$(r^2a^{-1})^{2^{\alpha-2}} \equiv \pm 1 \pmod{p}.$$

Let

$$j_0 \overset{\text{def}}{=} \begin{cases} 0, & \text{ha } (r^2a^{-1})^{2^{\alpha-2}} \equiv 1 \pmod{p}, \\ 1, & \text{ha } (r^2a^{-1})^{2^{\alpha-2}} \equiv -1 \pmod{p}. \end{cases} \tag{7.2}$$

Then using (7.1) and (7.2):

$$\left((b^{j_0}r)^2a^{-1}\right)^{2^{\alpha-2}} = b^{j_0\cdot 2^{\alpha-1}}(r^2a^{-1})^{2^{\alpha-2}} \equiv (-1)^{j_0}(r^2a^{-1})^{2^{\alpha-2}} \equiv 1 \pmod{p}.$$

Now suppose $1 \le k \le \alpha - 2$ and $j_0, j_1, \dots.\ j_{k-1}$ are already given such that

$$\left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}}r\right)^2 a^{-1}\right)^{2^{\alpha-k-1}} \equiv 1 \pmod{p}. \tag{7.3}$$

Exercise: Define j_k so that (7.3) is also satisfied in place of $k-1$ with k. To prove this consider

$$\left(\left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}}r\right)^2 a^{-1}\right)^{2^{\alpha-k-2}}\right)^2 \equiv 1 \pmod{p}.$$

Here, the part in the outer parenthesis (denoted by w) is modulo p congruent to $+1$ or -1. If for this $w \equiv +1 \pmod{p}, æ_k \overset{\text{def}}{=} 0$, while if $w \equiv -1$ $\pmod{p}, æ_k \overset{\text{def}}{=} 1$.

Then indeed

$$\begin{aligned} &\left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}+2^kj_k}r\right)^2 a^{-1}\right)^{2^{\alpha-k-2}} \equiv \\ &\equiv \left(b^{2^{k+1}j_k}\right)^{2^{\alpha-k-2}} \left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}}r\right)^2 a^{-1}\right)^{2^{\alpha-k-2}} \\ &\equiv \left(b^{2^{\alpha-1}}\right)^{j_k} \left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}}r\right)^2 a^{-1}\right)^{2^{\alpha-k-2}} \\ &\equiv (-1)^{j_k} \left(\left(b^{j_0+2j_1+\ldots+2^{k-1}j_{k-1}}r\right)^2 a^{-1}\right)^{2^{\alpha-k-2}} \equiv 1 \pmod{p}. \end{aligned}$$

Finally, taking $k = \alpha - 2$, we get $j_{\alpha-2}$, and then

$$\left(b^{j_0+2j_1+\ldots+2^{\alpha-2}j_{\alpha-2}}r\right)^2 a^{-1} \equiv 1 \pmod{p}.$$

So $x \equiv b^jr \pmod{p}$ (where $j = j_0 + 2j_1 + \ldots + 2^{\alpha-2}j_{\alpha-2}$) is solution of the congruence $x^2 \equiv a \pmod{p}$.

References

[1] L. E. Dickson, *History of the Theory of Numbers*, Vol. 1, Washington, Carnegie Institution of Washington (1919), 215–216.

[2] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[3] D. Shanks, *Five number-theoretic algorithms*, Proceedings of the Second Manitoba Conference on Numerical Mathematics (1973), 51–70.

7.2 Perelta's algorithm

In this chapter, we give an alternative tricky method for computing the square root modulo p, Perelta's algorithm [2]. The description of the algorithm is based on Robin Chapman's note [1], who provided a very clear description of Perelta's algorithm with matrices.

Two integer matrices A and B are said to be congruent modulo p if their corresponding elements are congruent modulo p. Notation:

$$A \equiv B \pmod{p}.$$

Congruences of matrices can be handled in the same way as for integers.

Let p be an odd prime and a be a quadratic residue, i.e., $\left(\frac{a}{p}\right) = 1$. Suppose that we would like to solve the following congruence:

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

In the first step, we look for an integer b for which

$$\left(\frac{b^2 - a}{p}\right) = -1. \tag{7.4}$$

This is achieved by random methods, the chance that $b^2 - a$ quadratic non-residue for a randomly chosen b is about $1/2$. Thus, by repeating this step a couple of times, sooner or later we get a b residue for which (7.4) holds. We define the matrices A and B by the following:

$$A = \begin{bmatrix} 0 & 1 \\ a & 0 \end{bmatrix}, \qquad B = bI + A = \begin{bmatrix} b & 1 \\ a & b \end{bmatrix},$$

where I is the 2×2 identity matrix. After that, we calculate the $B^{(p-1)/2}$ matrix modulo p by repeated squaring. Amazingly, we find that

$$B^{(p-1)/2} \equiv \begin{bmatrix} 0 & r \\ s & 0 \end{bmatrix} \pmod p,$$

where $s^2 \equiv a \pmod p$.

Let's see why this algorithm works. First, calculate the matrix $B^p = (bI + A)^p$ based on the binomial theorem:

$$B^p = \sum_{j=0}^{p} \binom{p}{j} b^{p-j} A^j \equiv b^p I + A^p \pmod p, \tag{7.5}$$

since for $1 \le j \le p-1$ the binomial coefficient $\binom{p}{j}$ is divisible by p. By Fermat's little theorem, $b^p \equiv b \pmod p$. By Euler's lemma, $a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) = 1 \pmod p$. Simple matrix multiplication shows that $A^2 = aI$, so

$$A^p = A \cdot \left(A^2\right)^{(p-1)/2} = A \cdot (aI)^{(p-1)/2} = a^{(p-1)/2} A \equiv A \pmod p.$$

Comparing the above with (7.5):

$$B^p \equiv bI + A = B \pmod p. \tag{7.6}$$

Since the determinant of the matrix B is not zero modulo p, it has an inverse modulo p, which we will denote by B^*. Multiplying the congruence (7.6) by B^* we get

$$B^{p-1} \equiv I \pmod p. \tag{7.7}$$

Using the equation $A^2 = aI$, it can be proved by induction that for every natural number n

$$B^n = x_n I + y_n A,$$

where x_n and y_n are integers. Indeed, the starting step of induction for $n = 1$ we have $B = bI + A$, that is $x_1 = b, y_1 = 1$. Next suppose that we have proved the statement for n, and now we would like to prove it for $n + 1$:

$$\begin{aligned} B^{n+1} &= B^n \cdot B = (x_n A + y_n I) \cdot (bA + I) = x_n b A^2 + (x_n + y_n b)A + y_n I \\ &= x_n abI + (x_n + y_n b)A + y_n I = (x_n + y_n b)A + (x_n ab + y_n), \end{aligned}$$

thus we may take $x_{n+1} = x_n + y_n b$ and $y_n = x_n ab + y_n$. Writing our statement for the case $B^{(p-1)/2}$, we get that there exists t and r, for which

$$B^{(p-1)/2} = tI + rA = \begin{bmatrix} t & r \\ ar & t \end{bmatrix}. \tag{7.8}$$

Recall that $B^{p-1} = I$ (see (7.7)), but the upper right element of the square of the matrix (7.8) is the number $2rt$, so we get

$$2rt \equiv 0 \pmod{p},$$

from which $r \equiv 0 \pmod{p}$ or $t \equiv 0 \pmod{p}$. We first exclude the case $r \equiv 0 \pmod{p}$. Indeed, if $r \equiv 0 \pmod{p}$, then by (7.8) we get $B^{(p-1)/2} \equiv tI \pmod{p}$. Thus:

$$I \equiv B^{p-1} = (B^{(p-1)/2})^2 \equiv (tI)^2 = t^2 I \pmod{p}.$$

That is, $t^2 \equiv 1 \pmod{p}$. Then $\det\left(B^{(p-1)/2}\right) \equiv t^2 \equiv 1 \pmod{p}$, but due to the multiplicative property of determinants we get

$$\det\left(B^{(p-1)/2}\right) = (\det B)^{(p-1)/2} = (b^2 - a)^{(p-1)/2} \equiv \left(\frac{b^2 - a}{p}\right) \equiv -1 \pmod{p},$$

which is contradiction. That is $r \not\equiv 0 \pmod{p}$, so $t \equiv 0 \pmod{p}$. Then:

$$B^{(p-1)/2} \equiv rA = \begin{bmatrix} 0 & r \\ ra & 0 \end{bmatrix} \pmod{p}.$$

Thus:

$$I \equiv B^{p-1} \equiv (rA)^2 = r^2 aI \pmod{p}.$$

That is, $r^2 a \equiv 1 \pmod{p}$. By the definition of s we have $s \equiv ra \pmod{p}$, so $s^2 \equiv r^2 a^2 \equiv a \pmod{p}$, which completes the proof.

It remains to prove that at least half of the residue classes $b^2 - a$ are quadratic non-residues and the other half are quadratic residues, except the two solutions of the congruence $x^2 \equiv a \pmod{p}$ are s and $-s$, when $b^2 - a \equiv 0 \pmod{p}$. For this, consider the following sum

$$\sum_{b=0}^{p-1} \left(\frac{b^2 - a}{p} \right).$$

If it is -1, we are done. To do this, replace a with s^2 and use the following identity:

$$\sum_{b=0}^{p-1} \left(\frac{b^2 - a}{p} \right) = \sum_{b=0}^{p-1} \left(\frac{b^2 - s^2}{p} \right) = \sum_{b=0}^{p-1} \left(\frac{(b-s)(b+s)}{p} \right)$$
$$= \sum_{b=0,\ b \neq -s}^{p-1} \left(\frac{(b-s)/(b+s)}{p} \right)$$

It is easy to check that as b runs on the last sum $(b-s)/(b+s)$ takes the elements of a complete remainder system except 1. Thus

$$\sum_{b=0}^{p-1} \left(\frac{b^2 - a}{p} \right) = -1,$$

and this proves our last statement.

References

[1] R. Chapman, *Perelta's algorithm*, https://empslocal.ex.ac.uk/people/staff/rjchapma/courses/nt13/peralta.pdf.

[2] R. C. Perelta, *A simple and fast probabilistic algorithm for computing square roots modulo a prime number*, I. E. E. E. Trans. Inform. Theory 32 (1986), 846-847.

7.3 Least quadratic non-residue

Finally, at the end of the chapter, a few words about which methods are worth using to find a quadratic non-residue modulo p. The random method is certainly the most effective. We choose a random n and see if the value of the Legendre symbol $\left(\frac{n}{p}\right)$ will be -1. If so, then n is quadratic non-residue modulo p. Since modulo p has a total of $(p-1)/2$ quadratic residues and $(p-1)/2$ quadratic non-residues, if $p \nmid n$, then the chance that n is quadratic non-residue is 50%. That is, if we try, say, 200 times the algorithm for different n's, the chance of not finding a quadratic non-residue modulo p is less than $\frac{1}{2^{200}}$. This is an even smaller probability that we hit the jackpot three times in a row in the Hungarian lottery.

Although this method works perfectly in practical applications, mathematicians also wanted to develop a deterministic algorithm for constructing quadratic non-residues. Perhaps the simplest idea for this is to try the natural numbers in a row, 2,3,4,5,6,7, and so on (in fact, it is enough to consider the prime numbers), and the algorithm stops at the first quadratic non-residue. In order to prove that this algorithm is polynomial time, it is necessary to have a sharp upper estimate for the smallest quadratic non-residue, i.e., we need an upper bound of the form $(\log p)^k$. This problem is interesting in itself. Let $n(p)$ denote the smallest positive quadratic non-residue modulo p. Burgess [2] proved the following in 1957:

$$\forall \varepsilon > 0 \ \exists p_0(\varepsilon), \text{ if } p > p_0(\varepsilon) \text{ then } n(p) < p^{1/(4\sqrt{e})+\varepsilon}.$$

Unfortunately, this does not yet convince us that there is a deterministic polynomial-time algorithm for finding a quadratic non-residue. However, assuming the generalized Riemann hypothesis, Bach [1] proved that

$$n(p) \leq 2(\log p)^2.$$

Thus, we have a conditional result, but as it is well known, neither the

Riemann conjecture nor the general Riemann conjecture has been proven yet.

References

[1] E. Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. 55 (191) (1990), 355-380.

[2] D. A. Burgess, *The distribution of quadratic residues and non-residues*, Mathematatika 4 (2) (1957), 106-112.

8 Fast multiplication

When an algorithm is programmed, the speed of the program depends on many details, one of which is how much time required for a standard arithmetic operation. We saw that

$$T(a+b) = O(\max\{\log a, \log b\}),$$

and using the multiplication that we learned in elementary school we have

$$T(a \times b) = O(\max\{\log a, \log b\}).$$

Faster methods for the multiplication algorithm were found in the 1960s and 1970s, and modern computers already use them. In this chapter, we describe these types of multiplication based on the book by Das [1].

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

8.1 Karatsuba-Ofman multiplication

The first fast multiplication algorithm is due to Karatsuba and Ofman [2]. We will describe the algorithm using Das's book [1].

Let a and b be two n-digit numbers in a B-based number system. For simplicity, assume that n is even. Let $m = n/2$. Write a and b in the form

$$a = A_1 B^m + A_0, \quad b = B_1 B^m + B_0,$$

where A_0, A_1, B_0, B_1 are numbers consisting of at most $m = n/2$ digits. Then

$$ab = (A_1 B_1) B^{2m} + (A_1 B_0 + A_0 B_1) B^m + A_0 B_0. \tag{8.1}$$

At first, we would think that 4 products need to be calculated here. However, we really only need the following 3 coefficients: $A_1B_1, A_1B_0 + A_0B_1, A_0B_0$.

We can notice that

$$A_1B_0 + A_0B_1 = (A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0.$$

That is, it is enough to calculate 3 products: $(A_1 + A_0)(B_1 + B_0)$, A_1B_1, A_0B_0. Unfortunately, two multiplication factors in $(A_1 + A_0)(B_1 + B_0)$ can also be numbers with $m + 1$ digits. We can help with this, too by

$$A_1B_0 + A_0B_1 = A_1B_1 + A_0B_0 - (A_1 - A_0)(B_1 - B_0).$$

In other words, we have to calculate the following 3 multiplications: A_1B_1, A_0B_0, $(A_1 - A_0)(B_1 - B_0)$. Each of these factors has at most $m = \lceil n/2 \rceil$ digits. Thus, if the the time required to multiply 2 numbers with n digits is denoted by t_n, then the time required to calculate the coefficients in (8.1) is $3t_{\lceil n/2 \rceil} + O(n)$, since the time required to calculate the 3 multiplications is $3t_m = 3t_{\lceil n/2 \rceil}$ and there are also some additions and subtractions, their time required is $O(n)$. That is, we get the following recursion:

$$t_n \leq 3t_{\lceil n/2 \rceil} + O(n).$$

If n is of the form $n = 2^k$, then

$$t_{2^k} \leq 3t_{2^{k-1}} + O(2^k).$$

Then, by induction on k, it is easy to prove that

$$t_{2^k} \leq 3^k + O(2^k).$$

We can obtain the estimate of t_n for a general n as follows. Let's take the smallest power of 2, which is greater than or equal to n:

$$2^{k-1} < n \leq 2^k.$$

Then

$$t_n \leq t_{2^k} \leq 3^k + O(2^k) \leq 3 \cdot 3^{k-1} + O(2^k) = 3 \cdot \left(2^{k-1}\right)^{\log 3/\log 2} + O(n)$$

$$= O(n^{\log 3/\log 2}).$$

That is, the time required for the Karatsuba-Ofman multiplication is $O(n^{\log 3/\log 2})$. This is already significantly faster than the time required for the standard schoolbook-method, which is $O(n^2)$.

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] A. Karatsuba, Yu. Ofman, *Multiplication of many digital numbers by automatic computers*, Doklady Akad. Nauk. SSSR, Vol. 145 (1962), 293-294.

8.2 Toom-Cook multiplication

Toom [2] and Cook [3] generalized the Karatsuba-Ofman multiplication as follows: Let a and b be two numbers consisting of n digits. Let $m = \lceil n/3 \rceil$ and write a and b in the form

$$a = A_2R^2 + a_1R + A_0, \quad b = B_2R^2 + B_1R + B_0, \tag{8.2}$$

where $R = B^m$. Then

$$c = ab = C_4R^4 + C_3R^3 + C_2R^2 + C_1R + C_0, \tag{8.3}$$

where

$$\begin{aligned} C_4 &= A_2B_2 \\ C_3 &= A_2B_1 + A_1B_2 \\ C_2 &= A_2B_0 + A_1B_1 + A_0B_2 \\ C_1 &= A_1B_0 + A_0B_1 \end{aligned}$$

$$C_0 = A_0 B_0.$$

At first examination, this appears to be a 9-piece multiplication. That is, if we denote by t_n the time required for multiplying n-digit numbers, then from the above (including the additions) we get the following recursion:

$$t_n \leq 9 t_{\lceil n/3 \rceil} + O(n).$$

However, now, similarly to the Karatsuba-Ofman's algorithm, in (8.3), less than 9 multiplications are enough to calculate the coefficients. Let

$$\begin{aligned}
a(x) &\overset{\text{def}}{=} A_2 x^2 + A_1 x + A_0 \\
b(x) &\overset{\text{def}}{=} B_2 x^2 + B_1 x + B_0 \\
c(x) &\overset{\text{def}}{=} C_4 x^4 + C_3 x^3 + C_2 x^2 + C_1 x + C_0 = a(x) b(x)
\end{aligned}$$

Here, the variable x can take any real or complex number. For every $k \in \mathbb{C}$ we have

$$c(k) = a(k) b(k). \tag{8.4}$$

Usually, the domain of a polynomial p is $\mathbb{R}$ or $\mathbb{C}$, but now we extend the domain with an ∞ symbol, for the polynomial

$$p(x) = r_n x^n + r_{n-1} x^{n-1} + \cdots + r_1 x + r_0$$

let $p(\infty) = r_n$. Obviously, then the formula (8.4) for $k = \infty$ also holds, i.e.,

$$c(\infty) = a(\infty) b(\infty).$$

We will use the following:

$$\begin{aligned}
c(\infty) &= C_4 = A_2 B_2 \\
c(0) &= C_0 = A_0 B_0 \\
c(1) &= C_4 + C_3 + C_2 + C_1 + C_0 = (A_2 + A_1 + A_0)(B_2 + B_1 + B_0) \\
c(-1) &= C_4 - C_3 + C_2 - C_1 + C_0 = (A_2 - A_1 + A_0)(B_2 - B_1 + B_0)
\end{aligned}$$

$$\begin{aligned}c(-2) &= 16C_4 - 8C_3 + 4c_2 - 2C_1 + C_0\\ &= (4A_2 - 2A_1 + A_0)(4B_2 - 2B_1 + B_0) \qquad (8.5)\end{aligned}$$

From this using matrices we get

$$\begin{pmatrix} c(\infty)\\ c(0)\\ c(1)\\ c(-1)\\ c(-2)\end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1 & 1\\ 16 & -8 & 4 & -2 & 1\end{pmatrix} \begin{pmatrix} C_4\\ C_3\\ C_2\\ C_1\\ C_0\end{pmatrix}.$$

Multiplying by the inverse matrix we get

$$\begin{pmatrix} C_4\\ C_3\\ C_2\\ C_1\\ C_0\end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0\\ 2 & -1/2 & 1/6 & 1/2 & -1/6\\ -1 & -1 & 1/2 & 1/2 & 0\\ -2 & 1/2 & 1/3 & -1 & 1/6\\ 0 & 1 & 0 & 0 & 0\end{pmatrix} \begin{pmatrix} c(\infty)\\ c(0)\\ c(1)\\ c(-1)\\ c(-2).\end{pmatrix}$$

That is

$$\begin{aligned}C_4 &= c(\infty)\\ C_3 &= (12c(\infty) - 3c(0) + c(1) + 3c(-1) - c(-2))/6\\ C_2 &= (-2c(\infty) - 2c(0) + c(1) + c(-1))/2\\ C_1 &= (-12c(\infty) + 3c(0) + 2c(1) - 6c(-1) + c(-2))/6\\ C_0 &= c(0) \qquad (8.6)\end{aligned}$$

That is, to calculate C_4, C_3, C_2, C_1, C_0 in (8.3), we need to calculate 5 products, namely the numbers $c(\infty), c(0), c(1), c(-1), c(-2)$, whose product form is given by (8.5).

Using Toom-Cook multiplication, for the time required to multiply two n-digit numbers we get the following recursion:

$$t_n \leq 5t_{\lceil n/3\rceil} + O(n).$$

If n is of the form $n = 3^k$, then

$$t_{3^k} \leq 5t_{3^{k-1}} + O(3^k).$$

From this, it is easy to prove by induction on k that

$$t_{3^k} \leq 5^k + O(3^k).$$

We can obtain the estimate of t_n for a general n as follows. Let's take the smallest power of 3, which is greater than or equal to n, that is

$$3^{k-1} < n \leq 3^k.$$

Then

$$\begin{aligned} t_n &\leq t_{3^k} \leq 5^k + O(3^k) \leq 5 \cdot 5^{k-1} + O(3^k) = 5 \cdot \left(3^{k-1}\right)^{\log 5/\log 3} + O(n) \\ &= O(n^{\log 5/\log 3}). \end{aligned}$$

That is, the time required for the Toom-Cook multiplication is $O(n^{\log 5/\log 3})$. The Toom-Cook multiplication can be further generalized to higher degree polynomials. If in (8.2) we have $m = \lceil n/k \rceil$ and

$$\begin{aligned} a &= A_{k-1}R^{k-1} + A_{k-2}R^{k-2} + \cdots + A_1R + A_0 \\ b &= B_{k-1}R^{k-1} + B_{k-2}R^{k-2} + \cdots + B_1R + B_0, \end{aligned}$$

where $R = B^m$, then in order to calculate the product

$$ab = C_{2k-1}R^{2k-1} + C_{2k-2}R^{2k-2} + \cdots + C_1R + C_0$$

it is enough to perform k multiplications. Thus, the time requirement becomes $O(n^{\log(2k-1)/\log k})$, where the constant in O depends only on k. If k is appropriately chosen as a function of n, the best theoretically achievable running time is $O\left(n2^{5\sqrt{\log n}}\right)$, but practical applications show that k should not be chosen greater than 4.

References

[1] A. Das, *Computational Number Theory*, Discrete Mathematics and its Applications, CRC Press, 2013.

