Number theory and cryptography in your smartphone

Rubí Ipiña and José A Vallejo
linarubi.campos,jvallejo@fc.uaslp.mx
Faculty of Sciences
State University of San Luis Potosí
Lat Av Salvador Nava s/n CP 78290 San Luis Potosí (SLP)
México

March 31, 2017

Abstract

We present the basics of a course on number theory and cryptography with a hands-on approach, using the capabilities of modern-day smartphones to implement the computational algorithms in real time, with the aid of the free software Maxima available in Android™ platforms[1].

1 Introduction

The sentence 'Mathematics is the queen of the sciences and number theory is the queen of Mathematics' is attributed to Gauss [16]. Number theory always had an halo of purity, even praised for its lack of applications[2]. The tremendous development of applied mathematics in the second half of the 20^{th} century somewhat eclipsed it, but nowadays a kind of a renaissance has taken place as number theory plays a prominent role in such areas as cryptography [6, 7], computer science [17] or theoretical physics [1, 8]. This interplay with more "applied" areas is, of course, not exclusive of number theory, but it is significant in this case because it shows that even the most abstract form of mathematics can find some interesting use in everyday life. And this is an exceptional opportunity to strengthen both, abstraction and applicability. Therefore, its inclusion in any scientific curriculum seems desirable and justifiable.

One of the advantages of number theory from the pedagogical point of view, is that a course on it can be taught at a variety of levels, ranging from the most elementary to the most advanced. Moreover, it can be used to put emphasis on applications. An additional benefit,

[1]Maxima is a Computer Algebra System licensed under the GNU General Public License. Android is a trademark of Google Inc.

[2]In this regard, it is commonplace to cite G. H. Hardy [3], although he was particularly referring to the applicability of Mathematics in the military industry, and explicitly stating a bound in time:

> 'No one has yet discovered any warlike purpose to be served by the theory of numbers or relativity, and it seems unlikely that anyone will do so for many years'.

which has the utmost importance for us, is that it allows to use modern technologies as an efficient tool in our teaching. There are lots of papers discussing the convenience of using calculators, computers and other devices (what is collectively denoted 'mobile learning') in the classroom, and we do not intend here to deepen into this topic (a good review is given in [14]). Based on our experience, the students participate and are more interested in a course based on the use of computers and software. Moreover, if the topics cover material related to modern technology that allows them to understand the way this technology works and the mathematics behind it, then such a course becomes a gratifying experience and a motivation for further study.

If we ask a classroom of first-year university students what their favorite gadget is, the most probable answer will be the smartphone. Any topic related to them, touched upon in a lecture, quickly becomes a trending topic (to use modern parlance) among students, and the corresponding course immediately receives the #*interesting* hashtag. Students have not had enough time to overcome their high school habits, and they still show this urge for classifying anything into their own schemes. It is our belief that we can use this state of affairs to attract them to the more prosaic maths, and this paper describes our proposal along the lines described above, that is, a course on basic number theory with applications to cryptography based on the use of the free software Maxima in Android devices.

Nihil novum sub sole: There is nothing new under the sun, and this work owes a lot to the nice book by A. McAndrew [11]. To our knowledge, this is a pioneering text, in that it conjugates an elementary (but highly non-trivial) approach with the use of free software to study cryptography. When we faced the task of selecting a good textbook for the course, this was the obvious choice. However, it uses Sagemath (previously SAGE [12]), whose installation in a computer requires above 1.5Gb of space, and, although it can be used on-line from a smartphone [13], needs a stable internet connection. This fact renders it unusable in our case, so we decided to use Maxima [9] instead, with the Android port Maxima on Android (MoA) [4], which can be fully installed in any smartphone (it only requires 45Mb of internal storage plus 60Mb of additional storage that can be placed in a external card) and used off-line. Fortunately, McAndrew's book is written in such a way that there is no problem in adapting its contents to Maxima or any other CAS. Nevertheless, for the sake of originality, we include here some examples not present in [11] (or, as in the case of the Elgamal cryptosystem, implemented in a different way). Another text that has proven useful is Stinson's [15].