[2] A. L. Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*, Doklady Acad. Nauk SSSR, 4 (3) (1963), 714-716, 1963.

[3] S. A. Cook, *On the Minimum Computation Time of Functions*, PhD thesis, Department of Mathematics, Harvard University, 1966.

8.3 Fast Fourier Transform

Schönhage and Strassen [3] in 1971 provided an alternative method for fast multiplication, which was based on polynomial evaluation and interpolation. The time requirement of this new method reached $O(n \log n \log\log n)$ limit, so it is practically the fastest algorithm for multiplying integers whose size consists of at least one or two thousand digits. The entire Schönhage-Strassen algorithm is a bit complicated to discuss in this introductory lecture notes, so here we present a slightly simplified version based only on the book by Das [1].

Suppose a and b are n-digit numbers in a B-based number system, where B is now a power of two, i.e., $B = 2^r$. Let

$$2^{t-1} < n \leq 2^t$$

and

$$N = 2^{t+1}.$$

(That is, N is the smallest power of two, which is $\geq 2n$.) Let's write both a and b in the B-based number system, so we write a few zero digits at the beginning of both numbers to consider both numbers as consisting of exactly

N ($= 2^{t+1}$) digits. Then:

$$a = a_{N-1}B^{N-1} + a_{N-2}B^{N-2} + \cdots + a_1 B + a_0$$
$$b = b_{N-1}B^{N-1} + b_{N-2}B^{N-2} + \cdots + b_1 B + b_0. \tag{8.7}$$

Since a and b originally (without leading zeros) had only $n \leq \frac{N}{2}$ digits, we know in (8.7) that $\frac{N}{2} \leq fori, j < N$, $a_i = 0$ and $b_j = 0$. Based on (8.7), we can assign a polynomial to both a and b:

$$\overline{a}(x) \stackrel{\text{def}}{=} a_{N-1}x^{N-1} + a_{N-2}x^{N-2} + \cdots + a_1 x + a_0$$
$$\overline{b}(x) \stackrel{\text{def}}{=} b_{N-1}x^{N-1} + b_{N-2}x^{N-2} + \cdots + b_1 x + b_0.$$

Clearly then

$$a = \overline{a}(B) \text{ and } b = \overline{b}(B).$$

The product of a and b is

$$ab = c = c_{N-1}B^{N-1} + c_{N-2}B^{N-2} + \cdots + c_1 B + c_0,$$

where

$$c_k = \sum_{\substack{0 \leq i,j \leq N-1 \\ i+j=k}} a_i b_j.$$

Similarly to polynomials $\overline{a}(x)$ and $\overline{b}(x)$, we can define a polynomial $\overline{c}(x)$ by the following:

$$\overline{c}(x) \stackrel{\text{def}}{=} c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + \cdots + c_1 x + c_0.$$

Then $\overline{c}(x) = \overline{a}(x)\overline{b}(x)$, moreover

$$c = ab = \overline{c}(B) = \overline{a}(B)\overline{b}(B).$$

Let ω_N be an N-th primitive root of unity. (If we calculate in the field of complex numbers, then ω_N can be taken as $e^{2\pi i/N}$. But we can also take fields different from complex numbers, we will return to this later.)

Definition 8.1. *The discrete Fourier transform (DFT) of the sequence* $(a_{N-1}, a_{N-2}, \ldots, a_1, a_0)$ *is that sequence* $(A_{N-1}, A_{N-2}, \ldots, A_1, A_0)$ *for which*

$$A_k \overset{\text{def}}{=} \sum_{j=0}^{N-1} \omega_N^{kj} a_j. \tag{8.8}$$

Remark 8.2. *Using the notations so far*

$$A_k = \overline{a}\left(\omega_N^k\right).$$

Let the discrete Fourier transform of the sequence $(b_{N-1}, b_{N-2}, \ldots, b_1, b_0)$ be $(B_{N-1}, B_{N-2}, \ldots, B_1, B_0)$ and the discrete Fourier transform of the sequence $(c_{N-1}, c_{N-2}, \ldots, c_1, c_0)$ be $(C_{N-1}, C_{N-2}, \ldots, C_1, C_0)$. Using the notations so far

$$B_k = \overline{b}\left(\omega_N^k\right) \quad \text{and} \quad C_k = \overline{c}\left(\omega_N^k\right).$$

Since $\overline{a}(x)\overline{b}(x) = \overline{c}(x)$, we have

$$C_k = \overline{c}\left(\omega_N^k\right) = \overline{a}\left(\omega_N^k\right)\overline{b}\left(\omega_N^k\right) = A_k B_k.$$

Thus, if we can quickly and efficiently calculate the discrete Fourier transform of a series (i.e., $(A_{N-1}, A_{N-2}, \ldots, A_1, A_0)$ and $(B_{N-1}, B_{N-2}, \ldots, B_1, B_0)$), then by n piecis of multiplications, we get $(C_{N-1}, C_{N-2}, \ldots, C_1, C_0)$. If we take another discrete Fourier transformation on the sequence $(C_{N-1}, C_{N-2}, \ldots, C_1, C_0)$ (let the resulting sequence be $(\widetilde{C}_{N-1}, \widetilde{C}_{N-2}, \ldots, \widetilde{C}_1, \widetilde{C}_0)$), then we get back the elements of the original sequence, more precisely:

$$(c_{N-1}, c_{N-2}, \ldots, c_1, c_0) = \frac{1}{N}(\widetilde{C}_0, \widetilde{C}_1, \ldots, \widetilde{C}_{N-1}, \widetilde{C}_N). \tag{8.9}$$

The above connection can be easily verified:

$$\begin{aligned} \widetilde{C}_k &= \sum_{j=0}^{N-1} \omega_N^{kj} C_j = \sum_{j=0}^{N-1} \omega_N^{kj} \left(\sum_{i=0}^{N-1} \omega_N^{ji} c_i\right) \\ &= \sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} \omega_N^{j(k+i)}\right) c_i. \end{aligned} \tag{8.10}$$

Here

$$\sum_{j=0}^{N-1} \omega_N^{j(k+i)} = \begin{cases} N & \text{ha } N \mid k+i \\ 0 & \text{ha } N \nmid k+i \end{cases} = \begin{cases} N & \text{ha } i = N-k \\ 0 & \text{ha } i \neq N-k \end{cases}$$

Writing this in (8.10), we get $\widetilde{C}_k = NC_{N-k}$, which proves (8.9).

In other words, the effectiveness of the above algorithm depends on how quickly we can calculate the DFT of a sequence.

Our next goal is to provide a fast algorithm for calculating the DFT of a N-long sequence, as well as to calculate the time required for this algorithm. For the sake of simplicity, the length of the sequence N should always be a power of two (this can easily be assumed, since we gave N as a power of two at the beginning of the chapter). The time required for Fourier transform of the sequence to be determined shortly is denoted by T_N in the case of a N long sequence.

Our algorithm for computing the DFT will be recursive. For a $N = 2^0 =$ 1-long sequence, the algorithm is very simple, since the sum in (8.8) has only one term, and thus the time required is $T_1 = 1$. Let's look at the recursive step. Let's assume that the algorithm has already been given in the cases $N = 2^0, 2^1, \dots, 2^{r-1}$ with $T_1, T_2, T_4, \dots, T_{2^{r-1}}$ time requirements. Now we would like to give the algorithm in the case of $N = 2^r$, and after giving the algorithm, we would like to estimate its time requirement T_{2^r}.

Let our sequence be $(a_{N-1}, a_{N-2}, \dots, a_1, a_0)$ and the primitive N-th root of unity be ω_N, with which we define the DFT, i.e., the sequence $(A_{N-1}, A_{N-2}, \dots, A_1, A_0)$, that is

$$A_k = \sum_{j=0}^{N-1} \omega_N^{kj} a_j.$$

Consider the polynomial assigned to the sequence $(a_{N-1}, a_{N-2}, \dots, a_1, a_0)$ i.e.,

$$\overline{a}(x) = a_{N-1}x^{N-1} + a_{N-2}x^{N-2} + \cdots + a_1 x + a_0,$$

and write it as the sum of two polynomials, namely

$$\overline{a}(x) = \overline{a_0}(x^2) + x\overline{a_1}(x^2),$$

where

$$\begin{aligned}\overline{a_0}(y) &\overset{\text{def}}{=} a_{N-2}y^{N/2-1} + a_{N-4}y^{N/2-2} + a_{N-6}y^{N/2-3} + \cdots + a_2y + a_0\\ \overline{a_1}(y) &\overset{\text{def}}{=} a_{N-1}y^{N/2-1} + a_{N-3}y^{N/2-2} + a_{N-5}y^{N/2-3} + \cdots + a_3y + a_1.\end{aligned}$$

From the coefficients of the two polynomials, we can form two $N/2$ long sequences. These are:

$$(a_{N-2}, a_{N-4}, a_{N-6}, \ldots, a_2, a_0) \tag{8.11}$$

and

$$(a_{N-1}, a_{N-3}, a_{N-5}, \ldots, a_3, a_1). \tag{8.12}$$

Since N is even (moreover a power of two), if ω_N is a primitive N-th root of unity, then ω_N^2 is a primitive $N/2$-th root of unity. Calculate the DFT of the $N/2$-long sequences (8.11) and (8.12). Then we get the following two sequences:

$$\begin{gathered}\left(A^{(0)}_{\frac{N}{2}-1}, A^{(0)}_{\frac{N}{2}-2}, \ldots, A^{(0)}_1, A^{(0)}_0\right)\\ \left(A^{(1)}_{\frac{N}{2}-1}, A^{(1)}_{\frac{N}{2}-2}, \ldots, A^{(1)}_1, A^{(1)}_0\right)\end{gathered}$$

Then

$$\begin{aligned}A^{(0)}_k &= \sum_{j=0}^{N/2-1} \left(\omega_N^2\right)^{kj} a_{2j}\\ A^{(1)}_k &= \sum_{j=0}^{N/2-1} \left(\omega_N^2\right)^{kj} a_{2j+1}\end{aligned}$$

Using the polynomials $\overline{a_0}(y)$ and $\overline{a_1}(y)$ we get

$$A^{(0)}_k = \overline{a_0}(\omega_N^{2k}), \quad A^{(1)}_k = \overline{a_1}(\omega_N^{2k}).$$

Since $\overline{a}(x) = \overline{a_0}(x^2) + x\overline{a_1}(x^2)$, writing $x = \omega_N^k$ we get

$$\overline{a}(\omega_N^k) = \overline{a_0}(\omega_N^{2k}) + \omega_N^k \overline{a_1}(\omega_N^{2k}).$$

That is

$$A_k = A_k^{(0)} + \omega_N^k A_k^{(1)}.$$

Since $\omega_N^{N/2} = -1$ (the reasoning is simple: $0 = \omega_n^N - 1 = (\omega_N^{N/2} - 1)(\omega_N^{N/2} + 1)$ but ω_N is primitive N-th root of unity, so $\omega_N^{N/2} - 1 \neq 0$, i.e., $\omega_N^{N/2} + 1 = 0$). Therefore, if k is replaced by $N/2 + k$ we get:

$$A_{N/2+k} = \overline{a}(\omega_N^{N/2+k}) = \overline{a_0}(\omega_N^{N+2k}) + \omega_N^{N/2+k}\overline{a_1}(\omega_N^{N+2k}).$$

Using $\omega_N^N = 1$ and $\omega_N^{N/2} = -1$ we get

$$A_{N/2+k} = \overline{a_0}(\omega_N^{2k}) - \omega_N^k \overline{a_1}(\omega_N^{2k}) = A_k^{(0)} - \omega_N^k A_k^{(1)}.$$

In summary: The DFT's of the sequences are calculated in $2T_{N/2}$ time, after these N pieces of multiplications and additions are used to obtain the DFT of the original sequence. In terms of time requirements, we get the following recursion

$$\begin{aligned} T_1 &= 1 \\ T_N &\leq 2T_{N/2} + N. \end{aligned}$$

From this, it can be easily proved by complete induction that for $N \geq 2$

$$T_N \leq 2N \log_2 N.$$

During the algorithm given above, we have not taken into account that the N-th roots of unity over complex numbers are infinite decimal fractions. According to Knuth [2], in the case of $N = 2^{t+1}$, it is enough to calculate to $6(t+1)$ decimal places during the algorithm. The time requirement of the resulting algorithm is $O(n \log n \log\log n \dots \log_k n)$, where this time $\log_k n$ denotes the logarithm function iterated k times so that $\log_k n = \mathit{log} \log \dots \log n$

and k is the smallest positive integer for which $\log_k n = \log\log\ldots\log n < 2$. Schönhage and Strassen [3] proposed the following: instead of $\mathbb{C}$, we use the algorithm $\mathbb{Z}_{2^s+1}$ where 2 primitive $2s$-th unit roots with suitable s. The running time thus achieved is $O(n \log n \log\log n)$. We omit the details in this lecture notes.

A few notes for programmers: For small numbers, the schoolbook-method is the best method. For slightly larger numbers, the Karatsuba-Ofman method, for even larger numbers, the Toom-Cook method, and for the largest numbers, the FFT-based multiplication suggested by Schönhage and Strassen. FFT multiplication becomes the quickest at a few tens of thousands of digits, and it is actually the most common multiplication for huge numbers in practice.

References

[1] A. Das, *Computational Number Theory*, Discrete Mathematics and its Applications, CRC Press, 2013.

[2] D. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical algorithms*, Pearson Education, Boston, United States 1997, 4. fejezet.

[3] A. Schönhage, V. Strassen, *Schnelle multiplikation großer zahlen*, Computing 7 (1971), 281-292.

9 Primality tests

Primality tests are algorithms that determine whether a given number is prime or not. In contrast to factorization, primality tests do not provide the prime factorization of a given number, they are just concerned with determining whether the integer in question is composite or prime. There are many different algorithms for primality testing, and the most modern primality tests are polynomial time algorithms. However, we conjecture that factorization is a non-polynomial algorithm.

The most effective in practical applications are the so-called probability primality tests, which, however with not absolute certainty, but can determine whether an integer number is prime or composite with a very high probability. We only briefly describe a polynomial-time and deterministic primality test in the final section of the chapter because it is substantially more complicated than the previous primality tests. In terms of running time, this deterministic test is slower than the probabilistic tests, but it is still a polynomial time algorithm.

9.1 Trial division

The simplest primality test is called the trial division. Let the number given as input be n, and we have to decide whether n is prime. During the trial division, we check whether d is a divisor of n for the numbers $d = 2, 3, 4, \cdots [\sqrt{n}]$. If we find a divisor, n is composite. If none of these d is a divisor of n, then n is prime. The basis of the above reasoning is that every composite number n has a divisor smaller than or equal to $[\sqrt{n}]$, because if $n = ab$, then a or b is less than $\sqrt{n}$. Trial division is not a polynomial algorithm, it takes $[\sqrt{n}](\log n)^2$ bit operations. The algorithm is based on Eratosthenes' sieve, which was first mentioned in Nicomákhos of Gerasa's Introduction to Arithmetic [1] (2nd century BC), and he attributed the sieve to Eratosthenes of Cyrene. If we wish to find all the primes up to a certain limit, the Eratosthenes sieve is still the fastest.

References

[1] R. Hoche and ed., *Nicomachi Geraseni Pythagorei Introductionis Arithmeticae Libri II*, Leipzig: B.G. Teubner, 1866, p. 31.

9.2 Fermat primality test

The Fermat primality test is based on Fermat's little theorem.

Theorem 9.1. (Fermat's little theorem) *If p is a prime and $(a, p) = 1$ for an integer a, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Next we describe the test based on [1] and [2]. The Fermat primality test is as follows: Let n be the input number.

Step 1: Take a randomly chosen integer a for which $n \nmid a$.

Step 2: Check if

$$a^{n-1} \equiv 1 \pmod{n}$$

holds or not. If $a^{n-1} \not\equiv 1 \pmod{n}$, then according to the little-Fermat theorem, n is composite. If $a^{n-1} \equiv 1 \pmod{n}$, we return to Step 1 with another randomly chosen integer a .

If we repeat the steps of the test for enough randomly chosen a and we always get that $a^{n-1} \equiv 1 \pmod{n}$, then a is "probably" a prime number. If for a single a we get that $a^{n-1} \not\equiv 1 \pmod{n}$, then (based on the little-Fermat theorem) n must be a composite number. Since this test never says that a number is definitely prime, it is called a probabilistic primality test.

Today, the most commonly used primality tests are probabilistic primality tests, since their running time is significantly faster than the running time of deterministic primality tests. Deterministic tests will be discussed later in the chapter. Now let's return to the Fermat primality test. This test never says that a number n is definitely prime. But is the inversion of the Fermat's little theorem true at all, i.e., is the following true?

Question: If for a given integer n, for every integer $(a, n) = 1$ we have

$$a^{n-1} \equiv 1 \pmod{n},$$

then is it certain that n is prime?

The answer to this question is no. For example, let $n = 561 = 3 \cdot 11 \cdot 17$. We prove that then $a^{560} \equiv 1 \pmod{561}$ for every $(a, 561) = 1$. But now 561 is a composite number, since $561 = 3 \cdot 11 \cdot 17$. Let's see the proof:

Let $(a, 561) = 1$. Then by $561 = 3 \cdot 11 \cdot 17$ we also have $(a, 3) = (a, 11) = (a, 17) = 1$. Write Fermat's little theorem for $p = 3$:

$$a^2 \equiv 1 \pmod{3}.$$

Raising the above congruence to the 280th power:

$$a^{560} \equiv 1 \pmod{3}.$$

That is $3 \mid a^{560} - 1$.

Write Fermat's little theorem for $p = 3$:

$$a^{10} \equiv 1 \pmod{11}.$$

Raising the above congruence to the 56th power:

$$a^{560} \equiv 1 \pmod{11}.$$

That is $11 \mid a^{560} - 1$.

Write Fermat's little theorem for $p = 17$:

$$a^{16} \equiv 1 \pmod{17}.$$

Raising the above congruence to the 35th power:

$$a^{560} \equiv 1 \pmod{17}.$$

That is $17 \mid a^{560} - 1$. So $3 \cdot 11 \cdot 17 = 561 \mid a^{560} - 1$, and this completes the proof.

Definition 9.2. *A positive integer n is called a* ***Carmichael number****, if n is a composite and for all $(a, n) = 1$ we have*

$$a^{n-1} \equiv 1 \pmod{n}.$$

The smallest Carmichael number is 561. We will study in more detail Carmichael numbers in the next section. Returning to the Fermat primality test, we can see that it determines the Carmichael numbers likely to be primes despite they are composite. As a result, the Fermat primality test is not considered one of the most optimal primality tests. Nonetheless, the test is only rarely incorrect since, according to Pomerence [3], the number of Carmichael numbers smaller than x is less than $x^{1-o(1)}$.

Assume that the integer n is a composite but not a Carmichael number. In other words, there is an integer b for which

$$b^{n-1} \not\equiv 1 \pmod{n}.$$

Next, we divide the integers $(a, n) = 1$ into two classes. The number a is called a Fermat liar if

$$a^{n-1} \equiv 1 \pmod{n}.$$

The name is justified by the fact that the Fermat's little theorem holds for the number a, even though the modulus n is a composite number. Thus, trying the Fermat primality test on this a, the test (wrongly) predicts that n is prime. The number a is called a witness if

$$a^{n-1} \not\equiv 1 \pmod{n}.$$

This definition is justified by the fact that when we apply the Fermat primality test to this a, we find that n is a composite. That is, the number a ”witnesses” to the fact that n is a composite.

Theorem 9.3. *Let n be a complex number that is not a Carmichael number. Then at least half of the numbers $(a, n) = 1$, $1 \le a \le n$ are witnesses.*

This theorem states that when the test is run for a fixed a, the probability that the test is incorrect (that is, that it determines n to be likely prime, however it is composite) is at most 50%. If we run the test on two pieces of a, the probability that the test is incorrect in both cases is $\left(\frac{1}{2}\right)^2$, i.e., 25%. If we run the test with r distinct a's, the error probability is $\left(\frac{1}{2}\right)^r$. For larger r, this is an exceedingly low likelihood. Even with 300 pieces of a, the error probability is less than $\left(\frac{1}{2}\right)^{300}$, which is less than the reciprocal of the total number of atoms in the universe... Let's see the proof of the theorem.

Proof of Theorem 9.3. Since n is not a Carmichael number, there is at least one witness. Let this be b. Then

$$b^{n-1} \not\equiv 1 \pmod{n}. \tag{9.1}$$

List the Fermat liars: $a_1, a_2, \dots, a_r$. Then

$$a_i^{n-1} \equiv 1 \pmod{n} \qquad (1 \le i \le r).$$

However, based on (9.1):

$$(ba_i)^{n-1} \not\equiv 1 \pmod{n}, \qquad (1 \le i \le r).$$

In other words, $ba_1, ba_2, \dots, ba_r$ are witnesses as well. That is, there are at least as many witnesses as liars, and thus we have proved the theorem.

References

[1] D. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical algorithms*, Pearson Education, Boston, United States, 1997, Chapter 4.

[2] N. Koeblitz, *A Course in Number Theory and Cryptography*, 2nd Edition, Springer, 1994.

[3] C. Pomerance, *On the distribution of pseudoprimes*. Math. Comp. 37 (1981), 587–593

9.3 Carmichael numbers

The Carmichael numbers were already defined in the previous chapter, but it's worth repeating, so I'll describe the definition now.

Definition 9.4. *A positive integer n is called a* ***Carmichael number****, if n is a composite and for all $(a, n) = 1$ we have*

$$a^{n-1} \equiv 1 \pmod{n}.$$

As the integers increase, Carmichael numbers become less frequent on the number line. To illustrate this, we show a table. Let $C(x)$ denote the number of Carmichael numbers between 1 and x. Then:

bigskip

n	1	2	3	4	5	6	7	8	9	10	11	12
$C(10^n)$	0	0	1	7	16	43	105	255	646	1547	3605	8241

n	13	14	15	16	17	18	19
$C(10^n)$	19279	44706	105212	246683	585355	1401644	3381806

For a very long time it was conjectured that there were an infinite number of Carmichael numbers. Finally Alford, Granville, and Pomerance [1] confirmed this and proved that

$$C(x) > x^{2/7}.$$

This was sharpened by Harman [3]:

$$C(x) > x^{0.33336704}.$$

In 1956, Erdős [2] conjectured that $C(x) < x^{1-o(1)}$. Erdős' heuristic argument was confirmed by Pomerance [5] in 1981. He proved that

$$C(x) < x^{1-\frac{(1+o(1)\log\log\log x)}{\log\log x}}.$$

The most important result concerning the form of Carmichael numbers is the Korselt criterion [4].

Theorem 9.5. (A. Korselt, 1899) *A positive integer n is a Carmichael number if and only if n is square-free and if $p-1 \mid n-1$ holds for every prime divisor p of n.*

Proof of Theorem 9.5. First, we see that if n is a square-free number and for every prime divisor p of n we have $p-1 \mid n-1$, then n is a Carmichael number. Indeed, let n have the following prime factor decomposition:

$$n = p_1 p_2 \dots p_r,$$

where the primes $p_1, p_2, \dots, p_r$ are distinct. Let

$$(a, n) = 1.$$

Then

$$(a, p_1) = (a, p_2) = \dots = (a, p_r) = 1.$$

Since $(a, p_i) = 1$ by Fermat's little theorem

$$a^{p_i - 1} \equiv 1 \pmod{p_i}. \tag{9.2}$$

Since $p_i - 1 \mid n - 1$, there exists a positive integer k_i such that

$$n - 1 = k_i(p_i - 1)$$

Thus, by raising the congruence (9.2) to the k_ith power, we get

$$\begin{aligned}
a^{k_i(p_i-1)} &\equiv 1 \pmod{p_i} \\
a^{n-1} &\equiv 1 \pmod{p_i} \\
p_i &\mid a^{n-1} - 1. \qquad (9.3)
\end{aligned}$$

Since (9.3) holds for $i = 1, 2, \ldots, r$, thus

$$\begin{aligned}
p_1 p_2 \cdots p_r &\mid a^{n-1} - 1 \\
n &\mid a^{n-1} - 1 \\
a^{n-1} &\equiv 1 \pmod{n}. \qquad (9.4)
\end{aligned}$$

Then the congruence (9.4) holds for every integer $(a, n) = 1$, so a is a Carmichael number.

Then we turn to the second part of the proof of the theorem, namely, if n is a Carmichael number, then n is square-free on the one hand, and for every prime divisor of n we have $p - 1 \mid n - 1$ on the other hand. First, we see that n is square-free. We prove this indirectly, we assume that there exists a prime p for which n is of the form

$$n = p^k m$$

where $k \geq 2$ is an integer and $(m, p) = 1$ for the integer m. Consider the integer a for which

$$a \equiv 1 + p \pmod{p^2} \qquad \text{and} \qquad a \equiv 1 \pmod{m}.$$

By the Chinese remainder theorem, such an integer a exists. Based on the binomial theorem

$$\begin{aligned}
&a^{n-1} \equiv (1+p)^{n-1} \pmod{p^2} \\
&a^{n-1} \equiv 1 + \binom{n-1}{1} p + \binom{n-1}{2} p^2 + \cdots + \binom{n-1}{n-1} p^{n-1} \pmod{p^2}
\end{aligned}$$

$$a^{n-1} \equiv 1 + (n-1)p \pmod{p^2}. \tag{9.5}$$

On the other hand, n is a Carmichael number, so

$$\begin{aligned} a^{n-1} &\equiv 1 \pmod{n} \\ n &\mid a^{n-1} - 1 \\ p^2 &\mid a^{n-1} - 1 \\ a^{n-1} &\equiv 1 \pmod{p^2}. \end{aligned}$$

Comparing this with (9.5), we get that

$$\begin{aligned} 1 &\equiv 1 + (n-1)p \pmod{p^2} \\ p^2 &\mid (n-1)p \\ p &\mid n-1, \end{aligned}$$

which contradicts $p \mid n$. By this, we realized that n has no prime square divisor, i.e., n is a squarefree number.

We can continue the proof by that for every prime divisor p of n we have $p-1 \mid n-1$. Write n in the form $n = pm$. Since n is squarefree number, we can assume that $(m, p) = 1$. Consider a primitive root g modulo p. Then $g, g^2, g^3, \ldots, g^{p-2} \not\equiv 1 \pmod{p}$, but $g^{p-1} \equiv 1 \pmod{p}$ since the order of g is $p-1$. That is, the infinite sequence $1, g, g^2, \ldots$ will be periodic modulo p, and the period length is $p-1$, so

$$g^k \equiv 1 \pmod{p} \Leftrightarrow p - 1 \mid k. \tag{9.6}$$

Assume that for all $(a, n) = 1$ we have

$$a^{n-1} \equiv 1 \pmod{n}.$$

By the Chinese remainder theorem, there exists an integer a for which

$$a \equiv g \pmod{p} \quad \text{and} \quad a \equiv 1 \pmod{m}.$$

Since g is a primitive root, $(g,p)=1$ and thus $(g+tp,p)=1$ also holds for every integer t. That is, $(a,p)=1$. Similarly, by $a\equiv 1 \pmod{m}$ we also have $(a,m)=1$ also holds. Thus $(a,n)=(a,pm)=1$. Then

$$\begin{aligned} a^{n-1} &\equiv 1 \pmod{n} \\ n &\mid a^{n-1}-1 \\ pm &\mid a^{n-1}-1 \\ p &\mid a^{n-1}-1 \\ a^{n-1} &\equiv 1 \pmod{p} \\ g^{n-1} &\equiv 1 \pmod{p}. \end{aligned}$$

Then, using (9.6), we get that $p-1 \mid n-1$, which completes the proof.

References

[1] W. R. Alford, A. Granville and C. Pomerance, *There are infinitely many Carmichael numbers*, Annals of Mathematics, Second Series, 139, No. 3 (1994), 703-722.

[2] P. Erdős, *On pseudoprimes and Carmichael numbers*, Publ. Math. Debrecen 4 (1956), 201–206.

[3] G. Harman, *On the number of Carmichael numbers up to x*, Bulletin of the London Mathematical Society 37 (2005), 641–650.

[4] A. R. Korselt, *Problème chinois*, L'intermédiaire des mathématiciens 6 (1899), 142–143.

[5] C. Pomerance, *On the distribution of pseudoprimes*. Math. Comp. 37 (1981), 587–593.

9.4 An example of the Fermat primality test

During the Fermat primality test, we must compute large powers modulo n. Consider the following example:

Question. Is $2^{11} - 1 = 2047$ prime or not? Let's test it with the Fermat primality test!

Solution. The algorithm is as follows:

Step 1. Take a random element $a \in \mathbb{Z}_{2047}$.
Step 2. Let's check that

$$a^{2046} \equiv 1 \pmod{2047} \tag{9.7}$$

holds or not? If it does not hold, then 2047 is composite, if it holds, return to Step 1 with another number a. If the congruence (9.7) holds for many a's, then the 2047 is probably prime.

During the first step of the algorithm, we choose a random number $a \in \mathbb{Z}_{2047}$, $2047 \nmid a$. For the sake of simplicity, let's assume that $a = 2$ (although, the probability of this is extremely small 1/2046...). During Step 2 of the algorithm, we must check whether the congruence

$$2^{2046} \equiv 1 \pmod{2047} \tag{9.8}$$

holds or not. Note that $2^{11} = 2048 \equiv 1 \pmod{2047}$. By this

$$2^{2046} = \left(2^{11}\right)^{186} \equiv 1 \pmod{2047}.$$

That is, (9.8) is indeed holds. So far it has not been revealed that 2047 is a composite number. That is, 2 is a Fermat liar. If $a^{2046} \equiv 1 \pmod{2047}$ holds for many a's, then we could say that it is probable that 2047 is a prime number or a Carmichael number. But for now, we only studied the case $a = 2$. Let's try again with another random a. As before, we can choose all $a \in \mathbb{Z}_{2047}$, $2047 \nmid a$ with equal probability, but now for the sake of simplicity, let's assume that $a = 3$. (In fact, it often happens in practice that when programmers are looking for random numbers that are not secret, i.e., they are not used for cryptographic purposes, they simply take consecutive natural numbers... This detracts a lot from the true effectiveness of the random algorithm. Even if we provide similar example, we strongly advise the reader to avoid such simplifications when they want to program...) In case of $a = 3$, we have to check that

$$3^{2046} \equiv 1 \pmod{2047} \tag{9.9}$$

holds or not. In other words, we have to calculate a very large power using modular exponentiation. This can be done by repeated squaring. To do this, first write 2046 as the sum of two powers.

$$2046 = 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2.$$

Let's make a table with the values of the powers of $3^{2^k} \pmod{2047}$.

	3^1	3^2	3^4	3^8	3^{16}	3^{32}	3^{64}	3^{128}	3^{256}	3^{512}	3^{1024}
mod 2047	3	9	81	420	358	1250	639	968	1545	223	601

Based on this:

$$3^{2046} \equiv 3^{1024} \cdot 3^{512} \cdot 3^{256} \cdot 3^{128} \cdot 3^{64} \cdot 3^{32} \cdot 3^{16} \cdot 3^{8} \cdot 3^{4} \cdot 3^{2}$$
$$601 \cdot 223 \cdot 1545 \cdot 968 \cdot 639 \cdot 1250 \cdot 358 \cdot 420 \cdot 81 \cdot 9 \cdot 3 \pmod{2047}.$$

By doing the multiplications in succesion and always reducing the result mod 2047, finally we get

$$3^{2046} \equiv 1013 \not\equiv 1 \pmod{2047}.$$

That is, 2047 is not a prime number, and 3 was a Fermat witness, while 2 was a Fermat liar. Indeed, 2047 is composite, since $2047 = 11 \cdot 23$. Usually, it is difficult to find the prime factor decomposition of large numbers, while the running time of the Fermat test is fast: $O(k \cdot \log^3 n)$, where k is the number of rounds, i.e., the testing is based on k different random a's.

9.5 Soloway-Strassen primality test

In the previous section we talked about the Fermat primality test. We have seen that the test does not work for Carmichael numbers, since the test determines Carmichael numbers as problable primes. To understand the following test, you need the know the definitions of Legendre and Jacobi symbols, see Chapter 3. We have seen that if n is a prime number, then the value of the Jacobian symbol $\left(\frac{a}{n}\right)$ is the same as the value of the Legendre symbol $\left(\frac{a}{n}\right)$. The Soloway-Strassen primality test [2],[3] is based on the following theorem:

Theorem 9.6. (Euler-lemma) *For any prime number p and integer a with $(a,p)=1$ we have*

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

where $\left(\frac{a}{p}\right)$ denotes the Legendre symbol.

Proof of Theorem 9.6. By Fermat's little theorem

$$\begin{aligned} a^{p-1} &\equiv 1 \pmod{p} \\ p &\mid a^{p-1} - 1 \\ p &\mid \left(a^{(p-1)/2} - 1\right)\left(a^{(p-1)/2} + 1\right), \end{aligned}$$

thus $p \mid a^{(p-1)/2} - 1$ or $p \mid a^{(p-1)/2} + 1$. That is

$$a^{(p-1)/2} \equiv 1 \pmod{p} \quad \text{or} \quad a^{(p-1)/2} \equiv -1 \pmod{p}. \tag{9.10}$$

If $\left(\frac{a}{p}\right) = 1$, then the congruence

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

is solvable. Denote a solution of the above congruence by x_0. Then

$$x_0^2 \equiv a \pmod{p}.$$

By Fermat's little theorem:

$$a^{(p-1)/2} \equiv \left(x_0^2\right)^{(p-1)/2} = x_0^{p-1} \equiv 1 \pmod{p}.$$

Now $\left(\frac{a}{p}\right) = 1$, thus

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

Next we turn to the proof of the case $\left(\frac{a}{p}\right) = -1$. Lagrange's theorem states the following:

Lemma 9.7. (Lagrange) *If p is a prime number, $f(x) \in \mathbb{Z}[x]$, then either all coefficients of $f(x)$ are divisible by p, or*

$$f(x) \equiv 0 \pmod{p}$$

congruence has at most $\deg f$ *incongruent solutions, where* $\deg f$ *denotes the degree of the polynomial $f(x)$.*

We will not prove Lagrange's theorem here, its proof can be found in any book dealing with elementary number theory. Based on Lagrange theorem, we know that the congruence

$$x^{(p-1)/2} \equiv 1 \pmod{p}$$

has at most $(p-1)/2$ solutions. We have seen that for $\left(\frac{a}{p}\right) = 1$

$$a^{(p-1)/2} \equiv 1 \pmod{p},$$

thus, the solutions of the congruence $x^{(p-1)/2} \equiv 1 \pmod{p}$ are the quadratic residues a. We know that there are a total of $(p-1)/2$ quadratic residues, thus we have found all the solutions of the $x^{(p-1)/2} \equiv 1 \pmod{p}$ congruence, these are the quadratic residues. So for $\left(\frac{a}{p}\right) = -1$ we have $a^{(p-1)/2} \not\equiv 1 \pmod{p}$. Then by (9.10)

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

Since now $\left(\frac{a}{p}\right) = -1$, we also proved

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

This completes the proof of Euler-lemma.

Then we describe the **Soloway-Strassen primality test** [2], [3]. The test was based on an idea of Artjuhov [1].

So we want to determine whether a given odd number n is prime or composite.

Step 1. We take a random a for which $(a, n) = 1$.

Step 2. By the repeated squaring algorithm, we calculate $a^{(n-1)/2} \pmod{n}$.

Step 3. We calculate the value of the Jacobi symbol $\left(\frac{a}{n}\right)$.

Step 4. We check whether the congruence $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$ holds or not. If the answer is NO, then it is certain that n is composite and a is called an <u>Euler witness</u>. If the answer is YES we return to Step 1 with another a
A new definition: if n is composite and $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$, we called a as an <u>Euler-liar</u>.

If we look at the above algorithm for many a's, and it always comes out that $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$, then the test determines n as a probable prime.

Example. The number -1 is always Euler-liar, since it always holds for the Jacobi symbol that

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

Theorem 9.8. *For every composite odd n, at least half of the reduced residue classes* mod n *are Euler witnesses.*

Proof of the Theorem 9.8.. Let the Euler liars be $\ell_1, \ell_2, \dots, \ell_k$, and the Euler witnesses be $w_1, w_2, \dots, w_m$. Then we need to prove that $k \le m$. First, we just prove that there is at least one Euler witness. If n is not squarefree, then n cannot be a Carmichael number, i.e., there exists a residue a for which $(a, n) = 1$ and $a^{n-1} \not\equiv 1 \pmod{n}$. Thus

$$a^{(n-1)/2} \not\equiv \pm 1 \pmod{n}$$

(because if $a^{(n-1)/2} \equiv \pm 1 \pmod{n}$, then after squaring we would get $a^{n-1} \equiv 1 \pmod{n}$ which is contradiction). However, the value of the Jacobi symbol $\left(\frac{a}{n}\right)$ is ± 1, that is

$$a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}.$$

Then a is an Euler witness. If n is squarefree, write the prime factor decomposition of n that is

$$n = p_1 p_2 \dots p_r,$$

where p_i's are distinct primes.

Fix a quadratic non-residue mod p_1, and denote it by m. That is, $\left(\frac{m}{p_1}\right) = -1$. Let a be the solution of the following simultaneous congruence system:

$$x \equiv m \pmod{p_1} \qquad x \equiv 1 \pmod{p_2 p_3 \dots p_r}.$$

By the Chinese remainder theorem, such a exists and it is unique mod $p_1 p_2 \dots p_r$. That is

$$a \equiv m \pmod{p_1} \qquad a \equiv 1 \pmod{p_2 p_3 \dots p_r}.$$

Calculate the value of the Jacobi symbol $\left(\frac{a}{n}\right)$:

$$\begin{aligned}\left(\frac{a}{n}\right) &= \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\dots\left(\frac{a}{p_r}\right)\\ &= \left(\frac{m}{p_1}\right)\left(\frac{1}{p_1}\right)\dots\left(\frac{1}{p_r}\right)\\ &= -1 \cdot 1 \cdots 1\end{aligned}$$

$$= -1.$$

If

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

holds, then

$$\begin{aligned}
a^{(n-1)/2} &\equiv -1 \pmod{n} \\
n &\mid a^{(n-1)/2} + 1 \\
p_2 &\mid a^{(n-1)/2} + 1 \\
a^{(n-1)/2} &\equiv -1 \pmod{p_2} \\
1^{(n-1)/2} &\equiv -1 \pmod{p_2},
\end{aligned}$$

which is contradiction. That is, $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$. So a is an Euler witness. By this, we proved that for every composite number there is at least one Euler witness. Let the Euler liars be $\ell_1, \ell_2, \ldots, \ell_k$, and the Euler witnesses be $w_1, w_2, \ldots, w_m$. We have already seen that $m \geq 1$, however, in order to prove the theorem, we need that $k \leq m$. Let's fix an Euler witness, say w_1. We prove that then $w_1\ell_1, w_1\ell_2, \ldots, w_1\ell_k$ are also Euler witnesses. Since the above elements are incongruent modulo n, it follows from this statement that $m \geq k$. By the multiplicity of the Jacobi symbol

$$\left(\frac{w_1\ell_i}{n}\right) = \left(\frac{w_1}{n}\right)\left(\frac{\ell_i}{n}\right). \tag{9.11}$$

Since ℓ_i is Euler-liar then $\left(\frac{\ell_i}{n}\right) \equiv \ell_i^{(n-1)/2} \pmod{n}$. However, w_1 is an Euler witness, so $\left(\frac{w_1}{n}\right) \not\equiv w_1^{(n-1)/2} \pmod{n}$. Comparing the above with (9.11), we get

$$\left(\frac{w_1\ell_i}{n}\right) = \left(\frac{w_1}{p}\right)\left(\frac{\ell_i}{p}\right) \not\equiv w_1^{(n-1)/2}\ell_i^{(n-1)/2} = (w_1\ell_i)^{(n-1)/2} \pmod{n},$$

that is, $w_1\ell_i$ is indeed an Euler witness. This completes the proof of the theorem.

Based on the above theorem, if we run the test on k pieces of a, the probability that n is a composite number, but the test determines as a probable prime number is less than 2^{-k}. This is an extremely small probability even for $k = 100$.

References

[1] M. M. Artjuhov, *Certain criteria for primality of numbers connected with the little Fermat theorem*, Acta Arith., 12 (1966/67), 355–364.

[2] R. M. Solovay, V. Strassen, *A fast Monte-Carlo test for primality*, SIAM Journal on Computing. 6 (1) (1977), 84–85.

[3] R. M. Solovay, V. Strassen, *Erratum: A fast Monte-Carlo test for primality*, SIAM Journal on Computing. 7 (1) (1978) 118.

9.6 An example of the Soloway-Strassen primality test

Below, we determine whether $n = 209$ is prime or composite using the Soloway-Strassen primality test.

Step 1. Take a random a, e.g., say $a = 153$.

Step 2. Calculate $a^{(n-1)/2} \pmod{n}$ using the repeated squaring algorithm.

$$153^{(209-1)/2} \equiv 153^{104} \equiv 153^{64} \cdot 153^{32} \cdot 153^{8} \pmod{209}.$$

Let's make a table with the values $153^{2^k} \pmod{209}$.

	153^1	153^2	153^4	153^8	153^{16}	153^{32}	153^{64}
mod 209	153	1	1	1	1	1	1

Based on these

$$153^{(209-1)/2} \equiv 1 \cdot 1 \cdot 1 \equiv 1 \pmod{209}.$$

Step 3. Let's calculate the Jacobi symbol $\left(\frac{a}{n}\right)$:

$$\begin{aligned}\left(\frac{153}{209}\right) &= (-1)^{(153-1)/2\cdot(209-1)/2}\left(\frac{209}{153}\right) = \left(\frac{209}{153}\right) = \left(\frac{56}{153}\right)\\ &= \left(\frac{2}{153}\right)^3\left(\frac{7}{153}\right) = (-1)^{(153^2-1)/8}\left(\frac{7}{153}\right) = \left(\frac{7}{153}\right)\\ &= (-1)^{(7-1)/2\cdot(153-1)/2}\left(\frac{153}{7}\right) = \left(\frac{153}{7}\right) = \left(\frac{6}{7}\right)\\ &= \left(\frac{2}{7}\right)\left(\frac{3}{7}\right) = (-1)^{(7^2-1)/8}\left(\frac{3}{7}\right) = \left(\frac{3}{7}\right)\\ &= (-1)^{(3-1)/2\cdot(7-1)/2}\left(\frac{7}{3}\right) = -\left(\frac{7}{3}\right) = -\left(\frac{1}{3}\right)\\ &= -1.\end{aligned}$$

That is, $153^{(209-1)/2} \not\equiv \left(\frac{153}{209}\right) \pmod{209}$. Thus 153 is an Euler witness, and so 209 is a composite number (actually $209 = 11 \cdot 19$).

9.7 Miller-Rabin primality test

The Miller-Rabin primality test was first discovered by Gary L. Miller [5] in 1976. Miller's version of the test was deterministic, but the reliability of the test was based on an unproven conjecture, the general Riemann hypothesis. Later Michael O. Rabin [6] modified the test in 1980 in such a way that it no longer depended on unproven conjecture, but thus the test was no longer deterministic, but a probabilistic primality test.

First, we note that if p is prime, then from congruence

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

follows that

$$x \equiv \pm 1 \pmod{p}.$$

Let's see the proof of this:

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

$$\begin{aligned}
&p \mid x^2 - 1\\
&p \mid (x-1)(x+1)\\
&p \mid x-1 \text{ or } p \mid x+1\\
&x \equiv \pm 1 \pmod{p}.
\end{aligned}$$

It is important to note that in the above proof it is a necessary condition that p is a prime. The statement is not even true for the most composite numbers. (Take 8 for example. The congruence $x^2 \equiv 1 \pmod{8}$ has 4 solutions: $x \equiv \pm 1,\ \pm 3 \pmod{8}$.)

Below the algorithm of the Miller-Rabin primality test is described. The test works for odd numbers. Assume that n is odd and write

$$n - 1 = 2^k r,$$

where $k \in \mathbb{N}$ and r is an odd positive integer. By Fermat's little theorem, if n is prime, then for every number $a \in \mathbb{Z}$, where $a \not\equiv 0 \pmod{n}$ we have

$$a^{n-1} \equiv 1 \pmod{n}.$$

Here $n - 1 = 2^k r$, that is

$$a^{2^k r} \equiv 1 \pmod{n}. \tag{9.12}$$

If n is prime, then for $x^2 \equiv 1 \pmod{n}$ we have $x \equiv \pm 1 \pmod{n}$, so it follows from (9.12) that

$$a^{2^{k-1} r} \equiv \pm 1 \pmod{n}.$$

In case $a^{2^{k-1} r} \equiv 1 \pmod{n}$, then $a^{2^{k-2} r} \equiv \pm 1 \pmod{n}$ and so on...