The next sections describe in some detail the prerequisites (both technological and mathematical) to develop the course, the main topics covered, and some examples of implementation in Maxima. The screenshots show examples that were written in a low-end Samsung Galaxy Core2 (Android 4.4.2), during the Spring semester of 2016, in an optional course entitled 'Topics on Applied Mathematics', given to students of Applied Mathematics in the State University of San Luis Potosí (México). Some data in the screenshots are different from those in the text because of this (of course, each time a function generating (pseudo)random numbers is run, the results are different).

2 Maxima on Android

Every Computer Algebra System (CAS) depends on a background programming language. Thus, the popular Mathematica™[3] runs on *C*, while Maple™[4] has its own programming language. LISP is the family of languages used by Maxima (the name has its origins in the acronym for *LISts Processor*); there are many flavors of LISP, such as CLISP (Common LISP), GCL (GNU Common Lisp), SBCL (Steel Bank Common Lisp), or ECL (Embeddable Common Lisp). Each one of them has its own advantages and shortcomings, of course, but this flexibility in the choice of the background has proven to be very profitable for the user. LISP is an interpreted language (although code can be compiled to gain better performance and speed), and the interpreters just mentioned are just a sampler of the wide variety of implementations that currently exist. In this note we will center our attention on ECL [2], a portable compiler with a very complete set of features. It supports many operating systems, among them AndroidOS (which usually runs on ARM processors, although it also works on Intel processors) thanks to Sylvain Ageneau who made the necessary adaptation of the original ECL code. Due to this fact, it has been possible to make a full-featured port of Maxima to the Android platform, a task performed by Yasuaki Honda. He wrote the Java interface for the code, allowing the use of nice mathematical fonts and resources of MathJax. The result is Maxima on Android, a powerful port of Maxima that runs in any smartphone (AndroidOS version 2.2 or higher is required), even those with very limited resources (as stated in the Introduction, we have used a really low-end phone to work out all the examples presented here).

The installation of the software is straightforward, as for any Android app. Within your smartphone, tap on the Play Store app, and in the window that opens, tap on the little magnifying glass icon located in the upper right corner. In the text box that appears, type 'Maxima on Android' (the Play Store app will offer you to auto-complete the text). In the list of matching apps, select it. You will be redirected to the homepage of MoA. Then, it suffices to tap on the 'install' button.

Once installed, upon opening the program the user sees a screen with the welcome message of Maxima and a prompt `(%i1)` indicating that Maxima is waiting for the user input. MoA displays a line at the bottom of the screen where this input is written, and a button with the label `Enter`. When the user has finished writing the command she wants to execute, pressing `Enter` will pass the code to Maxima and the result will appear with an output label such as `(%o1)`. The input line will remain occupied by the last written code, so it can be re-used if necessary (although this is rather unnecessary in Maxima , where you can refer to a previous output with `th(%oj)`, with `j` any number). MoA includes its own version of Gnuplot for the graphics, and almost all of the most commonly used commands and built-in functions of Maxima are available. The options menu of MoA (the three little squares in the right top of the window) includes the possibility of saving the current session or loading a previously saved one, a choice for the language of the Maxima's help manual, and a separate page for the last graph plotted.

A useful remark about the usage of MoA: it is possible to write a script containing the functions defined by the user, and to store it in the `/sdcard/Download` folder, as a single text file ending in `.txt`. Then, starting the MoA session with something like `load("myscript.txt")`,

[3] Mathematica is a trademark of Wolfram Research.

[4] Maple is a trademark of MapleSoft, a subsidiary of Cybernet Systems Group.