Based on the above, the residue a modulo n, where $a \not\equiv 0 \pmod{n}$, is called a Miller-Rabin liar if

$$a^r \equiv 1 \pmod{n}$$

or the sequence

$$a^r, a^{2r}, a^{4r}, \ldots, a^{2^{k-1} r} \pmod{n}$$

contains the residue -1 modulo n. If n is prime, then every residue class $a \not\equiv 0 \pmod{n}$ is a Miller-Rabin liar. A residue class a modulo n is called a Miller-Rabin witness if $(a, n) = 1$ and a is not a Miller-Rabin liar, i.e.,

$$a^r \not\equiv 1 \pmod{n},$$

and the sequence

$$a^r, a^{2r}, a^{4r}, \ldots, a^{2^{k-1}r} \pmod{n}$$

does not contain the residue class -1 modulo n. If n is prime, then there is no Miller-Rabin witness modulo n. That is, if we find a Miller-Rabin witness, it is certain that n is a composite number.

Using primitive roots, it can be proved that **for composite n at least 75% of all reduced residue classes modulo n are Miller-Rabin witnesses. Thus, if we check for k pieces of a's whether the given a is a Miller-Rabin liar, the probability that each a is a Miller-Rabin liar is less than 4^{-k} in case of composite n.** However, before moving on to the proof, we note that Keith [4] has given a tricky, elementary proof of a slightly weaker claim than the one above.

So let us see the proof that if $n > 9$ is an odd composite number, then at least 75% of all reduced residue classes modulo n are Miller-Rabin witnesses.

This is based on the following theorem, the proof of which is given based on Crandall and Pomerence's book [2].

Theorem 9.9. *Let $n > 9$ an odd composite number. Write $n-1$ of the form $n - 1 = 2^k r$, where $k > 1$ is an integer and r is odd. Let*

$$B = \{a \in \mathbb{Z}_n^* : \ a^r = 1 \text{ or } a^{2^i r} = -1 \textit{ for some } 0 \leq i < k\}.$$

Then

$$\frac{|B|}{\varphi(n)} \leq \frac{1}{4}.$$

Proof of Theorem 9.9.. Denote by 2^ℓ the largest power of two such that 2^ℓ is a divisor of $p-1$ for every prime divisor of p in n. Then the set B contains the following set:

$$B' = \{a \in \mathbb{Z}_n^* a^{2^{\ell-1}r} = \pm 1\}.$$

Indeed, it is clear that if $a^r = 1$, then $a \in B'$. On the other hand, if $a^{2^i r} = -1$ for some $0 \le i < k$, then $a^{2^i} r \equiv -1 \pmod p$ holds for all prime divisor p of n. Hence, 2^{i+1} *exact divisor* of order a is modulo p, i.e., $2^{i+1} \mid o_p(a)$, but $2^{i+2} \nmid o_p(a)$. But then $2^{i+1} \mid p-1$ for every prime divisor p of n. Hence $\ell \ge i+1$. That is, $a^{2^{\ell-1}r} = (-1)^{2^{\ell-i-1}}$, which can be -1 or $+1$. Thus indeed $B' \subset B$.

According to the Chinese remainder theorem, the number of a for which $a^{2^{\ell-1}r} = 1$ is exactly

$$\prod_{p|n} g(p),$$

where $g(p)$ is the number of the solutions of the congruence $x^{2^{\ell-1}r} \equiv 1 \pmod{p^{\alpha_p}}$, where p^{α_p} is the largest power of p that divides n. Since $\mathbb{Z}_{p^{\alpha_p}}^*$ is a cyclic group (i.e., there exists a primitive root mod p^{α_p}) we have

$$g(p) = ((p-1)p^{\alpha_p}, 2^{\ell-1}r) = (p-1, r)2^{\ell-1}. \tag{9.13}$$

Indeed, if q is a fixed primitive root modulo p^{α_p}, then $x \equiv q^u \pmod{p^{\alpha_p}}$ is a solution of the congruence $x^{2^{\ell-1}r} \equiv 1 \pmod{p^{\alpha_p}}$ if and only if $p^{\alpha_p - 1}(p-1) \mid u2^{\ell-1}r$. We know that $(r, p) = 1$ (due to $r \mid n-1$ and $p \mid n$), so then $p^{\alpha_p-1} \mid u$ and $\dfrac{p-1}{(p-1, 2^{\ell-1}r)} \mid u$ is also satisfied. So then $\dfrac{p^{\alpha_p-1}(p-1)}{(p-1, 2^{\ell-1}r)} \mid u$. On the interval $0 \le u < p^{\alpha_p-1}(p-1)$ there are $(p-1, 2^{\ell-1}r)$ pieces of u of this kind, and with this we proved (9.13).

Thus

$$\left|\{a : \mathbb{Z}_n^* : \ a^{2^{\ell-1}r} = 1\}\right| = \prod_p (p-1, r)2^{\ell-1}. \tag{9.14}$$

Similarly,

$$\left|\{a : \mathbb{Z}_n^* : \ a^{2^{\ell}r} = 1\}\right| = \prod_p (p-1, r)2^{\ell},$$

which is exactly twice (9.14). Thus

$$\left|\{a : \mathbb{Z}_n^* :\ a^{2^{\ell-1}r} = -1\}\right| = \prod_p (p-1, r)2^{\ell-1}.$$

That is

$$|\mathcal{B}'| = 2\prod_p (p-1, r)2^{\ell-1}.$$

Then:

$$\frac{|B'|}{\varphi(n)} = 2\prod_{p|n} \frac{(p-1,r)2^{\ell-1}}{(p-1)p^{\alpha_p-1}}.$$

Suppose that, contrary to the statement of the theorem $\frac{|B|}{\varphi(n)} > \frac{1}{4}$. Since $B \subset B'$ then

$$\frac{1}{4} < 2\prod_{p|n} \frac{(p-1,r)2^{\ell-1}}{(p-1)p^{\alpha_p-1}}. \tag{9.15}$$

Note that $(p-1, r)2^{\ell-1} \mid \frac{p-1}{2}$, so the right-hand side of (9.15) is at most $2^{1-\omega(n)}$, where $\omega(n)$ denotes the number of different prime divisors of the number n. Hence $\omega(n) \leq 2$.

Assume that $\omega(n) = 2$. Then n has two different prime divisors, denoted by p and q. If the square of one of them is also a divisor of n, say $p^2 \mid n$, i.e., $\alpha_p \geq 2$, then the right-hand side of (9.15) is at least $\frac{2^{1-2}}{p} \leq \frac{2^{1-2}}{3} = \frac{1}{6}$, which is a contradiction. So $\alpha_p = \alpha_q = 1$, i.e., $n = pq$. Then (9.15) can be written in the following form:

$$\frac{p-1}{(p-1,r)2^\ell} \cdot \frac{q-1}{(q-1,r)2^\ell} < 2.$$

Since the factors on the left are integers, both are necessarily 1. That is

$$p-1 = (p-1,r)2^\ell,\ q-1 = (q-1,r)2^\ell.$$

That is, 2^ℓ is an exact divisor of both $p-1$ and $q-1$, and the largest odd divisors of $p-1$ and $q-1$ are necessarily divisors of r. Denote the largest odd divisor of $p-1$ by s and the largest odd divisor of $q-1$ by t. According to the former, $s \mid r$ and $t \mid r$. Since $p \equiv 1 \pmod{s}$ and

$$pq - 1 = n - 1 = 2^k r,$$

we have

$$q-1 \equiv pq-1 \equiv 2^k r \equiv 0 \pmod{s}.$$

That is, $s \mid q-1$, but the largest odd divisor of $q-1$ is t, so $s \mid t$. Similarly, $t \mid s$. That is, $t = s$. In other words, $p-1 = q-1$, and in this case we also reached a contradiction.

Finally, if $n = p^\alpha$, then by (9.15) we have

$$\frac{1}{4} < \frac{(p-1,r)2^\ell}{(p-1)p^\alpha} \le \frac{p-1}{(p-1)p^\alpha} = \frac{1}{p^{\alpha-1}},$$

from which $p^{\alpha-1} < 4$. This can only happen if $p = 3$ and $\alpha = 2$, contradicting the condition $n > 9$ of the theorem. In each case, we got a contradiction, and so we have proved the statement of the theorem.

Erich Bach [1] proved that if the generalized Riemann hypothesis is true, then the smallest Miller-Rabin witness is $\le 2(\log n)^2$.

The primality tests listed so far are very fast and are still among the most popular primality tests due to their simplicity. In theory, however, there are also faster primality tests, e.g., Grantham [3] gave a test that is asymptotically about three times faster than the Miller-Rabin test. However, this primality test is significantly more complicated than the previous ones, so we will not describe its steps. However, interested readers can look it up in [3].

References

[1] E. Bach, *Explicit bounds for primality testing and related problems*, Mathematics of Computation, 55 (191) (1990), 355–380.

[2] R. Crandall, C. Pomerance, *Prime Numbers; a Computational Perspective*, Springer Verlag, New York 2001.

[3] J. Grantham, *A probable prime test with high confidence*, J. Number Theory, 72 (1998), 32–47.

[4] C. Keith, *Miller-Rabin primality test*, Published in Encyclopedia of Cryptography 2011 Mathematics, https://kconrad.math.uconn.edu/blurbs/ugradnumthy/millerrabin.pdf.

[5] G. L. Miller, *Riemann's Hypothesis and Tests for Primality*, Journal of Computer and System Sciences, 13 (3) (1976), 300–317.

[6] M. O. Rabin, *Probabilistic algorithm for testing primality*, Journal of Number Theory, 12 (1) (1980), 128–138.

9.8 An example of the Miller-Rabin primality test

Let $n = 561$ $(= 3 \cdot 11 \cdot 17)$ (see chapter 9.2). This is the Carmichael number, so the Fermat primality test does not tell us that n is composite. However, let's test 561 with the Miller-Rabin test. Then

$$561 - 1 = 2^4 \cdot 35.$$

Let's choose a random a. Let's say $a = 7$. Consider the following sequence

$$7^{35}, 7^{70}, 7^{140}, 7^{280}, 7^{560} \pmod{561}.$$

First, we calculate 7^{35} by repeated squaring.

	7^1	7^2	7^4	7^8	7^{16}	7^{32}
mod 561	7	49	157	-35	103	-50

Now

$$7^{35} \equiv 7^{32} \cdot 7^2 \cdot 7 \equiv (-50) \cdot 49 \cdot 7 \equiv 241 \pmod{561}.$$

Then, by repeated squaring, we calculate the elements of the sequence $7^{35}, 7^{70}, 7^{140}, 7^{280}, 7^{560} \pmod{561}$:

	7^{35}	7^{70}	7^{140}	7^{280}	7^{560}
mod 561	241	-263	166	67	1

Since the sequence $7^{35}, 7^{70}, 7^{140}, 7^{280} \pmod{561}$ does not contain the residue -1 modulo 561, so 7 is a Miller-Rabin witness. Thus 561 is a composite number according to the test.

9.9 AKS primality test

In 2002, three Indian mathematicians, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena [1] created a deterministic polynomial algorithm for primality testing. This algorithm is too complicated to describe here in full detail, but we will say a few words about it. The algorithm is based on the following theorem:

Theorem 9.10. *Let $a \in \mathbb{Z}$, $n \in \mathbb{Z}$ and $(a, n) = 1$. Then n is prime if and only if*

$$(x + a)^n \equiv x^n + a \pmod{n}.$$

Proof of the Theorem 9.10.

First, we see the following: if n is prime and $1 \leq i \leq n - 1$, then $n \mid \binom{n}{i}$. Indeed $\binom{n}{i} = \frac{n!}{i!(n-i)!}$ is an integer, so $i!(n-i)! \mid n!$. But n is a prime number and thus it is not included in the prime factorization of $i!(n-i)!$, i.e., $i!(n-i)! \mid (n-1)!$ also holds. Thus $\frac{(n-1)!}{i!(n-i)!}$ is an integer, and hence $\binom{n}{i} = \frac{n!}{i!(n-i)!} = n \cdot \frac{(n-1)!}{i!(n-i)!}$ is divisible by n. Now we can continue by the proof of the theorem.

First let n be a prime number. Now the coefficient of x^i in $(x+a)^n - x^n - a$ is $\binom{n}{i}a^{n-i}$ if $1 \leq i \leq n-1$. This coefficient is divisible by n. Moreover, the constant term is $a^n - a$, which is divisible by n by Fermat's little theorem.

Next assume that n is composite. Let q be prime such that $q^k \mid n$ for some k but $q^{k+1} \nmid n$. Then $q^k \nmid \binom{n}{q} = \frac{n(n-1)(n-2)\cdots(n-q+1)}{q!}$ and since $(q^k, a) = 1$ also holds, the coefficient of x^q is not divisible by n in the polynomial $(x + a)^n - x^n - a$. This completes the proof of the theorem.

The starting point of the AKS test is the above theorem, however, the algorithm itself is slightly complicated, and its proof requires serious number theory tools. We now only describe the steps of the algorithm without proof:

(1) If $n = a^b$ for some $a \in \mathbb{N}$ and $b > 1$, then n is composite.

(2) We look for the smallest r for which $o_r(n) > \log_2^2 n$, where $o_r(n)$ denotes the order of n modulo r. (This r is always less then or equal to $\lceil (\log_2 n)^5 \rceil$.)

(3) For every $2 \leq a \leq \min(r, n-1)$, we check whether a is a divisor of n. If so, then n is composite.

(4) If $n \leq r$, then n is prime. (This step can be omitted if $n \geq 5.7 \cdot 10^6$.)

(5) If $0 \leq a \leq \sqrt{\varphi(r) \log_2 n}$ exists, such that

$$(x+a)^n \not\equiv x^n + a \pmod{x^r - 1, n},$$

then n is composite, otherwise prime.

In the last step, $\pmod{x^r - 1, n}$ is connected to an equivalence relation that is understood over $\mathbb{Z}_n$ and the polynomial x^r is equivalent to the polynomial constant 1 (that is, both n and $x^r - 1$ is also equivalent to 0).

The running time of the algorithm is $O\Big((\log n)^{12+\varepsilon}\Big)$. In 2005, Lenstra and Pomerance [4] developed such a variant of AKS primality test, which has running time $O((\log n)^6 (\log\log n)^c)$. An updated version is also available in [5].

It is important to highlight that many Hungarian records have been set in the last 30 years in the research of the largest primes. For example, T. Csajbók, G. Farkas, A. Járai, Z. Járai and J. Kasza [2] from the Eötvös Loránd University IK held the world record in 2006, in the largest twin prime research. Additional records are also available on the following website: http://compalg.inf.elte.hu/~ajarai/worldr.htm. Those who are interested can read an interesting article about the so-called Járai method, for example, in [3]. These methods do not use AKS, but deterministic primality tests developed for specially shaped primes (such as Mersenne primes or generalized Fermat primes).

References

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Annals of Mathematics. 160 (2) (2004), 781–793.

[2] T. Csajbók, G. Farkas, A. Járai, Z. Járai, J. Kasza, *Report on the largest known Sophie Germain and twin primes*, Ann. Univ. Sci. Budap. Rolando Eötvös, Sect. Comput. 26 (2006), 181-183.

[3] G. Farkas, G. Gévay, P. Magyar, B. Szekeres *Járai's prime hunting methods reloaded (the largest known Cunningham chain of length 2 of the 2nd kind)*, Ann. Univ. Sci. Budap. Rolando Eötvös, Sect. Comput. 51 (2020), 69-75.

[4] H. W. Lenstra Jr., C. Pomerance, *Primality testing with Gaussian periods*, preliminary version July 20, 2005.

[5] H. W. Lenstra Jr., C. Pomerance, *Primality testing with Gaussian periods*, https://math.dartmouth.edu/~carlp/aks111216.pdf.

10 RSA

Ron Rivest, Adi Shamir, and Len Adlema [1] created one of the best-known public-key cryptosystem called RSA, in the mid-1970s. (The name RSA comes from the initials of the authors' names...) For a long time, RSA plays an important role in various IT, computer, and communication systems. However, many people today (for certain applications) do not consider it safe enough. The reason for this is that Peter Shor proved in [2] that factorization and the discrete logarithm problem can be solved with a quantum algorithm in polynomial time. In the last 30 years, quantum-resistant cryptography has developed a lot, but in terms of efficiency, they cannot even come close to methods based on RSA, discrete logarithm, or elliptic curve discrete logarithm.

References

[1] R. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM. 21 (2) (1978), 120–126.

[2] P. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings 35th Annual Symposium on Foundations of Computer Science (1994), 124 - 134.

10.1 RSA encrypting algorithm

Let $N = pq$ be the product of two large primes, where in the binary number system each prime consists of n digits. This N is called the RSA modulus. Today, the typical length of N is 4096 bits, which means 1234 decimal places. At first, the $N = 128$ bit modulus also proved to be safe, then as a result of the development of attacks and technology, it continuously increased: 256, 512, 1024 and then 2048 bits.

Let e, d be two integers, where

$$ed \equiv 1 \pmod{\varphi(N)}.$$

Here, by $N = pq$ we have $\varphi(N) = (p-1)(q-1)$. The exponent e is called the public exponent, while d is called the private exponent.

The pair (N, e) is the public key, while the pair (N, d) is the private or (in other words) secret key. The latter is known only to the person to whom we send the encrypted message and who will decode our messages.

A message can be thought of as an integer such that $0 < M < N$. (For the sake of simplicity, we now assume that $(M, N) = 1$ is satisfied for the message. This can be easily achieved with a possible small modification of the message. However, in fact, the condition $(M, N) = 1$ is not needed, only in the general case the proof of the decoding procedure is a few lines longer than the one below.)

The encrypted message is

$$C \equiv M^e \pmod{N} \qquad \leftarrow \text{RSA function.}$$

During decoding we know

$$ed \equiv 1 \pmod{\varphi(N)},$$

thus $\exists k \in \mathbb{Z}^+$ such that $ed = k\varphi(N) + 1$. By Euler-Fermat theorem

$$M \equiv C^d \pmod{N}, \qquad \leftarrow \text{this is the decoding,}$$

since

$$C^d \equiv (M^e)^d = M^{k\varphi(N)+1} = M\left(M^{\varphi(N)}\right)^k \equiv M \cdot 1^k \equiv M \pmod{N}.$$

RSA is a one-way function, since the encryption can be easily done with the knowledge of the public key (N, e) by modular exponentiation, the time required of which is $O(\log e(\log N)^2)$ bit operations, however, the inversion without knowing the private key d is very difficult. Most of the RSA attacks focus on how to invert the RSA function without knowing d. More precisely, if only the triple (N, e, C) is given (and p, q, d are secret), can we recover the original message M from C?

Remark: d can be calculated from p, q, e, since d is the solution of the following linear congruence:

$$ex \equiv 1 \pmod{(p-1)(q-1)}.$$

But here p, q are secret, only $N = pq$ is given, to know p, q we need to factorize N, but this is very slow for large numbers N.

Is there another inversion that avoids the factorization of N?

References

[1] R. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM. 21 (2) (1978), 120–126.

10.2 RSA attacks

Based on the informative paper by Tamás Dénes [3], below, we describe some known RSA attacks that we should definitely pay attention to when using RSA. For those interested in the topic I also recommend reading the paper [4].

Implementation dependent attacks

By examining the computer's server, e.g. delimiting the area of the server in which the keys are stored, even in coded form.

Time measurement of current consumption.

These attacks require not only theoretical considerations, but also serious technique.

Timing attack

Known, see e.g. Lovász-Gács book [5] that the complexity of squaring a n digit number is less than multiplying two n digit numbers. This is not only a theoretical result, but also practically so. Modular exponentiation includes exactly such operations. Implementations have been made that take advantage of this difference and achieve a significant speedup.

In response, the electrical engineers drew attention to the fact that if someone could observe the processor, due to the time difference, he could determine when he was squaring and when he was multiplying, from which the binary form of the exponent of the secret decoder could easily be written. After that, the above acceleration option was no longer used.

Attacks based on incorrect applications of RSA

Often, only p is chosen as a random prime, and q is chosen as the smallest prime following p. Then

$$N = p(p+2) \qquad \text{e.g.,}$$

from which p can be calculated very quickly. If p, q are consecutive primes and $q - p = d$ is relatively small,

$$\text{and from } N = p(p+d) \ p \text{ is easily calculated} \ldots$$

There are several similar cases that are not excluded by the official RSA algorithm, but in which cases the RSA can be broken.

Careful use of RSA is important!

Common moduli

It is important that a fixed N module is used by only one person, because people with the same modulus can decipher each other's messages based on the following theorem of Simmons. (The full description of the theorem below was first published in [2] in the publication J. DeLaurentis, who mentioned at the beginning of the paper that the theorem originated from Simmons.)

Theorem 10.1. (Simmons) *Let (N, e) be an RSA public key. If the private exponent d is given, the factorization of N can be done efficiently. The reverse is also true: If the factorization of N is known, then for each public exponent e, the private exponent d can be efficiently calculated.*

Some computer systems (communities) do not care about this theorem and give all users the same N modulus, although they give each a different private and public exponent... There are often economic reasons behind this, since generating large p and q primes costs a lot, and if each user needs a different pair of primes, it certainly multiplies the prices... However, this is certainly not worth saving on it if IT security is important! (We note here that if we do not stipulate that p and q are **definitely** prime, but only with a very, very high probability, it significantly speeds up the generation time of p and q. For example, in practice, a number that satisfies the Miller-Rabin primality test for 511 integers can be considered a prime number.) The probability that such a number is composite is less than $4^{-511} = 2^{-1022}$, which is a negligible probability in practice. Then number of 1024 bit odd numbers number is 2^{1023}, so it is expected that there are at most two "cheating" numbers. By the way, it does not take long to run a 511 Miller-Rabin primality test, but in practice a 100 test is more than enough. Deterministic primality tests are much slower than the above primality tests.)

Proof of Theorem 10.1.

p, q, e given $\Rightarrow d$ is efficiently calculated

Then d is the solution of the following congruence:

$$\begin{aligned} ex &\equiv 1 \pmod{\varphi(N)} \\ ex &\equiv 1 \pmod{(p-1)(q-1)}. \end{aligned}$$

<u>N, e, d given $\Rightarrow p, q$ is efficiently calculated</u>

Let $k \overset{\text{def}}{=} ed - 1$.

$$\begin{aligned} ed &\equiv 1 \pmod{\varphi(N)} \\ \varphi(N) &\mid ed - 1 \\ \varphi(N) &\mid k. \end{aligned}$$

Now $\varphi(N)$ is even, since $\varphi(N) = (p-1)(q-1)$. That is, k is also even: $k = 2^t r$, where r is odd. Then

$$\forall\ (g, N) = 1 \text{ we have} \qquad g^k \equiv 1 \pmod{N}$$

since $\varphi(N) \mid k$. Thus $g^{k/2}$ second root of unity modulo N:

$$\left(g^{k/2}\right)^2 = g^k \equiv 1 \pmod{N}.$$

By the Chinese remainder theorem, there are 4 roots of unity modulo pq (now $N = pq$). Two of them are ± 1. The other two roots of unity are $\pm x$, where

$$x \equiv 1 \pmod{p} \qquad x \equiv -1 \pmod{q}.$$

Then by $p \mid x - 1$ and $p \mid N$ we get

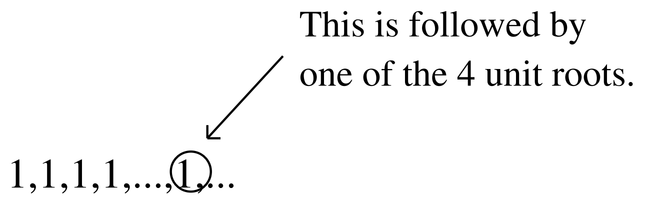
$$p \mid (x-1, N) \Rightarrow p = (x-1, N). \tag{10.1}$$

(Here we used the fact that N has only 4 divisors: $1, p, q, pq$ so $(x-1, N)$ can be one of the above four values that is divisible by p, that is, p and pq. But $x - 1$ is no longer divisible by q, so indeed $p = (x-1, N)$.)

Consider the following sequence:

$$g^k, g^{k/2}, g^{k/4}, \ldots, g^{k/2^t} \qquad \pmod N.$$

It starts with some 1's:



In other words, there is at least a 1/2 chance that the beginning of the sequence has the following form:

$$1, 1, 1, 1, \ldots, 1, \pm x, \ldots.$$

In this case $g^{k/2^s} \equiv \pm x \pmod N$ for some s, so x can be quickly determined by modular exponentiation applied to g, and if we calculate the greatest common divisor of $x - 1$ and N by the Euclidean algorithm, we get the prime divisor p (see (10.1)).

<u>Small private exponent</u>

In practical applications of RSA, a small d may be chosen instead of a randomly chosen private exponent d to reduce key generation time or to make decryption faster. However, a theorem proved by M. Wiener [6] shows that if d is a given becomes smaller than a certain bound, then the RSA will be decipherable...

Let the prime-factor decomposition of N be pq, where now $q < p$. Since p and q have the same number of digits in the binary number system, we also know that $p < 2q$. Wiener proved the following:

Theorem 10.2. (Wiener) *Let $N = pq$, where $q < p < 2q$ and let*

$$d < \frac{1}{3}N^{1/4}. \tag{10.2}$$

If (N, e) is given (i.e., the RSA modulus and the public exponent), but we do not know the value of the private exponent d, but we know that (10.2) is satisfied, then the private exponent d can be efficiently calculated from the values of N and e (provided that d is a solution of the congruence $ed \equiv 1 \pmod{\varphi(N)}$).

Proof of Theorem 10.2. We know that

$$ed \equiv 1 \pmod{\varphi(N)}.$$

Let

$$ed = t\varphi(N) + 1 \qquad \Rightarrow ed - t\varphi(N) = 1.$$

Then:

$$\left|\frac{e}{\varphi(N)} - \frac{t}{d}\right| = \frac{1}{d\varphi(N)}.$$

Since $\frac{e}{\varphi(N)}$ is close to $\frac{e}{N}$, $\frac{t}{d}$ is an approximation of $\frac{e}{N}$, and therefore (as a consequence of theorem 3.24.) d can be obtained from the continued fraction form. Precisely:

$$\begin{aligned}
N &= pq \quad q < p < 2q \\
q^2 &< pq = N \quad \Rightarrow q < \sqrt{N} \\
\frac{p^2}{2} &< pq = N \quad \Rightarrow p < \sqrt{2}\sqrt{N} \\
p + q &< 3\sqrt{N} \\
N - \varphi(N) &= pq - (p-1)(q-1) = p + q - 1 < 3\sqrt{N} \\
|N - \varphi(N)| &< 3\sqrt{N}.
\end{aligned}$$

Thus

$$\begin{aligned}
\left|\frac{e}{N} - \frac{t}{d}\right| &= \left|\frac{ed - t\varphi(N) - tN + t\varphi(N)}{Nd}\right| \\
&= \left|\frac{1 - t\left(N - \varphi(N)\right)}{Nd}\right| \le \frac{3t\sqrt{N}}{Nd} = \frac{3t}{d\sqrt{N}}.
\end{aligned}$$

Here

$$t\varphi(N) = ed - 1 < ed.$$

By $e < \varphi(N)$ we have

$$\begin{aligned} t\varphi(N) &< \varphi(N)d \\ t < d &< \frac{1}{3}N^{1/4}. \end{aligned}$$

That is

$$\left|\frac{e}{N} - \frac{t}{d}\right| \leq \frac{3t}{d\sqrt{N}} < \frac{N^{1/4}}{d\sqrt{N}} = \frac{1}{dN^{1/4}} < \frac{1}{2d^2}.$$

Based on the theory of continued fractions, see Theorem 3.24., all such fractions $\frac{t}{d}$ can be obtained from the continued fraction form $\frac{e}{N}$. Since

$$ed - t\varphi(N) = 1 \qquad \Rightarrow \qquad (t, d) = 1.$$

That is, if the value of $r = \frac{t}{d}$ is given, then since we know that $(t, d) = 1$, we immediately get both t and d.

Attacks based on the factorization of the modulus

Theorem 10.3. *If a number of the form $N = pq$ is given, so N is the product of two different primes, then obtaining the values of p and q is essentially polynomially equivalent to the computation of $\varphi(N)$. More precisely:*

Knowing N, p, q we get $T(\varphi(N)) = O(\log N)$

Knowing $N, \varphi(N)$ we get $T(p, q) = O((\log N)^3)$.

Proof of Theorem 10.3.

a) The value of $\varphi(N) = (p-1)(q-1) = pq - p - q + 1 = N - p - q + 1$ can be calculated with 2 subtractions and 1 addition, so the time required is $O(\log N)$.

b) Since $\varphi(N) = N - p - q + 1$, thus

$$\begin{aligned} p + q &= N - \varphi(N) + 1 \\ pq &= N \end{aligned}$$

From the relation between the roots and coefficients, we get that p and q are the roots of following second degree polynomial:

$$x^2 - (N - \varphi(N) + 1)x + N = 0.$$

as a quadratic equation. By the quadratic formulas

$$p, q = \frac{N - \varphi(N) + 1 \pm \sqrt{(N - \varphi(N) + 1)^2 - 4N}}{2},$$

which requires $O\left((\log N)^3\right)$ bit operations.

The most well-known group of theoretical RSA attacks is the one in which the attack is based on the factorization of the modulus, since $\varphi(N)$ can be easily calculated from the prime factorization of N, and from this $d \equiv e^{-1} \pmod{\varphi(N)}$ can be calculated easily.

Factorization algorithms are slow, they will be discussed in more detail in the next chapter.

References

[1] D. Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices of the American Mathematical Society. 46 (2) (1999), 203–213, http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html.

[2] J. DeLaurentis, *A further weakness in the common modulus protocol for the RSA cryptoalgorithm*, Cryptologia 8 (1984), 253–259.

[3] T. Dénes, *Új eredmények az RSA kulcsok megfejtéséhez*, Híradástechnika, 2002/1. 47-55, http://www.titoktan.hu/_raktar/_e_vilagi_gondolatok/HTRSA.htm.

[4] T. Dénes, *A Public Key System és az RSA biztonsági kérdései*, Híradástechnika, 2002/1. 47-55 (in Hungarian), http://www.titoktan.hu/_raktar/_e_vilagi_gondolatok/PKS-RSA.htm.

[5] P. Gács, L. Lovász, *Algoritmusok*, Tankönyvkiadó 1989.

[6] M. J. Wiener, *Cryptanalysis of short RSA secret exponents*, IEEE Transactions on Information Theory. 36 (3) (1990), 553–558.

11 Factorization

All primality tests discussed so far (except for the "trial division"), which very slow), just shows that n is composite or not, but usually does not give a factor. Thus, numbers of the form $n = pq$, where $q > p > q/2$ as used in RSA [13], cannot also be factorized.

In almost all cases, Miller-Rabin [8] primality testing, which determines whether a given integer is prime, is a polynomial time algorithm. Although the AKS primality test [1] takes slightly longer, it is always a polynomial time algorithm. To the best of our knowledge, factorization cannot be a polynomial-time algorithm (perhaps it is not).

Many factorization methods are known, and I will present some of them in this lecture notes, using the book of Koeblitz [4].

These are: Fermat's factorization method [3], [7], Dixon's random square method [2], continued fraction method [9] and Lenstra's elliptic curve algorithm [5] (which will be discussed later in section 13.5). Apart from those mentioned above, various more factorization algorithms are known, like: Pollard ρ method [10], quadratic sieve [11], number field sieve [6].

To the best of our knowledge, the quadratic sieve is the fastest up to 100 decimal digits, and the number field sieve is the fastest above 100 decimal digits.

References

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Annals of Mathematics. 160 (2) (2004), 781–793.

[2] J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. 36 (153) (1981), 255–260.

[3] P. Fermat, *Oeuvres de Fermat*, vol. 2, 1894, o. 256.

[4] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[5] H. W. Lenstra Jr., *Factoring integers with elliptic curves*, Annals of Mathematics. 126 (3) (1987), 649–673.

[6] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The number field sieve*, kiterjesztett absztrakt: Proc. 22nd Annual ACM Sympos. Theory of Computing (STOC) (Baltimore, May 14-16, 1990), 564-572.

[7] J. McKee, *Speeding Fermat's factoring method*, Mathematics of Computation (68) (1999), 1729–1737.

[8] G. L. Miller, *Riemann's hypothesis and tests for primality*, Journal of Computer and System Sciences, 13 (3) (1976), 300–317.

[9] M. A. Morrison, J. Brillhart, *A method of factoring and the factorization of F7*. Mathematics of Computation. American Mathematical Society. 29 (129) (1975), 183–205.

[10] J. M. Pollard, *A Monte Carlo method for factorization*. BIT Numerical Mathematics. 15 (3) (1975), 331–334.

[11] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in Computational Methods in Number Theory, Part I, H.W. Lenstra, Jr. and R. Tijdeman, eds., Math. Centre Tract 154, Amsterdam, 1982, pp 89-139.

[12] M. O. Rabin, *Probabilistic algorithm for testing primality*, Journal of Number Theory, 12 (1) (1980), 128–138.

[13] R. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM. 21 (2) (1978), 120–126.

11.1 Fermat's factorization method

Look at the example where we need to factor an odd integer n that is the product of two nearly equal large numbers.

$$ab = n, \quad a > b, \quad a - b \text{ is small compared with } a.$$

Instead of factorizing n (i.e., searching for $a, b \in \mathbb{Z}$ with $n = ab$), we attempt to find $x, y \in \mathbb{Z}$ for which

$$n = x^2 - y^2.$$

In the following, we will assume that n is an odd number. These two problems are equivalent due to the following arguments. Let $n = ab$ first. Define x, y by

$$\begin{aligned} x + y &= a, \\ x - y &= b. \end{aligned} \tag{11.1}$$

Then:

$$x = \frac{a+b}{2}, \quad y = \frac{a-b}{2}$$
$$x^2 - y^2 = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2 = \frac{2ab}{4} + \frac{2ab}{4} = ab = n.$$

Conversely, if $x^2 - y^2 = n$, then define a and b by (11.1). Then

$$x^2 - y^2 = (\underbrace{x+y}_{a})(\underbrace{x-y}_{b}) = ab = n.$$

Following these:

Definition 11.1. (Fermat's factorization method [1]) . *Suppose that n is not a square. Let's take the first square number greater than n, which is t^2, where $t = [\sqrt{n}] + 1$. First, let $x = t = [\sqrt{n}] + 1$. If*

$$\begin{aligned} x^2 - n = t^2 - n = \ &\textit{square} = y^2, \\ x^2 - y^2 = n. \end{aligned}$$

If $t^2 - n$ is not a square, we look at the following square, $x = t + 1$:

$$\begin{aligned} x^2 - n = (t+1)^2 - n = \ \textit{square}\ = y^2, \\ x^2 - y^2 = n. \end{aligned}$$

If $(t+1)^2 - n$ is not a square, we look at $x = t + 2$-t, and so on.

References

[1] P. Fermat, *Oeuvres de Fermat*, vol. 2, 1894, p. 256.

[2] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

11.2 Factor base algorithm

Fermat factorization [2], can be further developed into a more efficient algorithm using a tricky idea.

Suppose that we would like to factorize the odd number n. During Fermat factorization, we saw that it is sufficient to find such integers x, y for which $x^2 - y^2 = n$. Instead, start looking for integers x, y for which:

$$\begin{aligned} x^2 - y^2 \equiv 0 \pmod{n}, \\ y \not\equiv \pm x \pmod{n}. \end{aligned} \tag{11.2}$$

For this:

$$\begin{aligned} &n \mid x^2 - y^2 = (x-y)(x+y), \\ &n \nmid x - y, \quad n \nmid x + y, \\ &1 < (n, x-y), (n, x+y) < n. \end{aligned}$$

So we can use the Euclidean algorithm to calculate $(n, x+y)$ and get a real divisor of n, which is a factor. The procedure can be repeated for the factors of n and so on until the prime factorization of n is obtained.

Next we consider the factor base algorithm [4] or in another name Dixon's random square algorithm [1], in which we would like to find the pair (x, y) that satisfies (11.2). The factor base algorithm was established by Lehmer and Powers [4].

Definition 11.2. *The set* $\{p_1, p_2, \dots, p_k\}$, *where* $p_1 < p_2 < \dots < p_k$ *is called* factor base, *if* $p_1 = -1$ *and* $p_2, \dots, p_k$ *are different primes. Furthermore, if we fix the integer* $n \in \mathbb{N}$ *to be factored, we say that* a *is a* B*-number if the least absolute residue of* a^2 *modulo* n *can be expressed as a product of integers from* B.

Example. If $n = 4633$ and $B = \{-1, 2, 3\}$, then 67, 68, 69 are B-numbers, since

$$\begin{aligned}
&67^2 \equiv -144 \equiv (-1) \cdot 2^4 \cdot 3^2 \qquad (4633),\\
&68^2 \equiv -9 \equiv (-1) \cdot 3^2 \qquad (4633),\\
&69^2 \equiv 128 \equiv 2^7 \qquad (4633).
\end{aligned}$$

Definition 11.3. (Dixon's random square algorithm [1]) *Let* n, $B = \{b_1, b_2, \dots, b_h\}$ *be given. Then we need to find* $h+1$ *pieces of* B*-numbers, say we will denote them by* $a_1, \dots, a_{h+1}$. *Then*

$$a_i{}^2 \equiv b_1^{\alpha_{i_1}} \dots b_h^{\alpha_{i_h}} \pmod n \quad \textit{for } i = 1, \dots, h+1.$$

Let $\alpha_{i,j} \mod 2$ *is the least nonnegative residue of* $\beta_{i,j}$ *modulo* 2, *that is* $\alpha_{i,j} \equiv \beta_{i,j} \pmod 2$ *and* $0 \leq \beta_{i,j} \leq 1$. *Consider the* $h+1$ *vectors*

$$\underline{u_i} = (\beta_{i_1}, \dots, \beta_{i_h}) \quad (i = 1, \dots, h+1)$$

over $\mathbb{F}_2$. *These vectors are from a vectorspace of* h *dimensions over* $\mathbb{F}_2$. *Using linear algebra we know that these vectors are linearly dependent, that is* $\exists\ \varepsilon_1, \dots, \varepsilon_{n+1} \in \mathbb{F}_2$ *such that*

$$\varepsilon_1 \underline{u_1} + \dots + \varepsilon_{h+1} \underline{u}_{h+1} = \underline{0} \qquad \textit{over } \mathbb{F}_2. \tag{11.3}$$

Then:
$$\prod_{i=1}^{h+1} a_i^{2\varepsilon_i} = \prod_{i=1}^{h+1} b_1^{\varepsilon_i\alpha_{i,1}} \dots b_h^{\varepsilon_i\alpha_{i,h}} \equiv \prod_{j=1}^{h} b_j^{\sum_{i=1}^{h+1}\varepsilon_i\alpha_{i,j}}.$$

Here $\sum\limits_{i=1}^{h+1} \sum_{i=1}^{h+1} \varepsilon_i\alpha_{i,y}$ *is even because of* (11.3)*, so let* $\sum_{i=1}^{h+1}\varepsilon_i\alpha_{i,j} \overset{def}{=} 2k_j$*. Then:*

$$\underbrace{\left(\prod_{i=1}^{h+1} a_i{}^{\varepsilon_i}\right)^2}_{x} \equiv \underbrace{\left(\prod_{j=1}^{h} b_j{}^{k_j}\right)^2}_{y} \pmod{n}.$$

If $x \not\equiv \pm y \ (n)$*, we are done, we have found a pair* x, y *which satisfies* (11.2)*.*

If $x \equiv -y$ *or* $x \equiv +y \pmod n$*, we need to find new* $h+1$ *pieces of* B*-numbers, say,* $a_1{}', \dots, a_{h+1}{}'$ *and so on.*

In the general Dixon's random square algorithm [1], the candidates for B-numbers are chosen randomly. If we find a B-number, we are pleased; if the integer does not turn out to be a B-number, we choose another candidate at random.

Example. In the previous example

$$67^2 \to (1,4,2) \to (1,0,0) = \underline{u_1},$$
$$68^2 \to (1,0,2) \to (1,0,0) = \underline{u_2},$$
$$69^2 \to (0,7,0) \to (0,1,0) = \underline{u_3}.$$

One additional vector would be required for a solid linear dependence, but in this case, the first two vectors are already linearly dependent since

$$1\underline{u_1} + 1\underline{u_2} + 0 \cdot \underline{u_3} = (2,0,0) \to (0,0,0).$$

Based on this:

$$67^2 \cdot 68^2 \equiv (-1)^1 2^4 3^2 \cdot (-1)^1 3^2 = (-1)^2 2^4 3^2 \equiv (2^2 \cdot 3^2)^2 \equiv 36^2 \pmod{4633}.$$

Here $67 \cdot 68 \equiv -77 \pmod{4633}$, thus

$$77^2 \equiv 36^2 \pmod{4633}.$$

That is

$$4633 \mid \underbrace{(77-36)}_{41}\underbrace{(77+36)}_{113}.$$

Computing

$$(41, 4633) = 41$$

we get a real divisor of 4633, and, indeed, the prime factorization of 4633 is:

$$4633 = 41 \cdot 113.$$

The factor base in Dixon's random square algorithm [1] is typically

$$B = \{-1\} \cup \{p : \ p \text{ prím }, \ p \le y\},$$

where y is a function of n such that the running time of the algorithm tends to be optimal. For optimally chosen y, the running time of this algorithm is $\exp(c\sqrt{\log n \log\log n})$. An abbreviated description of this estimate can be found e.g., in Koeblitz's book [3].

References

[1] J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. 36 (153) (1981), 255–260.

[2] P. Fermat, *Oeuvres de Fermat*, vol. 2, 1894, p. 256.

[3] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[4] D. H. Lehmer, R. E. Powers, *On factoring large numbers*, Bull. Amer. Math. Soc. 37 (10) (1991), 770-776.

11.3 Continued fraction method

We would like to factor n using Dixon's random square algorithm [1], and in order to do so we use a

$$B = \{-1\} \cup \{p : \; p \leq y\}$$

factor base and look for B-numbers.

To be more exact, we took a number $a \in \mathbb{Z}$, computed the least absolute residue of $a^2 \pmod{n}$ (which is between $-\frac{n}{2}$ and $\frac{n}{2}$) and tested it to see whether it is a B-number. This method was extended further by Morrison and Brillhart [3] using continued fractions.

The continued fracton method [3]: In this algorithm a is chosen so that in the continued fraction form of $\sqrt{n}$, we take the convergents $\frac{p_i}{q_i}$ and fix an $a = p_i$. If $r_n(p_i^2)$ denotes the least absolute residue of $p_i^2 \pmod{n}$, that is

$$r_n(p_i^2) \equiv p_i^2 \pmod{n}, \text{ where } -\frac{n}{2} \leq r_n(p_i^2) \leq \frac{n}{2},$$

then we know that

$$\left| r_n({p_i}^2) \right| \leq 2\sqrt{n}. \tag{11.4}$$

That is, $r_n(p_i^2)$ is more likely to be B-number than a^2 for a randomly chosen $a \in \mathbb{Z}_n$, since it is significantly (radically) smaller than the expected value around cn. To see (11.4), simply apply Theorem 3.23. for $x = \sqrt{n}$:

$$\left| {p_i}^2 - n{q_i}^2 \right| < 2\sqrt{n},$$

that is

$$\left| r_n({p_i}^2) \right| < 2\sqrt{n}.$$

Since for p_i's we have

$$p_i = a_i p_{i-1} + p_{i-2},$$

we also know the same recursion modulo n:

$$p_i \equiv a_i p_{i-1} + p_{i-2} \pmod{n}.$$

Thus in order to compute p_i'sit is enough to determine the continued fraction digits a_i's of $\sqrt{n}$. Since $\sqrt{n}$ is the root of a quadratic equation, thus by Theorem 3.20., the sequence of digits $a_0, a_1, a_2, \ldots$ will sooner or later become periodic.

The authors also suggested that, if necessary, we repeat the procedure with $\sqrt{kn}$, where k is a small natural number. It can even be $k = 2$. For example, in Morrison and Billhart's paper [3], during the factorization of the 7th Fermat number, $2^{2^7} + 1$, $k = 257$ was a good choice.

The running time of the algorithm is $\exp(c\sqrt{\log n \log\log n})$, but with a better constant c than in the case of general Dixon's random square algorithm.