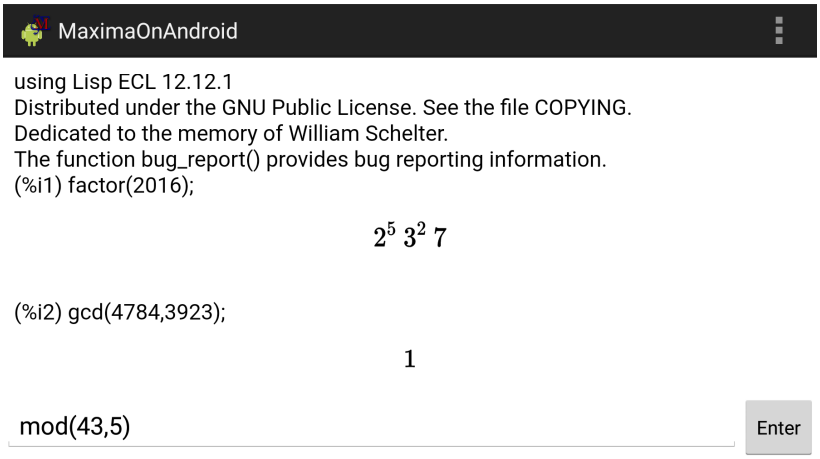


Figure 1: Some basic functions: prime factorization, greatest common divisor and a computation modulo 5

will make these functions available directly from MoA, without having to write them each time. That is (for the *connaisseur*): You can write your scripts as if they were Maxima's `.mac` files, with the only proviso that they have the `.txt` suffix, and are stored in `/sdcard/Download` (and no other place). This is the method we have followed to take the screenshots, in order to show more clearly the final results. As an example, using the standard correspondence between letters of the alphabet and numbers

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

we show in the following code two functions to encode and decode messages using the affine cipher. These functions can be written and saved in a computer or the smartphone using a text editor (with `.txt` extension, as mentioned):

```
affine_cipher(s,a,b):=block([tmp],
if not(equal(gcd(a,26),1)) then
return("Bad parameters: a is not coprime with 26")
else
(tmp:mod(a*(map(cint,charlist(s))-97)+b,26),
simplode(map(ascii,tmp+65)))
)$

affine_decipher(s,a,b):=block([tmp],
tmp:mod(inv_mod(a,26)*(map(cint,charlist(s))-65-b),26),
simplode(map(ascii,tmp+97))
)$
```

Put the resulting file (let us call it `affine.txt`) in `/sdcard/Download` and fire up MoA. At the prompt, write

```
(%i1) load("affine.txt")
```

Now you have the functions `affine_cipher` and `affine_decipher` available. The first one accepts three arguments `(s,a,b)`: the first one, `s`, must be a string in lowercase letters (strings are written in Maxima between double quotes, as in `‘‘aleajactaest’’`), the second and third ones, are the parameters of encryption, recall that in the affine cryptosystem, to each letter a number between 0 and 25 is assigned, then each one of these numbers x is transformed as $x \mapsto ax + b$, the result is computed modulo 26, and then translated again into letters (this time capital letters, as is traditional in cryptography courses). Thus, to encrypt `‘‘aleajactaest’’` in the original Caesar's cipher (consisting in shifting each letter to the third place to its right in the alphabet, modulo 26, that is, $a = 1$ and $b = 3$), one simply would do

```
(%i2) affine_cipher(‘‘aleajactaest’’,1,3)
(%o2) DOHDMDFWDHVW
```

Check that this is correct by deciphering it. Notice that you do not have to write the previous output, you can refer to it by its label (and Maxima knows that it is a string!):

```
(%i3) affine_decipher(%o2,1,3)
(%o3) aleajactaest
```

Of course, just as a one more quick check, the same result is obtained if you cipher the output `dohdmdfwdhvw` (this time in lowercase letters) with the inverse parameters $a = 1$, $b = -3$:

```
(%i4) affine_cipher(‘‘dohdmdfwdhvw’’,1,-3)
(%o4) ALEAJACTAEST
```

The procedures described in this section, from saving a Maxima script to execute it inside MoA, can be seen in action at either the URL `http://galia.fc.uaslp.mx/~jvallejo/Software.html` or `https://www.youtube.com/watch?v=QIK4XjzGRYs`.