References

[1] J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. 36 (153) (1981), 255–260.

[2] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[3] M. A. Morrison, J. Brillhart, *A method of factoring and the factorization of F7.* Mathematics of Computation. American Mathematical Society. 29 (129) (1975), 183–205.

11.4 Quadratic sieve method

The quadratic sieve method [2] is a tricky variant of Dixon's random square algorithm [1], developed by Pomerance. Then, up to a certain limit, we add primes p to the factor base, for which p's the n is the quadratic residue modulo p, more precisely:

$$B = \{-1, 2\} \cup \{p : \ p \text{ prím } \ p \leq y, \ \left(\frac{n}{p}\right) = 1\}.$$

Then we look for B-numbers in a set

$$S = \{t^2 - n : \ [\sqrt{n}] + 1 \leq t \leq [\sqrt{n}] + A\},$$

where A (as a function of n) is appropriate chosen constant. The running time is still $\exp(c\sqrt{\log n \log\log n})$, but with a smaller constant c than the ones used so far.

References

[1] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[2] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in: Computational Methods in Number Theory, Part I, H.W. Lenstra, Jr., R. Tijdeman, eds., Math. Centre Tract 154, Amsterdam, 1982, pp 89-139.

11.5 Number field sieve

In number theory, the general number field sieve [1] is a classic algorithm that is to be known as the fastest in case of integers to factorize greater than 10^{100}

In this algorithm, knowing the number n to be factorized, using a cleverly fixed irreducible polynomial f, we look for B-numbers between the elements of a certain set S based on the irreducible polynomial f.

The detailed description of the algorithm can be found in [1]. The running time of the algorithm is $exp(c(\log n)^{1/3}(\log\log n)^{2/3})$.

References

[1] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The number field sieve*, extended abstract: Proc. 22nd Annual ACM Sympos. Theory of Computing (STOC) (Baltimore, May 14-16, 1990), 564-572.

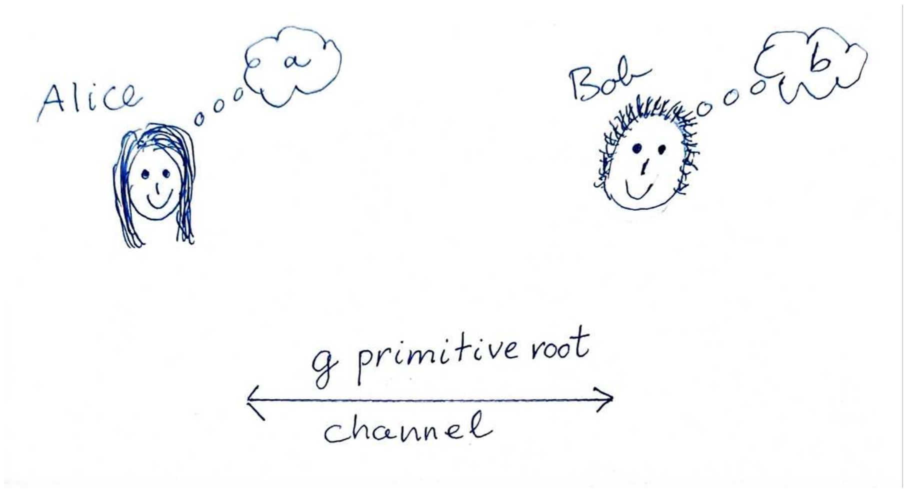
12 Diffie–Hellman key-exchange

The Diffie–Hellman key exchange system [1], [2] is one of the most important chapters of public key cryptography, in which the partners agree on a common secret key that others will not be able to figure out even if all of their communications are public. Since this algorithm was first published in [1], it is named after the authors of that paper, who were inspired by Merkle's idea. (Indeed, in 2002, Hellman suggested that the algorithm be renamed Diffie - Hellman - Merkle key exchange, which is still in use today, see e.g., [3].)

Assume that the two parties, Alice and Bob, are geographically separated (living in different countries), want to agree on a common secret key, and are worried that the channel through which they communicate (telephone or email) will be monitored. How can they come to an agreement on a common secret key?

Consider the simplest case, where the common secret key is an element of $\mathbb{Z}_p^*$. Then Alice chooses a secret integer $1 \leq a \leq p-1$, and Bob chooses a secret integer $1 \leq b \leq p-1$, but they never share the values of a and b to anyone. They kept them secret. They have agreed on a common primitive root g, which may have been made public.

It is not absolutely necessary that g to be a primitive root (though it is preferable), but it is enough that the order of g is very large.



Alice computes $g^a \pmod p$ Bob computes $g^b \pmod p$

$$\xleftrightarrow{g^a,\, g^b}$$

they send to each other

$$\text{common private key: } g^{ab} \pmod p$$

Alice can easily compute $g^{ab} \pmod p$ since Bob sent her $g^b \pmod p$ and she created $a \in \mathbb{Z}$, so $g^{ab} \pmod p$ can be calculated quickly using a simple modular exponentiation:

$$\text{Alice: } g^{ab} \equiv (g^b)^a \pmod p.$$

Bob does the same procedure: $g^{ab} \equiv (g^a)^b \pmod p$, thus they can both compute $g^{ab} \pmod p$.

Suppose Eve eavesdrops on the channel. She does not know a and b because Alice and Bob kept them in mind, but she may be able to obtain the values of

$$g, g^a, g^b \pmod p$$

Eve now must solve a Diffie-Hellman problem to calculate $g^{ab} \pmod p$. This is what it is:

Diffie–Hellman-problem. If you know the prime p, the primitive root g, the powers g^a and $g^b \pmod p$, how can you quickly determine $g^{ab} \pmod p$? Abbreviation: DHP.

Conjecture. There is no fast algorithm for solving the Diffie-Hellman problem.

A related problem is the discrete logarithm problem, which is discussed in the next sections.

The Diffie-Hellman key exchange algorithm can be easily generalized from $\mathbb{Z}_p^*$ to cyclic groups of order n, where the role of the primitive root is replaced by a generator element of the cyclic group.

References

[1] W. Diffie, M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory. 22 (6) (1976).

[2] R. C. Merkle, *Secure communications over insecure channels*, Communications of the ACM. 21 (4) (1978), 294–299.

[3] Wikipedia, *Diffie–Hellman key exchange*, https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

12.1 Discrete logarithm problem

The problem studied in the following few sections is:

Discrete logarithm problem. For a given prime p, primitive root g and $c \in \mathbb{Z}_p^*$, compute the x for which

$$c \equiv g^x \pmod{p}.$$

Abbreviation: DLP.

Clearly, if DLP is known to be solved fast, so is DHP, since

$$g^a, g^b \underset{\text{by DLP}}{\rightsquigarrow} a, b \text{ given } \underset{\text{Modular exponentiation}}{\rightsquigarrow} (g^a)^b \equiv g^{ab} \,(\bmod\, p)$$

However, it is not clear that if a fast solution is known for DHP, it is also for DLP. The general assumption is that these two problems are equivalent. But that is just a conjecture.

In the following, I present one or two (far from ideal, slow) methods for calculating the discrete logarithm. The DLP is widely assumed to be impossible to solve in a reasonable amount of time (for large groups). This assumption is widely used in cryptography.

References

[1] Wikipedia, *Discrete logarithm*, https://en.wikipedia.org/wiki/Discrete_logarithm

12.2 Baby–Step–Giant–Step

Based on Das' book [1], the Baby–Step–Giant–Step method is presented here.

Although this method is slightly out of date, it has the advantage of working not only in $\mathbb{Z}_p^*$, but in any group. However, no faster approach is known in generic groups, such as those based on elliptic curves.

In the following, let $\mathcal{G}$ be a finite multiplicative, cyclic group of size n. Let g be a generator element of $\mathcal{G}$.

Shank's [2] **Baby–Step–Giant–Step** algorithm:

Let m be the ceiling function of $\sqrt{n}$, i.e., $m = \left\lceil \sqrt{n} \right\rceil$. For $i = 1, 2, \ldots, m$ we compute the values g^i's. We create a table, e.g .:

$$\mathcal{G} = \mathbb{F}_{97}{}^* \qquad g = 23 \qquad m = \left\lceil \sqrt{97} \right\rceil = 10$$

i	0	1	2	3	4	5	6	7	8	9
$g^i \pmod{97}$	1	23	44	42	93	5	18	26	16	77

.

This is $O(\sqrt{n}(\log n)^2)$ bit-operations, expressed by m the running time is: $O(m(\log m)^2)$. Then, sort the table by the size of g^i:

i	0	5	8	6	1	7	3	2	9	4
$g^i \pmod{97}$	1	5	16	18	23	26	42	44	77	93

.

Next we compute g^{-m}, where g^{-m} is the inverse of g^m, that is

$$g^{-m} g^m = 1 \quad \text{ in } \mathcal{G}.$$

By Lagrange's theorem $g^n = 1$ for $g \in \mathcal{G}$, where $n = |\mathcal{G}|$. Thus

$$g^{-m} = g^{n-m} \qquad \text{can be computed by } O((\log n)^3) \text{ bit-operations,}$$

since using modular exponentation we get the value of g^{-m}. In our example ($\mathcal{G} = \mathbb{Z}_{97}^*,\ n = 96,\ g = 23,\ m = \lceil \sqrt{97} \rceil = 10$):

$$g^{-m} = 66 \pmod{97}$$

Suppose the DLP we need to solve is as follows: We are looking for the solution of the equation $a = g^x$ in $\mathcal{G}$, where a and g are given. As $j = 1, 2, \ldots, m-1$, we calculate

$$a g^{-jm}.$$

We will see if it matches any of the g^i in the table. This is a $O(\log n)$ pieces of comparisons for each j, so it takes time $O((\log n)^2)$. Studied for all j, the time required is: $O(\sqrt{n}(\log n)^2)$.

If we find i for which

$$ag^{-jm} = g^i \qquad \text{in } \mathcal{G},$$

then

$$a = g^{jm+i},$$

and we have solved the discrete logarithm problem. Since all x can be written in the form $jm + i$, this algorithm always gives a solution.

Disadvantages: slow, with a running time of $O(\sqrt{n}(\log n)^3)$ and a high storage demand of $O(\sqrt{n}\log n)$.

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] D. Shanks, *Class number, a theory of factorization and genera*, Proceedings of the Symposia in Pure Mathematics 20 (1971), 415-440.

12.3 Pollard's ϱ-algorithm for logarithms

This section discusses a variant of the rho-algorithm [3], which is related to integer factorization. Although I describe the algorithm using Das' book [1], Pollard published the first variant of the method in [4].

Let us suppose we would like to solve the DLP problem

$$a = g^x$$

in a cyclic group $\mathcal{G}$, where $|\mathcal{G}| = n$, and g is a generator element of $\mathcal{G}$. To do this, we construct a random walk in $\mathcal{G}$. Starting element:

$$w_0 = g^{s_0}a^{t_0}.$$

Then for $i = 1, 2, 3, \ldots$, we consider the i-th element of the walk in the form

$$w_i = g^{s_i} a^{t_i}$$

Then $w_0, w_1, w_2, \ldots$ behave like a random walk in $\mathcal{G}$. According to the Birthday Paradox [2], there will be a coincidence after $20\sqrt{n}$ steps in this walk with high probability (99%). Then

$$g^{s_i} a^{t_i} = g^{s_j} a^{t_j}.$$

That is

$$a^{t_i - t_j} = g^{s_j - s_i}.$$

Since g is a generator element of $\mathcal{G}$, its order is n. Thus:

$$(t_i - t_j)\mathrm{ind}_g a \equiv s_j - s_i \pmod{n}.$$

If $(t_i - t_j, n) = 1$, then

$$\mathrm{ind}_g a \equiv (s_j - s_i)(t_i - t_j)^{-1} \pmod{n}.$$

To make the method work, we will need a function $f : \mathcal{G} \to \mathcal{G}$ that assigns w_i to w_{i-1}. To achieve this, we fix a relatively small positive integer r and assign an element of the set $\{0, 1, 2, \ldots, r-1\}$ to each $w \in \mathcal{G}$. In addition, we generate r pieces of "multipliers"

$$M_j = g^{\sigma_j} a^{\tau_j}, \qquad j = 0, 1, 2, \ldots, r-1.$$

If for w $(= g^{s_i} a^{t_i})$ we assigned $u \in \{0, 1, \ldots, r-1\}$, then

$$f(w) = w \underset{\substack{\uparrow \\ \text{group multiplication}}}{\cdot} M_u \quad \left(= g^{s_i} a^{t_i} \cdot g^{\sigma_u} a^{\tau_u}\right).$$

In practice, this method works well with $r \approx 20$. Time required for this method: $O(\sqrt{n}(\log n)^3)$. With a clever idea the method of needing storage space can be minimized at the cost of that the time required doubles. We know that after about $20\sqrt{n}$ steps there will be a coincidence with high probability, but then there will be another coincidence at each step, as the walk in the resulting loop goes round and round. So, if an element is fixed after $20\sqrt{n}$ steps, it is most likely in the loop section of the walk. It is enough to store this fixed element in permanent memory and always compare it to the current element of the walk, as there will be a coincidence with this fixed element in another $20\sqrt{n}$ steps (which is greater than the length of the loop).

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] Wikipedia, *Birthday problem*, https://en.wikipedia.org/wiki/Birthday_problem

[3] J. M. Pollard, *A Monte Carlo method for factorization*, BIT Numerical Mathematics. 15 (3) (1975), 331–334.

[4] J. M. Pollard, *Monte Carlo methods for index computation (mod p)*, Mathematics of Computation. 32 (143) (1978), 918–924.

12.4 Pollard's kangaroo algorithm

Pollard described this method in the same paper as the ρ method, see [2]. This approach is a slightly modified version of Pollard's ϱ method, which I describe in the same way as Das' book [1]. The only difference is that there are now two walks:

$$w_i = f(w_{i-1}) \quad \text{and} \quad w_i' = f(w_{i-1}').$$

When the two random walks meet, the walks begin to coincide from this point, forming the Greek letter λ. (As a result, another name for the method is the Pollard's lambda-algorithm.) If

$$w_i = {w_j}',$$

then

$$g^{s_i} a^{t_i} \equiv g^{{s_j}'} a^{{t_j}'},\ g^{s_i} a^{t_i} = g^{{s_j}'} a^{{t_j}'},$$

so

$$a^{t_i - t_j} = g^{s_j' - s_i}.$$

Thus

$$(t_i - {t_j}')^{-1}({s_j}' - s_i) \equiv \mathrm{ind}_g a \pmod{n}.$$

This method is sometimes referred to as the "method of wild and tame kangaroos".

The tame kangaroo walks through each step of its walk, digging holes at each step, and when the wild kangaroo reaches a point in its walk, it falls into the hole and becomes trapped...

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] J. M. Pollard, *Monte Carlo methods for index computation (mod p)*, Mathematics of Computation. 32 (143) (1978), 918–924.

[3] Photo by Pat Whelen from Pexels, https://www.pexels.com/hu-hu/foto/5615406/

12.5 Pohlig–Hellman-algoirthm

Now let $|\mathcal{G}| = n$, where the prime factorization of n is

$$n = {p_1}^{\alpha_1} {p_2}^{\alpha_2} \dots {p_r}^{\alpha_r}.$$

Pohlig and Hellman [2] discovered an algorithm that solves the discrete logarithm problem in

$$O\left(\sqrt{\max\{p_1, \dots, p_r\}} (\log n)^c\right)$$

bit operations.

That is, the method works if the largest prime divisor of n is small. The algorithm appears to have been discovered by Roland Silver, but the result was never published, hence the algorithm is sometimes referred to as the Silver-Pohlig-Hellman algorithm. The following description is based on Das's book [1].

We will compute x by first computing the residue of x modulo $p_i^{\alpha_i}$ for each i, and then using the Chinese remainder theorem to get x.

Assume p is prime and $p^\alpha \mid n$, but $p^{\alpha+1} \nmid n$, and $a = g^x$ in $\mathcal{G}$. We'd like to find the residue of x modulo p^α. Then:

$$\begin{aligned} a &= g^x, && \backslash\hat{}\{n/p^\alpha\} \\ a^{n/p^\alpha} &= \left(g^{n/p^\alpha}\right)^x, \end{aligned}$$

but here the order of g^{n/p^α} is p^α, and thus we need to solve the DLP

$$a' = (g')^x$$

in $\mathcal{G}'$, where $a' = a^{n/p^\alpha}$ and $g' = g^{n/p^\alpha}$. This gives the residue of x modulo p^α, which will be denoted b $r_{p^\alpha}(x)$.

Next write $r_{p^\alpha}(x)$ of the form

$$r_{p^\alpha}(x) = x_0 + x_1 p + x_2 p^2 + \cdots + x_{\alpha-1} p^{\alpha-1}. \tag{12.1}$$

We start with x_0 and then determine x_1, x_2, and so on. Let's take a closer look at one of the steps of the algorithm. Let us suppose we have $x_0, x_1, x_2, \ldots, x_{i-1}$ and we would like to compute x_i. Let

$$\lambda = x_0 + x_1 p + \cdots + x_{i-1} p^{i-1}.$$

We know:

$$a' = (g')^x \qquad \text{in } \mathcal{G}',$$

thus

$$a'(g')^{-\lambda} = (g')^{x_i p^i + x_{i+1} p^{i+1} + \cdots + x_{\alpha-1} p^{\alpha-1}} \qquad \text{in } \mathcal{G}'.$$

Taking $p^{\alpha-i-1}$-th power:

$$\left(a'(g')^{-\lambda}\right)^{p^{\alpha-i-1}} = (g')^{x_i p^{\alpha-1} + x_{i+1} p^{\alpha} + \cdots + x_{\alpha-1} p^{2\alpha-i-2}} \qquad \text{in } \mathcal{G}'.$$

But the order of g' is p^α, thus

$$\left(a'(g')^{-\lambda}\right)^{p^{\alpha-i-1}} = (g')^{x_i p^{\alpha-1}} = \left((g')^{p^{\alpha-1}}\right)^{x_i} \qquad \text{in } \mathcal{G}'.$$

That is, we would like to solve the DLP

$$a'' = (g'')^{x_i}$$

in $\mathcal{G}'' =< (g')^{p^{\alpha-1}} >$, where $a'' = \left(a'(g')^{-\lambda}\right)^{p^{\alpha-i-1}}$ and $g'' = (g')^{p^{\alpha-1}}$. Then the order of $\mathcal{G}''$ is p.

We calculate the digits $x_0, x_1, \ldots, x_{\alpha-1}$ using the prior approach, giving the residue of x modulo p^α.

Then we get x by repeating the process for all prime powers divisors of n and then applying the Chinese remainder theorem to the resulting residues.

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] S. Pohlig, M. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Transactions on Information Theory 24 (1978), 106-110.

12.6 Index-calculus algorithm

I describe the index calculus algorithm method in about the same way as Das does in his book [1]. This method is an adaption of Dixon's random square algorithm [2], but it has historically gone beyond that. Western and Miller [4] created the algorithm, which is based on Kraitchik's ideas [3].

The factor base contains t small primes (t to be determined later):

$$B = \{p_1, p_2, \ldots, p_t\}.$$

The first step is to find a u's for which the least absolute value residue of the power g^u modulo p is a B-number. Say

$$g^u \equiv {p_1}^{\gamma_1} \ldots {p_t}^{\gamma_t} \pmod{p}.$$

From such u you need $s \gg t$ pieces (where $\gg$ means that about $s \geq 2t$). For the indexes, we get the following:

$$\begin{aligned}
&\gamma_{11}\,\mathrm{ind}_g\,(p_1) + \ldots + \gamma_{1t}\,\mathrm{ind}_g(p_t) \equiv u_1 \pmod{p-1}\\
&\quad\vdots\\
&\gamma_{s1}\,\mathrm{ind}_g\,(p_1) + \ldots + \gamma_{st}\,\mathrm{ind}_g(p_t) \equiv u_s \pmod{p-1}
\end{aligned}$$

Here γ_{ij}'s are given. The values of $\mathrm{ind}_g(p_1), \ldots, \mathrm{ind}_g(p_t)$ are obtained by solving the system of linear congruences.

Now $g^x = a$. In the second step, we look for α for which ag^α is a B-number. That is,

$$ag^\alpha \equiv {p_1}^{\gamma_1} \ldots {p_t}^{\gamma_t} \pmod{p}.$$

From this we get $\operatorname{ind} a$:

$$\operatorname{ind} a \equiv -\alpha + \gamma_1 \operatorname{ind} p_1 + \ldots + \gamma_s \operatorname{ind} p_s \pmod{p-1}.$$

References

[1] A. Das, *Computational Number Theory*, CRC Press, 2013.

[2] J.D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. 36 (153) (1981), 255–260.

[3] M. Kraitchik, Théorie des nombres, Gauthier–Villards, 1922

[4] A. E. Western, J. Miller, *Tables of indices and primitive roots*, Royal Society Mathematical Tables, vol 9 (1968), Cambridge University Press.

12.7 Zero-knowledge protocol

The Zero Knowledge Protocol is the name of a cryptographic approach for addressing the discrete logarithm issue that was developed in the early 1980s [2]. The purpose of the approach is to show people that you can solve a cryptographic challenge without revealing the method of solution. A great example of this is the discrete logarithm problem.

Let's say Picara wants to show others that he can solve the DLP

$$a = g^x,$$

but he does not want to tell them anything about x. (The first letter of Picara's name matches to the first letter of the word "prover.")

So, a and g are given, Picara claims to have calculated x, but he has no intention of telling anyone the value of x. Can he convince us that he knows the value of x?

The verifier, let's call him Vivale, checks it as follows:

1st step Picara generates a random number $y < p - 1$ and sends $a' = g^y$ to Vivale.

2nd step: Vivale flips a coin.

If head: Picara sends y to Vivale. Vivale checks to see if $a' = g^y$ is correct.

If tail: Picara sends Vival $x + y \pmod p$. Vivale checks to see if $a'a = g^{x+y}$ is correct.

These two steps are repeated until Vivale is convinced that Picara really knows the solution or Picara falls for bluffing.

Indeed, if Picara does not cheat, that is, if he knows both a' and the related y, then the solutions of the discrete logarithm problems $a'a = g^{x+y}$ and $a = g^x$ are equivalent.

If Vivale flips a coin, he can check whether $a'a = g^{x+y}$ really holds, i.e., whether Picara knows the answer to the corresponding discrete logarithm problem $a = g^x$. However, unlike Picara, Vivale only knows a' while y is a secret, so he does not come any closer to solving $a = g^x$ (unless he knows how to solve another discrete logarithm problem $a' = g^y$, but in theory Vivale does not know any method of solving discrete logarithm problems).

Picara might cheat by generating a random z and then calculating the a' for which $a' = g^z a^{-1}$, i.e., $a'a = gz$ and intending to send z to Vivale as $x+y$. Then, if Vivale flips a coin with a tail, Picara is not busted. Yes, but if Vivale flips a coin with head, he asks about y, which Picara can not answer unless he knows the solution of $a' = g^y$. Vivale, on the other hand, does not get any closer to solving the discrete logarithm problem $a = g^x$ by knowing the equation $a' = g^y$...

If Vivale repeats this verification method enough times, cheating will almost likely be discovered sooner or later. However, if Picara never fails, it is practically certain that he knows the answer to the discrete logarithm problem $a = g^x$. At the same time, whether Vivale flips heads or tailes, he gets no closer to solving the discrete logarithm problem $a = g^x$. As a result, the above method is truly a zero-knowledge protocol.

Further descriptions of zero-knowledge protocols can be found in e.g., [1], [4] is. For number theory related zero-knowledge protocols can be found in e.g., [3].

References

[1] M. Blum, P. Feldman, S. Micali, *Non-interactive zero-knowledge and its applications*, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC 1988), 103–112.

[2] D. Chaum, J.-H. Evertse; J. van de Graaf, *An improved protocol for demonstrating possession of discrete logarithms and some generalizations*, Advances in Cryptology – EuroCrypt '87, in: Proceedings. Lecture Notes in Computer Science. Vol. 304. (1987), 127–141.

[3] N. Koeblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

[4] H. Wu, F. Wang, *A survey of non interactive zero knowledge proof system and its applications*, The Scientific World Journal. 2014.

13 Elliptic Curve Cryptography

We frequently calculate in a finite field $\mathbb{F}_p$ (where p is prime) or its extension $\mathbb{F}_q$ (where q is a prime power) during cryptographic applications of number theory. However, it is frequently useful if the group in which the operations are carried out has a different, more ”random” structure than the ones described above, ensuring more protection against future hacking attempts. Groups defined on elliptic curves are ideal for this.

The history of elliptic curves itself goes back to Diophantus, who, without today's notations, identities, group addition, elementary studied the curve $y(a-y) = x^3 - x$ see [3]. The paper [1] contains a brief but informative survey of this and the history of elliptic curves.

In 1985, Koeblitz [4] and Miller [5] independently proposed the cryptographic usage of elliptic curves. However, widespread cryptographic applications have been available only after 2004.

For readers who are more interested in this issue, I recommend studying the book ”Handbook of Elliptic and Hyperelliptic Curve Cryptography” [2].

We provide algebraic structures during the definition of elliptic curves that offer a large number of Abelian groups with easily countable operations, ”far more” than $\mathbb{F}_q^*$'s. These are the following.

Definition 13.1. a) *Let K be a field, mostly $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{C}$ or $\mathbb{F}_q$ for some q whose characteristic is greater than 3, and let $x^3 + ax + b$ $(a, b \in K)$ is a third-degree polynomial with no multiple roots. An **elliptic curve** over K is the set of points (x, y) (with $x, y \in K$) for which*

$$y^2 = x^3 + ax + b \tag{13.1}$$

plus one more point marked with 0, called “the point at infinity”.

b) *If K is a field with characteristic 2, then an elliptic curve over K is the* *set of points (x, y) for which*

$$y^2 + cy = x^3 + ax + b, \tag{13.2}$$

or

$$y^2 + xy = x^3 + ax^2 + b \tag{13.3}$$

is fulfilled by given $a, b, c \in K$ (now the third-degree polynomial on the right can have multiple roots) + "the point at infinity" 0.

c) *If the characteristic of K is 3, then an elliptic curve over K is the following*

$$y^2 = x^3 + ax^2 + bx + c, \tag{13.4}$$

where the polynomial in the right-hand side has no multiple roots.

In order to be able to define a well-interpreted group on an elliptic curve, the curve must be non-singular, which means that the curve is "smooth", i.e., it has no peaks or intersections. This is equivalent to the fact that all points of the curve are not singular, which is defined as follows:

Definition 13.2. *Let's write* (13.1) *(or similarly* (13.2), (13.3), (13.4)*) in the form $F(x, y) = 0$:*

$$F(x, y) \overset{\text{def}}{=} y^2 - (x^3 + ax + b) = 0.$$

We say that a point (x, y) on the curve is "nonsingular" (or "plain"), if at least one of $\dfrac{\partial F}{\partial y} \neq 0$ or $\dfrac{\partial F}{\partial x} \neq 0$ is satisfied at (x, y).

We do not prove it in this lecture notes, but we can see the condition that it is on the right-hand side of (13.1) and (13.4) polynomial does not have multiple roots, then all points are on the curve is not singular. Using a discriminant, we also know that e.g., in the case of (13.1), this is equivalent to $4a^3 + 27b^2 \neq 0$.

In the remaining parts of this chapter, we mostly studied fields whose characteristic is greater than 3**, i.e., our elliptic curve It is of the form** (13.1):

$$\mathbf{y^2 = x^3 + ax + b}.$$

In most cases, the elliptic curve is defined over a field, but there is also an application where it is over a group $\mathbb{Z}_n$. Here, we are thinking of the factorization method based on elliptic curves, where the elliptic curve is not defined over a field, but over a group $\mathbb{Z}_n$, where (usually) n is a composite number.

The applicability of elliptic curves depends on the fact that the points of the curve form an Abelian group for a suitable operation. For this, an operation must be defined between the points of the curve. For the sake of clarity, we first restrict ourselves to $K = \mathbb{R}$ (the same applies to other fields, only the geometric illustration does not work there), this will be the topic of our next subsection.

References

[1] M. W. Barsagade, Dr. S. Meshram, *Overview of history of elliptic curves and its use in cryptography*, International Journal of Scientific & Engineering Research 5 (4) (2014), https://www.ijser.org/researchpaper/Overview-of-History-of-Elliptic-Curves-and-its-use-in-cryptography.pdf.

[2] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman and Hall/CRC 2005, https://www.hyperelliptic.org/HEHCC/

[3] Heath, Thomas Little, Sir, 1861-1940; Euler, Leonhard, 1707-1783, *Diophantus of Alexandria; a study in the history of Greek algebra*,

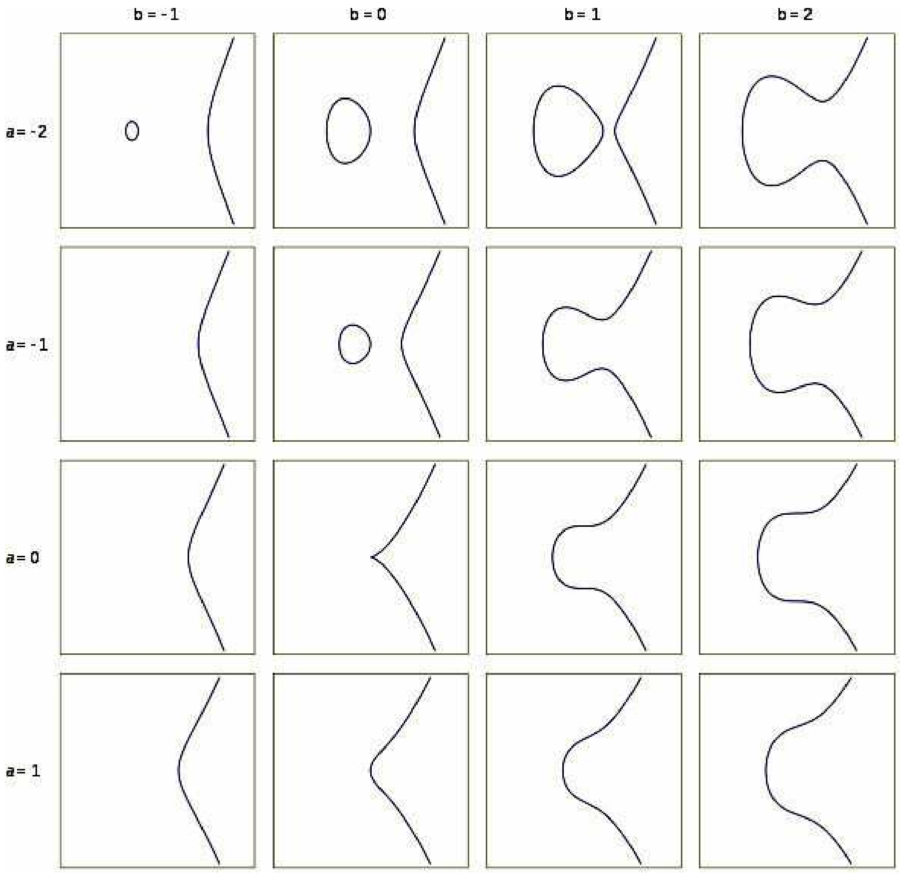
https://archive.org/details/diophantusofalex00heatiala/page/n5/mode/2up.

[4] N. Koeblitz, *Elliptic curve cryptosystems*, Mathematics of Computation. 48 (177) (1985), 203–209.

[5] V. S. Miller, *Use of elliptic curves in cryptography*, Advances in Cryptology — CRYPTO '85, Proceedings. CRYPTO, Lecture Notes in Computer Science. Vol. 85 (1985), 417–426.

13.1 Elliptic curves over $\mathbb{R}$

In this subsection, we define an addition on the points of elliptic curves over $\mathbb{R}$. However, before moving on to the definition of addition, we show a figure of the possible forms of elliptic curves over $\mathbb{R}$.



Then we can define the addition. First, we give a definition based on illustrative geometry.

Definition 13.3. *Let E be an elliptic curve over $\mathbb{R}$, and let P and Q be two points of E. The point at infinity is still denoted by 0. Then:*

1. $P + 0 = 0 + P = P$.

2. *If $P = (x_P, y_P) \neq 0$ and $Q = (x_Q, y_Q) \neq 0$, then*

a) *For $x_P \neq x_Q$, the line connecting P, Q intersects the curve at a point $R = (x_R, y_R)$. Let $P + Q$ be the mirror image of R on the x axis, i.e.,*

$$P + Q \overset{\text{def}}{=} (x_R, -y_R).$$

b) *In the case of $x_P = x_Q$ we have one of the following two cases.*

b$_1$) *If $y_P = y_Q$, i.e., $P = Q$, then the tangent in P intersects the curve at only one other point $R = (x_R, y_R)$, let*

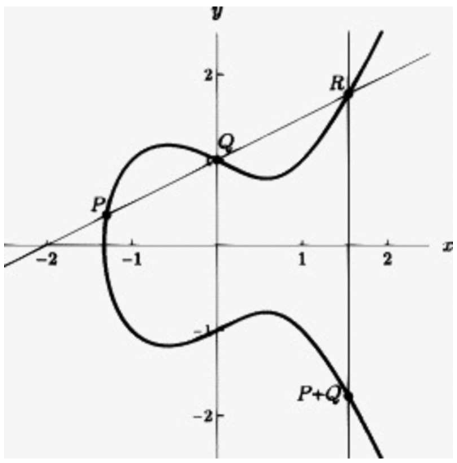
$$P + Q = 2P \overset{def}{=} (x_R, -y_R)$$

(if P is an inflection point: $R = (x_R, y_R) = P$).

b$_2$) *If $y_P = -y_Q$ then*

$$P + Q \overset{\text{def}}{=} 0$$

.



The proof that the addition given in Definition 13.3. is well defined and that this addition forms an Abel group at the points E can be found e.g., in Silverman's book [1]. It is also clear that the unit element in this group is 0, the point at infinity since

$$P + 0 = 0 + P.$$

The inverse of an element $P = (x, y)$ is $-P = (x, -y)$.

The following formulas can be calculated using elementary geometry (see [1]):

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, where $P \neq \pm Q$, then the slope of the line PQ is:

$$m = \frac{y_p - y_Q}{x_P - x_Q}.$$

The line PQ intersects the curve at point $R = (x_R, y_R)$, here:

$$\begin{aligned} x_R &= m^2 - x_P - x_Q \qquad (13.5)\\ y_R &= y_p + m(x_R - x_P), \end{aligned}$$

or equivalent

$$y_R = y_Q + m(x_R - x_Q).$$

If $S = P + Q$, then the point S is the mirror image of the point R on the x axis, so for the coordinates of $S = P + Q = (x_S, y_S)$ we have

$$\begin{aligned} x_S &= m^2 - x_P - x_Q,\\ y_S &= -y_p + m(x_P - x_R) = -y_Q + m(x_Q - x_R). \end{aligned}$$

Let us see the proof of (13.5). Our curve is of the form $y^2 = x^3 + ax + b$, while the line PQ is of the form $y = mx + d$. Both the line PQ and the curve have the points P, Q and R, so x_P, x_R and x_Q are solutions of

$$(mx + d)^2 = x^3 + ax + b.$$

By rearranging

$$x^3 - m^2x^2 - 2mdx + ax + b - d^2 = 0.$$

Then x_P, x_R and x_Q are roots of the above equation, so according to the relationship between roots and coefficients:

$$x_P + x_Q + x_R = m^2,$$

which proves (13.5).

If $P = Q$ by adding the points P and Q of the curve, then the slope of the curve at the point P must be written, this is

$$m = \frac{3x_p^2 + a}{2y_p}.$$

Then for the coordinates of $S = P + Q = (x_S, y_S)$:

$$\begin{aligned} x_S &= m^2 - x_p - y_p, \\ y_S &= -y_p + m(x_P - x_R) = -y_Q + m(x_Q - x_R). \end{aligned}$$

Finally, for $P = -Q$, $P + Q = 0$.

References

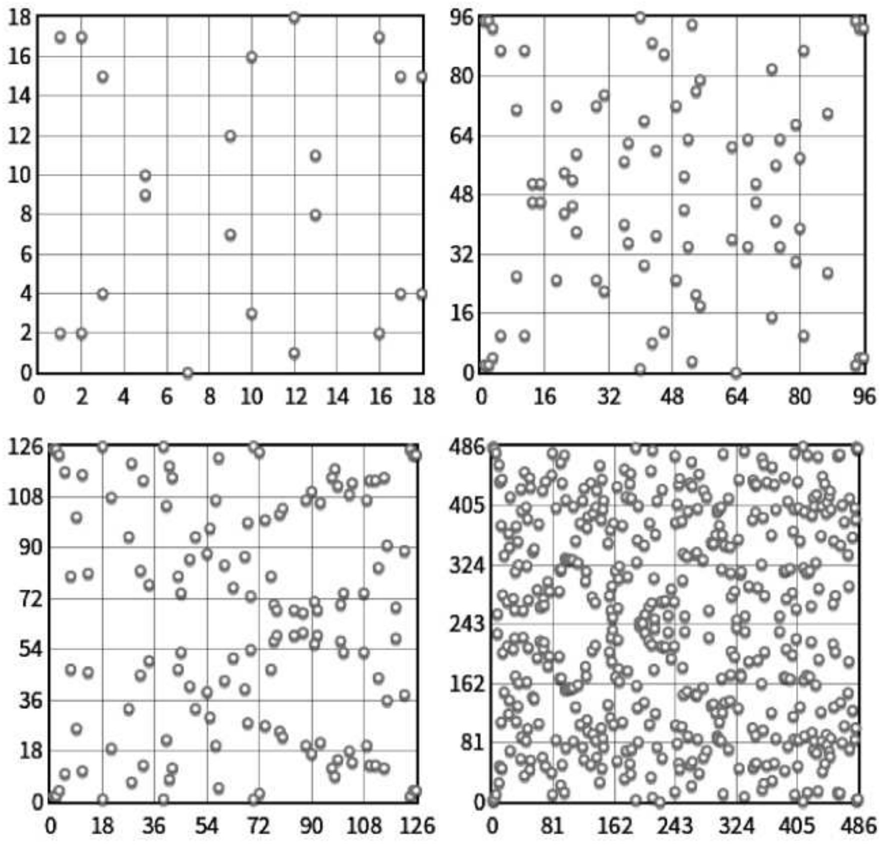
[1] J. H. Silverman, *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, Vol. 106, Springer-Verlag, 1986.

[2] Figure, Wikipedia, *Possible shapes of elliptic curves*, https://en.wikipedia.org/wiki/Elliptic_curve

[3] Figure, *Group law for an elliptic curve*, https://www.researchgate.net/figure/The-group-law-for-an-elliptic-curve-P-Q-R-The-points-P-and-Q-sum-to-the-point-R_fig1_23552588

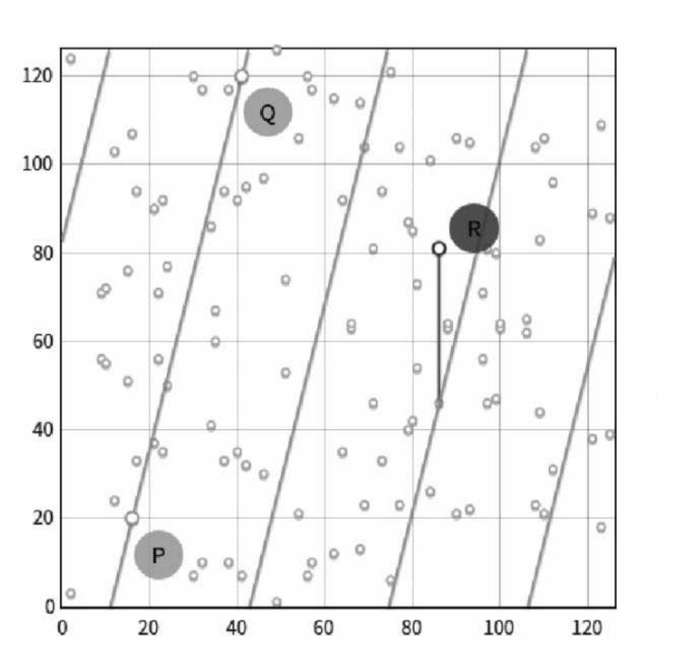
13.2 Elliptic curves over $\mathbb{F}_p$

In cryptographic applications, elliptic curves are usually defined over a field $\mathbb{F}_p$. Unfortunately, at that time the nice continuous curve disappears, instead we get a figure consisting of discrete points. In the subsection, we rely on Corbellini's online lecture note [1], the figures in the chapter also come from him. The following figure shows the points of the elliptic curve $y^2 = x^3 - 7x + 10$ over the field $\mathbb{F}_p$, where p is $19, 97, 127$ and 487, respectively.



Note that there are at most 2 points with the given x coordinate on the curve, and the figure is always symmetric to the line $y = p/2$.

It is clear that the figure of the "line" passing through the points P and Q of the curve will be completely different from the real case. But fortunately, lines have a coordinate geometric formula, $ax+by=c$, with which we can also work over $\mathbb{F}_p$. Now again the line passing through he points P, Q intersects the curve at a third point R, whose reflection on the line $y = p/2$ will be $P+Q$. This is illustrated by the following figure, which shows the sum of the points $P = (16, 20)$ and $Q = (41, 120)$ on the elliptic curve $y^2 = x^3 - x + 3$ (mod 127).



It can be seen that if the field of the elliptic curve is finite, the geometric formulation will be difficult to handle. In this situation, though, the addition described by algebraic formulas can be transferred without any further explanation:

Definition 13.4. *If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, where $P \neq \pm Q$, then the slope of the line PQ is:*

$$m = \frac{y_p - y_Q}{x_P - x_Q}.$$

If $S = P + Q$, then the point S is the mirror image of the point R on the x axis, so for the coordinates of $S = P + Q = (x_S, y_S)$ we have:

$$\begin{aligned} x_S &= m^2 - x_P - x_Q, \\ y_S &= -y_p + m(x_P - x_R) = -y_Q + m(x_Q - x_R). \end{aligned}$$

If $P = Q$, then the slope of the curve at the point P must be written, this is

$$m = \frac{3x_p^2 + a}{2y_p}.$$

Then for the coordinates of $S = P + Q = (x_S, y_S)$:

$$\begin{aligned} x_S &= m^2 - x_p - y_p, \\ y_S &= -y_p + m(x_P - x_R) = -y_Q + m(x_Q - x_R). \end{aligned}$$

Finally, for $P = -Q$, $P + Q = 0$.

A very important question is how many points of an elliptic curve E defined over a finite field $\mathbb{F}_q$ can have. According to Hasse's theorem [2] this value can only differ from $q + 1$ by $2\sqrt{q}$:

Theorem 13.5. (Hasse)

$$|\text{card } E(\mathbb{F}_q) - (q + 1)| \leq 2q^{1/2}.$$

As a conjecture, the theorem was formulated by Artin in his thesis in 1924. It was only 12 years later that Hasse managed to prove it, and his proof was presented in a series of articles [2]. Weil [5] further generalized the estimate to curves more general than elliptic curves.

It is also known that the structure of the group $E(\mathbb{F}_q)$ is either cyclic or the direct product of two cyclic groups.

In cryptographic applications, we often need the exact order (i.e., exact number of elements) of the elliptic curve. Schoof [3], [4] gave a polynomial algorithm for this, however, the description and proof of the algorithm goes beyond the scope of this lecture notes.

A brief description of elliptic curve cryptography (ECC) can be found on the related Wikipedia page [6]. In this lecture notes, we take a look at some chapters of the ECC without claiming to be complete. Thus, we will talk about the analog of Diffie-Hellman key exchange, digital signature based on elliptic curves and Lenstra's factorization algorithm.

References

[1] A. Corbellini, *Elliptic Curve Cryptography: finite fields and discrete logarithms*, `https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/`

[2] H. Hasse, *Zur theorie der abstrakten elliptischen funktionenkörper.* I, II & III, Crelle's Journal, 1936 (175), 55-62, 69-88, 293-208.

[3] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*. Math. Comp., 44 (170), 483–494, 1985, `http://www.mat.uniroma2.it/~schoof/ctpts.pdf`.

[4] R. Schoof, *Counting points on elliptic curves over finite fields*, J. Theor. Nombres Bordeaux 7, 219–254, 1995, `http://www.mat.uniroma2.it/~schoof/ctg.pdf`.

[5] A. Weil, *Numbers of solutions of equations in finite fields*, Bulletin of the American Mathematical Society, 55 (5) (1949).

[6] Wikipedia, *Elliptic-curve cryptography*, `https://en.wikipedia.org/wiki/Elliptic-curve_cryptography`.