3 Topics in number theory applied to cryptography

As stated in the introduction, a course on number theory applied to cryptography can be taught at a variety of levels, and we have chosen to maintain ourselves at the most elementary one. We hope that this choice, along with the strong use of the CAS Maxima, will achieve two goals:

1. To show, even to first-year university students, that mathematics is useful in everyday life.

2. To develop their computational skills through the use of software.

Additionally, we would hope to catch their attention and interest by showing them how they can use their smartphone in an unexpected way (by the way, this is also a good method for avoiding having them curled upon their phones for other tasks, such as messaging or checking their Facebook accounts). According to our choice, the prerequisites of the course are kept to a bare minimum: just some working knowledge of basic arithmetic and the elementary logic and set theory, methods of proof by contradiction and induction will suffice.

The contents are quite standard and the ordering follows basically that of McAndrews's book, with some exceptions. The main reason for that is that we cover much less material. We start with a review of basic arithmetic (Euclid's algorithm, gcd and lcm, prime numbers and their properties, etc.). Then we move to modular arithmetic; this topic is developed in greater detail, as it is crucial for the rest of the course. The text that we have found to be more suitable for the mathematics in this course is [5], its pace is appropriate for the students to follow it, the explanations are clearly detailed enough, and it comes with lots of worked examples (it also includes solutions to selected exercises at the end).

We think that applications should be presented as soon as possible. Right after talking about modular arithmetic, we present the first example of cryptography: the Caesar cipher. It can be implemented in less than 5 minutes in MoA (see the preceding section), and we propose our students to exchange coded messages between them, thus leading to the deciphering process, which is also implemented trivially. Also, it is very easy to write a program for brute-force cracking the messages encrypted this way, and this brings some fun to the classroom. The students learn very quickly that the more modern encrypted communication protocols used in programs such as Whatsapp or Snapchat are just refinements of this basic idea, and that their security depends crucially on their strength against brute-force attacks.
Another application of modular arithmetic –not related to cryptography– is the ISBN code. We can not help to include a cursory description of coding theory; the notions of alphabet, codewords blocks, syndromes,etc., are reviewed, and it is shown that the ISBN is a linear block code of block length 10 over $\mathbb{Z}_{11}$. We also show how it is possible, in some cases, to use the syndrome to correct errors in this code.

Returning to mathematics, we reach the core of the course: linear congruences , the Chinese remainder theorem, and the theorems of Fermat and Euler. The application in this case is the Miller-Rabin test, including a proof that the test fails in showing that a given number is composite with a probability $\frac{1}{4}$, students are asked to write their own version of the algorithm, and then to compare it with the one included in Maxima (this is the great and true advantage of using free software). It is surprising that some students can write a program which is nearly as efficient as Maximas's one!.
Another smartphone-lab in this part, consists in checking that the number $\varphi(n)/n$ (where φ is Euler's totient function) approximates the probability that a positive integer chosen at random, be coprime with $p_1, \ldots, p_k$, where $n = p_1^{e_1} \cdots p_k^{e_k}$ is the prime factorization of n. These mini-projects are used to learn some Maxima coding. In this example, one could write first a program that selects m random numbers between 1 and $100m$:

```
(%i1) numgen(m):=block([s,N],
N:100*m,
for j:1 thru m do s[j]:random(N),
makelist(s[j],j,1,m)
)$
(%i2) numgen(10);
(%o2) [612,302,734,585,204,391,429,985,298,403]
```

The syntax is self-explanatory (another great advantage of Maxima). For a list of such numbers, we can complete the fraction of those coprime with n:

```
(%i3) P(m,n):=block([tmp,N],
```

```
N:numgen(m),
for k:1 thru m do t[k]:gcd(n,N[k]),
tmp:makelist(t[k],k,1,m),
length(sublist_indices(tmp,lambda([x],x='1)))/m
)$
```

Notice the use of lambda functions here. It is hard to overemphasize that using a CAS-oriented approach (particularly one based on functional programming, such as Maxima), reinforces the analytic skills of students. Finally, we can check that $P(m,n)$, for large values of m, approaches $\varphi(n)/n$:

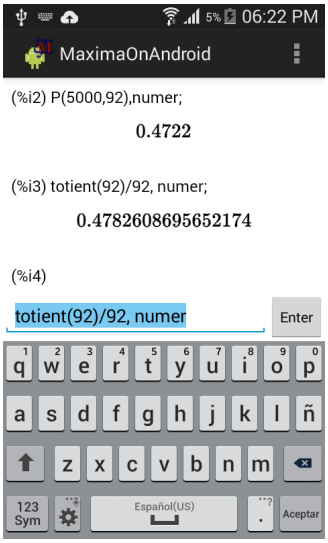


Figure 2: The totient function

```
(%i4) P(5000,92),numer;
(%o4) 0.4718
(%i5) totient(92)/92,numer;
(%o5) 0.4782608695652174
```

At this point, we introduce the main examples of classical cryptography: the affine, Vigènere and Hill ciphers. Next, we study the attacks on these systems, mainly based on n-grams frequencies. We closely follow McAndrew here, but the case of the Hill cipher, we present a scheme of ciphertext-only attack. As we think it is of some interest, and not so easily found in the literature, we discuss this case in detail in section 5.

When studying the cryptanalysis of the Vigènere cipher, we introduce the basics of Shannon's information theory, using the entropy of the encrypted message as a method for computing the key length alternative to that of Kasiski.

The last part of the course deals with modern cryptography, based on public keys. As a warm-up, we review some properties of primitive roots modulus a prime and the Diffie-Hellman-Merkle protocol for exchanging keys, as well as quadratic residues in $\mathbb{Z}_p$ and the Rabin pseudo-cryptosystem. The final topics are the usual ones: El Gamal and RSA cryptosystems, and digital signatures.

4 Encryption with Elgamal

In this section we show a full implementation in Maxima of the Elgamal cryptosystem, as taught in the course. It is a good example of the combination of a usable system, available technology (a smartphone, a tablet, a personal computer), and elementary mathematics. Also, it provides some insight on the inner workings of Maxima, as we will see.

The basic mathematical notion involved in Elgamal cryptosystem is that of a primitive root in $\mathbb{Z}_p$, where p is a prime number (so $\mathbb{Z}_p$ is a field). Maxima has the command `zn_primroot_p(x,p)`, which checks whether x is a primitive root modulo p or not:

```
(%i1) zn_primroot_p(6,11);
(%o1) true
(%i2) zn_primroot_p(4,11);
(%o2) false
```

Of course, this command can be applied to compute all the primitive roots in $\mathbb{Z}_p$:

```
(%i3) zp_primroot_all(p):=sublist(makelist(j,j,1,p-1),
                                  lambda([x],zn_primroot_p(x,p)))$
(%i4) zp_primroot_all(29);
(%o4) [2,3,8,10,11,14,15,18,19,21,26,27]
```

In the Elgamal cryptosystem, a large prime p is chosen, along a primitive root r in $\mathbb{Z}_p$. For values of p of order of magnitude 10^3, these roots can b computed in a reasonable time, but for larger values, the direct use of `zn_primroot_p(x,p)` is unfeasible. The Maxima manual offers a workaround in this case, by first factoring the Euler totient function. Thus, if we were to work with the prime $p = 2^{142} + 217$, we would do first

```
(%i5) ifs:ifactors(totient(2^142+217));
(%o5) [[2,3],[5,1],[11,1],[1301,1],[14155556315601773,1],
       [688022228164201531661,1]]
```

Now, we can easily select those primitive roots comprised between a random number m (such that $m + 100 < p$) and p, and choose one of them r, say, the first one[5]:

```
(%i6) m:random(2^142+117);
(%o6) 486710190544315947409260670147655123607174 4
(%i7) sublist(makelist(i,i,m,m+100),lambda([x],zn_primroot_p(x,2^142+217,ifs)));
(%o7) [486710190544315947409260670147655123607174 5,
       ...,
       486710190544315947409260670147655123607184 4]
(%i8) first(%);
(%o8) 486710190544315947409260670147655123607174 5
```

The system's administrator (sysadmin) makes publicly available the pair (p, r). Now, Alice must create a public key A from (p, r) and a random number $N < p$ chosen by her (her private

[5]We have slightly edited the output in order to make it more readable, so we only show the first and the last elements of the list. The total number of elements is 30.

key), through the formula $A = r^N \bmod p$. If Bob wants to send a message to Alice, encoded in a number M, he must choose first a random $k < p$, then compute $K = A^k \bmod p$, and the pair of values $M_1 = r^k \bmod p$, $M_2 = KM \bmod p$. He finally sends the pair (M_1, M_2) to Alice, who, to decipher it, must recover K as the first step. This is done (modulo p) as

$$K = A^k = (r^N)^k = (r^k)^N = M_1^N \, .$$

Notice that this does not pose any problem to Alice, who knows the private key N. The original message can be recovered now as $M = K^{-1} M_2$ (operations modulo p). We now describe some Maxima functions which implement these computations quite efficiently. We will need some functions to code plaintext into numbers:

```
(%i9) text_to_numbers(s):=map(cint,charlist(s))-96$
(%i10) base27(list):=block(
                sum(reverse(list)[j]*27^(j-1),j,1,length(list))
)$
```

In general, we could write a function such as

```
(%i11) dec2base(n,b):=block([remainders,tmp],
                remainders:[n],
                do (tmp:rest(remainders,1),
                    remainders:flatten(push(divide(remainders[1],b),tmp)),
                    if is(first(remainders)<b) then return (remainders)
                   )
)$
```

Next, a function for the sysadmin (it generates the parameters (p, r) of the cryptosystem):

```
(%i12) load(distrib)$
(%i13) Elgamal_parameters(n):=block([p,tmp,roots],
         p:next_prime(floor(random_normal(2^n+12345,100,100)[random(100)])),
         tmp:ifactors(totient(p)),
         m:random(p-100),
         roots:sublist(makelist(i,i,m,m+99),lambda([x],zn_primroot_p(x,p,tmp))),
         [p,first(roots)]
       )$
(%i14) parameters:Elgamal_parameters(150);
(%o14) [142724769270595988105828596944949513638274677 1,
139409600337411403995567793142155623799003228 3]
(%i15) p:first(parameters);
(%o15) 142724769270595988105828596944949513638274677 1
(%i16) r:second(parameters);
(%o16) 139409600337411403995567793142155623799003228 3
```

Notice that we have chosen relatively small numbers in order to respect the LaTeX format of the document. There is no problem in choosing, say, $n = 1000$, the time invested in the computations in this case is about 1 second in a Pentium i5, and the numbers (p, r) will have about 300 digits each. Now, Alice can generate her private key as (see Figure 3 (a)):

```
(%i17) N:random(p);
(%o17) 89236024494349426052663903535322024328283031 5
```

Public keys can be generated with the following function,

```
(%i18) Elgamal_publickey(p,r,N):=power_mod(r,N,p)$
(%i19) A:Elgamal_publickey(p,r,N);
(%o19) 485887141515879278922730857548485384618668787
```

The function used by the sender (Bob) to encrypt a message given by a string `s`, with the public key A, and the parameters (p, r) (the function also asks for Bob's private key `k`), see Figure 3 (b):

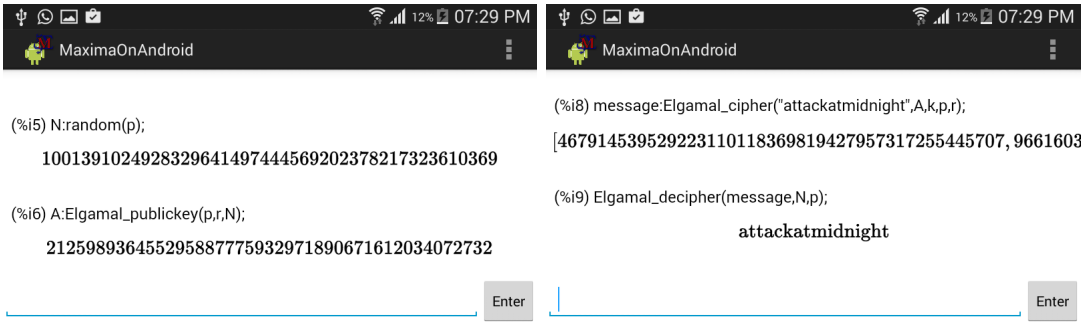
```
(%i20) Elgamal_cipher(s,A,k,p,r):=block([M,K],
                                        M:base27(text_to_numbers(s)),
                                        K:power_mod(A,k,p),
                                        M1:power_mod(r,k,p),
                                        M2:mod(K*M,p),
                                        [M1,M2]
                                       )$
(%i21) k:random(p);
(%o21) 272257742710230641981665814872866632782818419
(%i22) messsage:Elgamal_cipher("attackatmidnight",A,k,p,r);
(%o22) [127076134428212954149025970628748817641787157 2,
        123876863973837866324190937736553312627592662 5]
```