[7] Ábra, A. Corbellini, *Elliptic Curve Cryptography: finite fields and discrete logarithms*, `https://andrea.corbellini.name/2015/05/`

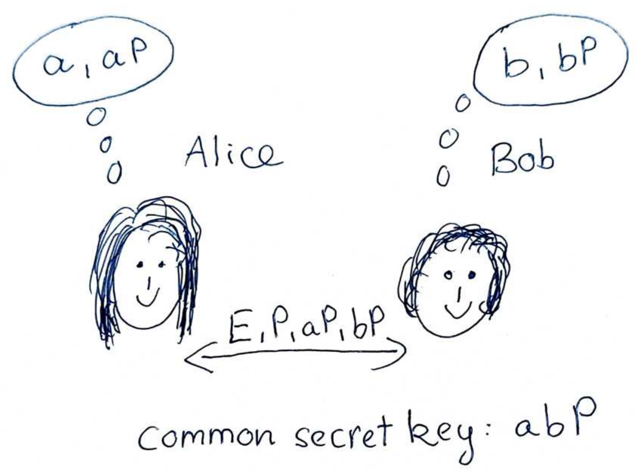
23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/

13.3 Diffie-Hellman key exchange on elliptic curves

Suppose that Alice and Bob want to agree on a shared secret key, which is now symbolized by a point S of an elliptic curve over $\mathbb{F}_p$. (We can easily make a point of $\mathbb{F}_p$ correspond to S, e.g., we take the x or y coordinates of S.) Eve watches the channel on which Alice and Bob communicate. The question is whether Eve can figure out the shared secret key S from what she listens to on the channel.

The analog of Diffie-Hellman key exchange for elliptic curves is the following:

Alice and Bob agree on a public elliptic curve E over $\mathbb{F}_p$ and a point P of it whose order is a large prime. Alice thinks of a natural number a, Bob thinks of a natural number b. They both keep the number they thought a secret, they don't tell anyone, not even each other. Alice calculates aP, Bob calculates bP. The values of aP and bP are sent to each other on the channel, and the shared secret key is abP.



But how to quickly calculate the multiple of an arbitrary point P on the elliptic curve, say cP? This is done using a double-and-add algorithm. Let's write c as a sum of two powers:

$$c = 2^{a_1} + 2^{a_2} + \cdots + 2^{a_r},$$

where 2^{a_1} is the largest power of two. First, we calculate the multiples of the form $P_i = 2^i P$, where $i = 1, 2, \ldots, a_1$, with the following recursion:

$$\begin{aligned} &P_0 = P, \\ &P_i = P_{i-1} + P_{i-1} \qquad \text{ha } i \geq 1. \end{aligned}$$

Then:

$$cP = P_{a_1} + P_{a_2} + \cdots + P_{a_r}.$$

This algorithm is very fast and polynomial time. (Here we note that Morain and Olivos [7] noticed that if we write c in the binary number system, but with digits $\{0, 1, -1\}$, and we try to make this form as short as possible in the context the non-0 digits, then a $25-30\%$ speed increase can be achieved. This is due to the fact that addition and subtraction on elliptic curves take almost the same amount of time.)

So, knowing a, Alice can quickly calculate aP, and Bob, knowing b, can calculate bP. Both of them can quickly calculate the common secret key, since Alice adds aP exactly b times, and Bob adds bP exactly a times.

However, it is generally assumed that in order to find out the common secret key, Eve needs to be able to determine a or b, but she only knows the values of the points P, aP and bP on the curve. This is a discrete logarithm problem, discussed in more detail in Chapter 12.

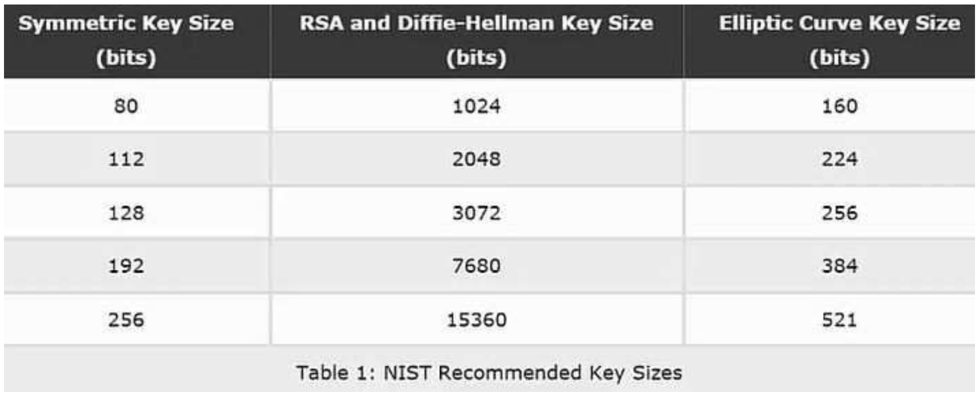
The algorithm still has some questionable points. For example, Alice and Bob, how do they find a high-order common point P? It happens the other way around than we think. So it is not the case that we take a random P point, calculate its order, and if this order is large, we keep P, and if it is not large we try a small new random P point. Based on Schoof's algorithm, we can determine the order of the curve [9], [10], but this is not suitable for determining the order of any point on the curve. Compared to this, we are moving in the opposite direction. We calculate the order of the curve, let it be N. We take a large prime divisor n of this N. Next, we search for a point P of order n. For this we choose a random R. We calculate:

$$P = \frac{N}{n} R.$$

If it is 0, then we choose a new random R, if it is not 0, then the order of P is n.

It is easy to think that any random curve is good for the cryptographic application of elliptic curves. This is far from the case. So, for example, based on Smart's attack [11], if the order of the curve over $\mathbb{F}_p$ is exactly p, then the discrete logarithm can be solved in linear time. We can also think of the well-known MOV attack [6] (e.g., when the order of the curve is $p + 1$). Fortunately, the number of such, so-called anomalous curves is relatively small. To exclude the above and similar attacks, in 1999 NIST published a publication [8] about elliptic curves that were considered safe at the time. You can also read about attacks against cryptography based on elliptic curves and the weak cryptographic properties of certain elliptic curves, e.g., [1], [4], [5] and [3]. The last reference is also related to quantum computer attacks.

An important question is why is an ECC encryption system based on an elliptic curve better or more reliable than a traditional encryption system? The size of ECC encryption keys is much smaller than those used before. This is illustrated in the table published by NIST:



Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

You can see the size difference between the keys. Thus, smaller electronic devices, e.g. in the case of mobile phones, encryption based on *ECC* is more obvious.

References

[1] , I. Biehl, B. Meyer, V. Müller, *Differential fault attacks on elliptic curve cryptosystems*, Advances in Cryptology – CRYPTO 2000. Lecture Notes in Computer Science Vol. 1880. (2000), 131–146, https://www.iacr.org/archive/crypto2000/18800131/18800131.pdf.

[2] A. Corbellini, *Elliptic Curve Cryptography: finite fields and discrete logarithms*, https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/

[3] L. De Feo; P. Jao, Plut, *Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies*, Cryptology ePrint Archive, Report

2011/506. IACR. Archived from the original on 2014-05-03. Retrieved 3 May 2014:
https://eprint.iacr.org/2011/506

[4] M. Hedabou, P. Pinel, L. Beneteau, *A comb method to render ECC resistant against Side Channel Attacks*, https://eprint.iacr.org/2004/342.pdf

[5] *How to design an elliptic-curve signature system*, Cr.yp.to: 2014.03.23, http://blog.cr.yp.to/20140323-ecdsa.html

[6] A. Menezes, T. Okamoto, S. A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory. 39 (5) (1993), 1639–1646.

[7] F. Morain, J. Olivos, *Speeding up the computations on an elliptic curve using addition-subtraction chains*, RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications 24.6 (1990), 531-543, http://eudml.org/doc/92374

[8] National Institute of Standards and Technology, *Recommended Elliptic Curves for Federal Government Use*, July 1999. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.doc

[9] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*. Math. Comp., 44 (170), 483–494, 1985, http://www.mat.uniroma2.it/~schoof/ctpts.pdf.

[10] R. Schoof, *Counting points on elliptic curves over finite fields*, J. Theor. Nombres Bordeaux 7, 219–254, 1995, http://www.mat.uniroma2.it/~schoof/ctg.pdf.

[11] N. P. Smart, *The discrete logarithm problem on elliptic curves of trace one*, 1997, http://www.hpl.hp.com/techreports/97/HPL-97-128.html.

13.4 Elliptic Curve Digital Signature Algorithm

Digital signatures are extremely important in today's IT world. In addition to traditional handwritten signatures, it is now often necessary to electronically verify that a given document originates from us.

The idea of a digital signature was proposed by Diffie and Hellman in [2], but did not become feasible until when RSA was published. It took decades for the digital signature to be recognized as legal equivalent to the traditional one.

Since then, nathematicians have invented many algorithms for digital signatures. In accordance with the topic of this chapter, we will now discuss elliptic curve digital signature algorithms (ECDSA).

ECDSA was first proposed by Vanstone [4] in 1992, adopted by ISO (International Standard Organization) in 1998, and ANSI (American National Standard Institute) in 1999. A paper summarizing the main cryptographic properties of ECDSA was published in 2001 by Johnson, Menezes and Vanstone [3]. In this chapter, we only briefly analyze ECDSA, based on the work of Corbellini [1].

Let's say that Alice wants to electronically sign a message. In the case of ECDSA, this requires a public prime p, an elliptic curve E over $\mathbb{F}_p$, and a point G of prime order in it. These are all public. Denote the order of G by n. Then n is a prime number. (We note here that the order of most standard elliptic curves in use are prime numbers, i.e., n is the same as the order of the elliptic curve E, so there is no need to factorize the order of E.) Alice has another secret key, which is a natural number between 1 and n, denote it by d_A. Alice's public key is the following point of the elliptic curve:

$$H_A = d_A G.$$

It is important that anyone in the world can be convinced that Alice is the person who signed the message.

For example, in our narrative, let's call the other character Bob, who wishes to ensure that the sent message truly has been signed by Alice. He can use it for this the elliptic curve E, the point G on it, the signed document, and Alice's public key H_A.

Most of the time, Alice does not sign the original message (because it's usually too long), but a shortened version of it with a hash function. Cryptographically secure hash functions must satisty many requirements, e.g., you can read about it in more details in [5]. The shortened version of the message with the hash function is an integer whose binary length cannot be greater than the binary length of n (where n is the order of the subgroup generated by point G). Let z denote the hash function shortened version of the message. Then $z \in \mathbb{N}$ and z can be greater than n, but its binary form cannot be longer than n.

Alice's algorithm for signing is the following:

1. Alice chooses a random k from the set $\{1, 2, 3, \ldots, n-1\}$.

2. Alice calculates the point $P = kG$ of the elliptic curve E.

3. Alice denotes the coordinate of the point P on the x axis with r: $r = x_p$.

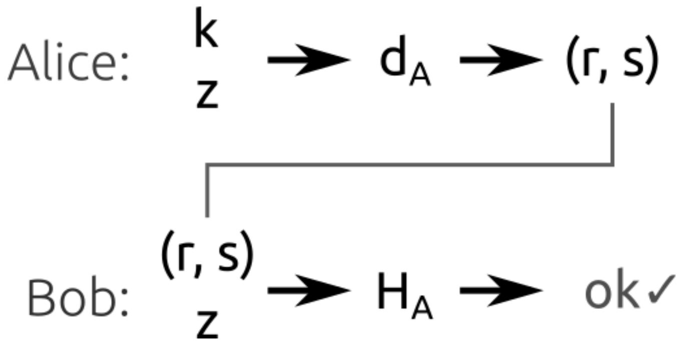
4. If $r = 0$, she chooses another random k and tries again, i.e., she goes back to step 1.

5. She calculates $s \equiv k^{-1}(z + rd_A) \pmod{n}$ (where d_A is Alice's secret key, and k^{-1} is the multiplicative inverse of k modulo n.)

6. If $s \equiv 0 \pmod{n}$, Alice chooses another random k and tries again, i.e., goes back to step 1.

The digital signature is the pair (r, s).

(Note that the operations in Step 5 are the usual addition and multiplication in $\mathbb{Z}_n$. In contrast, in Step 2 the addition is defined over the points of the elliptic curve, and $kG = \underbrace{G + G + \cdots + G}_{k \text{ pieces of } G}$.)



We can then move on to how Bob can verify the authenticity of the signature.

1. Bob calculates the number $u_1 = s^{-1}z \pmod{n}$.

2. Bob defines the number $u_2 = s^{-1}r \pmod{n}$.

3. Bob calculates the point $P = u_1G + u_2H_A$ of the elliptic curve.

The digital signature is valid if $r = x_P$.

At first sight, it is unclear why this control method works, but when the equations are put together, everything becomes clear:

We know that $P = u_1G + u_2H_A$ and $H_A = d_AG$, thus

$$\begin{aligned} P &= u_1G + u_2H_A \\ &= u_1G + (u_2d_A)G \end{aligned}$$

$$= (u_1 + u_2 d_A)G.$$

If we recall the definition of u_1 and u_2, i.e., $u_1 = s^{-1}z \pmod n$ and $u_2 = s^{-1}r \pmod n$, then

$$\begin{aligned} P &= (u_1 + u_2 d_A)G \\ &= (s^{-1}z + s^{-1}rd_A)G \\ &= s^{-1}(z + rd_A)G. \end{aligned}$$

Previously, s was given by $k^{-1}(z + rd_A)$, so $ks = z + rd_A$, from which

$$P = kG.$$

This proved the correctness of the control method.

Today, ECDSA is used e.g. during the TLS protocol that provides protection for communication over the Internet. Its safety depends on the fact that there is no fast algorithm for solving the discrete logarithm problem on elliptic curves. (That is, from the equation $H_A = d_A G$, knowing only H_A and G, there is no fast algorithm to calculate the private key d_A.) Thus, if the parameters of the elliptic curve are large enough, the discrete logarithm problems given on the given elliptic curve cannot be solved even with the help of computers.

It is important that the random k used in the algorithm has to be used only once. If it is used repeatedly, Alice's private key becomes quickly computable. In the same way, it is very important to choose k in a truly random way, because if k can be predicted in some way, then d_A can be easily determined. You can read more about this in Corbellini's note [1] and on the ECDSA-related Wikipedia page [6].

References

[1] A. Corbellini, *Elliptic Curve Cryptography: ECDH and ECDSA*, https://andrea.corbellini.name/2015/05/30/elliptic-curve-

cryptography-ecdh-and-ecdsa/

[2] W. Diffie, M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory. 22 (6) (1976), 644-654.

[3] D. Johnson, A. Menezes, S. Vanstone *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, International Journal of Information Security 1, 36–63 (2001).

[4] S. Vanstone, *Responses to NIST's Proposal*, Communications of the ACM 35 (1992), 50-52.

[5] Wikipedia, *Cryptographic bash function*, http://en.wikipedia.org/wiki/Cryptographic_hash_function.

[6] Wikipedia, *Elliptic Curve Digital Signature Algorithm*, https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

[7] Ábra, A. Corbellini, *Elliptic Curve Cryptography: ECDH and ECDSA*, https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/

13.5 Lenstra's factorization method based on elliptic curves

Until now, elliptic curves were always defined over a field. But there is an important case when the elliptic curve is not defined over a field, but over a group $\mathbb{Z}_n$. This is Lenstra's [3] factorization algorithm. If n is composite, the elliptic curve will have points that cannot be added with the usual addition, because the denominator in the slope of the line passing through the two points will not be invertible modulo n. However, this at first sight unpleasant property is actually very useful: it can lead to finding a real divisor of n, because by calculating the greatest common divisor of this denominator and n, we get a divisor of n.

The starting point of the factorization method is Pollard's $p-1$ method [4]. This algorithm will only be described very briefly, since there are many faster methods available today. However, this is the method that is the starting point of Lenstra's [3] algorithm, which is still considered one of the fastest factorization algorithms.

The algorithm works if n is a prime divisor of p and $p-1$ is a B-smooth number, i.e., $p-1$ is a divisor of every prime power $\leq B$. In other words, $p-1 \mid [1,2,3,\ldots,B]$, where the square bracket denotes the least common multiple.

From now on

$$L_B = [1,2,3,\ldots,B].$$

By the little-Fermat theorem, if p is prime and $(a,p)=1$ then

$$a^{p-1} \equiv 1 \pmod{p}.$$

If $p-1$ is a smooth number related to B, i.e., $p-1 \mid L_B$, then by raising the above congruence to the power of $L_B/(p-1)$ we get

$$a^{L_B} \equiv 1 \pmod{p}.$$

That is, for $p \mid n$ we have

$$p \mid (n, a^{L_B}-1).$$

Thus, if we use the Euclidean algorithm to calculate the greatest common divisor of n and $a^{L_B}-1$, if it is smaller than n, we get a real divisor of n. If we do not succeed, we choose another a. In the algorithm, the value of B is continuously increased: $B = 1,2,3,\ldots$. The interesting feature of Pollard's paper [4] is that he determines the time requirement of the algorithm based on the operating time of Turing machines.

Pollard's $p-1$ method works fast if $p-1$ is a smooth number related to B, in case of relatively small B. Here $p-1$ is the order of the multiplicative group $\mathbb{Z}_p^*$. Lenstra moved from this group to groups of elliptic curves defined over $\mathbb{Z}_p$. This will be good because the order of such a group is of the form

$$p + 1 \pm k$$

based on Hasse's theorem [2], where $k \leq 2\sqrt{p}$. If the order $p + 1 \pm k$ is a smooth number related to B, the algorithm will probably give a real divisor of the composite number n.

In the following, we describe a simplified version of Lenstra's algorithm. Let n be a composite number that we would like to factor, and let B be a suitably chosen constant depending on n. First, we take a point $P = (x_P, y_P)$ of Z_n^2, then we choose integers $a \neq 0$ and b for which

$$6(4a^3 + 27b^2)$$

is relative prime to n-hez. Moreover let

$$y^2 = x^3 + ax + b$$

be an elliptic curve such that it contains the point P, i.e.,

$$b = y_p^2 - x_p^3 - ax_P.$$

Then, by repeated addition, we consider the points

$$P,\ 2!P,\ 3!P, \cdots, B!P$$

of the elliptic curve. Suppose that we study the same elliptic curve and the same points, but now over $\mathbb{Z}_p$. Now if the order of the curve, for s we have

$$s \mid B!,$$

then, based on Lagrange's theorem, $B!P$ gives the point 0 (now the elliptic curve is studied over $\mathbb{Z}_p$). What does it mean? During the repeated addition, there was a point R and S of the curve (where $R = aP$ and $S = bP$), when the denominator of the slope of the line passing through R and S is divisible by p, i.e., $p \mid x_R - x_S$. That is, if we add R and S on the curve defined over $\mathbb{Z}_n$, then the result of the addition is either 0, or we simply cannot add the two points, since in the formula of the line passing through them $x_R - x_S$ is not invertible. If we cannot add the two points, then

$$1 < (x_R - x_S, n) < n,$$

and by calculating it, we get a real divisor of n. The points $P,\ 2!P,\ 3!P, \cdots, B!P$ are calculated by repeated addition. Indeed, if $P_i = i!P$, then $P_i = iP_{i-1}$ can be calculated quickly, e.g., with the double-and-add algorithm. If we cannot add two points during the procedure, say R and S, then by calculating

$$1 < (x_R - x_S, n) < n$$

we get a real divisor of n.

Let us return to the order of the elliptic curve defined over $\mathbb{Z}_p$ which is s. According to Hasse's theorem [2], $s = p + 1 \pm k$, where $k \leq 2\sqrt{p}$. If B is large enough for and we have

$$s \mid B!,$$

then $B!P$ is the 0 element over $\mathbb{Z}_p$, in which case either $B!P$ is the 0 element also over $\mathbb{Z}_n$, or the addition is undefined, and in this case we get a real divisor of n.

During the procedure, it may also happen that during additions over $\mathbb{Z}_n$ a

$$P,\ 2!P,\ 3!P, \cdots, B!P$$

points are all elements of the elliptic curve. In this case, we choose a new point P and a new elliptic curve, until we get a real divisor of n.

The time requirement of the algorithm is $\exp\Big((\sqrt{2} + o(1))(\log p)^{1/2} \log\log p\Big)$, where p is the smallest prime factor of n .

A famous competition was the RSA Factorization Challenge (see [1], [7]), which was published by the RSA Laboratory in 1991, where the factorization of semiprimes (the product of two large primes) given by them was the task, in return for a significant fee. For the factorization of a single large number, they paid up to $20,000 at that time. The last two, RSA-240 and RSA-250 (12-13 years after the deadline) were factored in 2019 and 2020 using the combined application of general number field sieve and elliptic curves factorization see [5], [6]. After the results were published, experts in the applied field suggested that it is safe to use modulus with a length of at least 2048 bits when applying RSA (although, the first criterions requiring such a large modulus already appeared around the end of the twentieth century).

Unfortunately, the awarding of the competition ended in 2007, but we hope that there will be a similar call in the future...

Thank you for your attention!

References

[1] K., Burt (18 Mar 1991), *Announcement of "RSA Factoring Challenge"*, Retrieved 8 March 2021, https://groups.google.com/u/0/g/sci.crypt/c/AA7M9qWWx3w/m/EkrsR69CDqIJ?pli=1.

[2] H. Hasse, *Zur theorie der abstrakten elliptischen funktionenkörper.* I, II & III, Crelle's Journal, 1936 (175), 55-62, 69-88, 293-208.

[3] H. W. Lenstra, Jr., *Factoring Integers with Elliptic Curves*, Annals of Mathematics Second Series, 126, (3) (1987), 649-673, https://www.jstor.org/stable/1971363.

[4] J. M. Pollard, *Theorems of factorization and primality testing*. Proceedings of the Cambridge Philosophical Society. 76 (3) 521–528.

[5] E. Thomé et al. (December 2, 2019), *795-bit factoring and discrete logarithms*, cado-nfs-discuss (Mailing list), https://sympa.inria.fr/sympa/arc/cado-nfs/2019-12/msg00000.html.

[6] P. Zimmermann et al. (28 February 2020), *Factorization of RSA-250*, cado-nfs-discuss (Mailing list), https://sympa.inria.fr/sympa/arc/cado-nfs/2020-02/msg00001.html.

[7] Wikipedia, *RSA Factoring Challenge*, https://en.wikipedia.org/wiki/RSA_Factoring_Challenge.