Now, the deciphering process. The following function accepts as its arguments a list $[M_1, M_2]$, the private key N, and the parameter p:

```
(%i23) Elgamal_decipher(M,N,p):=block([K,tmp],
                                      K:power_mod(M[1],N,p),
                                      tmp:power_mod(power_mod(K,-1,p)*M[2],1,p),
                                      simplode(map(ascii,dec2base(tmp,27)+96))
 )$
(%i24) Elgamal_decipher(message,N,p);
(%o24) "attackatmidnight"
```

5 Ciphertext-only attack on Hill's cipher

Hill's cipher, based on the use of matrices, is one of the first examples of a cryptosystem based on non-elementary mathematics. Indeed, it is one of the most suitable cryptosystems to be taught to students of mathematics and computer sciences in first courses, see [10]. It is very easy to implement, illustrates well the basic operations of matrix theory and modular arithmetic, and has several good cryptographic properties. Among them, perhaps the most important is that it diffuses the distribution of letters in the ciphertext, so it is very robust against attacks based on frequency analysis. Indeed, it is usual to consider that a ciphertext-only attack on Hill's cipher is difficult.



(a) (b)

Figure 3: Elgamal cryptosystem

The encryption process starts with a $k \times k$ matrix K. We will take here $k = 2$ for simplicity. Then, the cleartext is divided into blocks of 2 letters each, translated to pairs of numbers with the aid of the usual dictionary (see page 4). Acting on these pairs with K, through matrix multiplication, gives another pair of numbers which is translated back to letters. As an example, if

$$K = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix},$$

and the text to be encrypted is 'cats', we first write it as $2\,0\,|\,19\,18$. To encode the blocks we compute

$$\begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix} \text{ and } \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix} \begin{pmatrix} 19 \\ 18 \end{pmatrix} = \begin{pmatrix} 56 \\ 130 \end{pmatrix} \equiv \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

(in the last step, we have reduced the numbers modulo 26). By looking in the dictionary, we see that these numbers correspond to the ciphertext 'EIEA'. To decode, the process is the same, this time using K^{-1} instead of K.

Here is a function to encrypt a text string `s` with Hill's cipher, using an encryption matrix `K` (it includes some additional features, such as padding when the length of the cleartext is not a multiple of 2):

```
Hill_cipher(s,K):=block([tmp1,tmp2,tmp3,tmp4,tmp5,L1,L2,m],
m:length(K),
if is(equal(second(divide(slength(s),m)),0)) then
    (
    tmp4:map(cint,charlist(s))-97,
    L2:length(tmp4),
    tmp5:makelist(create_list(tmp4[j],j,(i-1)*m+1,m*i),i,1,L2/m),
    simplode(map(ascii,
     mod(flatten(
      makelist(list_matrix_entries(tmp5[j].transpose(K)),j,1,length(tmp5)))
     ,26)
    +65))
    )
else
```

```
    (
    tmp1:smake(m-second(divide(slength(s),m)),ascii(120)),
    tmp2:map(cint,charlist(concat(s,tmp1)))-97,
    L1:length(tmp2),
    tmp3:makelist(create_list(tmp2[j],j,(i-1)*m+1,m*i),i,1,L1/m),
    simplode(map(ascii,
     mod(flatten(
      makelist(list_matrix_entries(tmp3[j].transpose(K)),j,1,length(tmp3)))
     ,26)
    +65))
    )
)$
```

The function at work (see also the screenshot in Figure 4):

```
(%i1) K:matrix([2,1],[4,3]);
(%o1) matrix([2,1],[4,3])
(%i2) cleartext:"cats";
(%o2) "cats"
(%i3) Hill_cipher(cleartext,K);
(%o3) "EIEA"
```

Here we will see a plausible ciphertext-only attack on a Hill's cipher of block size 2, using the $n-$grams frequency analysis as a tool. This type of attack is very easily implemented on Maxima, but it is not fully automatic, it needs a good deal of intervention and educated guesses from the user, so it is valuable from a pedagogical point of view.

In what follows we will consider only messages containing the English alphabet letters with the usual correspondence between letters and classes in $\mathbb{Z}_{26}$ (see page 4). Ciphertexts will be written in capital letters.

Consider the following ciphertext, encrypted using a 2×2 matrix by the Hill method from a plain text written in Spanish:

$$NPYLUGEBILECNRJLTIECYGVBNHDLTTDQ$$

Our task is to decipher it, and for this we need to find the encryption matrix

$$K = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix}.$$

The idea is the following. First, we will traverse the text, analyzing blocks of four letters in each step and comparing them with the most usual $4-$grams in Spanish, which are (in descending order of appearance frequency): *cion*, *dela*, *acio*. For example, we would start with

$$\begin{array}{cc|cc} c & i & o & n \\ N & P & Y & L \end{array}$$

If the correspondence would be an exact one, that is, if $NPYL$ were the encryption of *cion*, then we would have

$$\begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} c \\ i \end{pmatrix} = \begin{pmatrix} N \\ P \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} o \\ n \end{pmatrix} = \begin{pmatrix} Y \\ L \end{pmatrix}$$

This translates into the system of equations modulo 26

$$\begin{cases} 2\mathbf{a}+8\mathbf{b}=13 \\ 2\mathbf{c}+8\mathbf{d}=15 \\ 14\mathbf{a}+13\mathbf{b}=24 \\ 14\mathbf{c}+13\mathbf{d}=11 \end{cases},$$

which has no solutions. Next we move on to the next block of four letters:

$$\begin{array}{c|c|c} c & i\ \ o & n \\ NP & YL & UG \end{array}$$

The first and last two-letter blocks are incomplete, we complete them using bi-grams: For the left block, we look for the most frequent bi-gram in Spanish having c as its second letter, which is ac. Then we would be lead to consider

$$\begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix}\begin{pmatrix} a \\ c \end{pmatrix}=\begin{pmatrix} N \\ P \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix}\begin{pmatrix} i \\ o \end{pmatrix}=\begin{pmatrix} Y \\ L \end{pmatrix}$$

or, equivalently,

$$\begin{cases} 2\mathbf{b}=13 \\ 2\mathbf{d}=15 \\ 8\mathbf{a}+14\mathbf{b}=24 \\ 8\mathbf{c}+14\mathbf{d}=11 \end{cases},$$

which neither has solution. We would pass to consider the right two-letter block, using the most frequent bi-gram having n as its first letter, which is nt, constructing the equivalent system of linear equations in $\mathbb{Z}_{26}$ and analyzing its solutions. If we do it, we will not find any solution. The next step is to consider the *second* most frequent bi-grams for both, the left and the right blocks of two letters. In case we do not find solutions (and that is exactly what happens in this example) we would try with the *third* most frequent bi-grams. But if these do not give a solution, we would move on to the next block of four letters.
Notice that we could obtain a solution to the system of linear congruences, but one leading to a non-sense message. For instance, moving the $4-$gram four positions, we would get

$$\begin{array}{c|c} i\ \ o & n \\ UG & EB \end{array}$$

Completing the right block with nt (the most frequent bi-gram starting with n), we get

$$\begin{cases} 8a+14b=20 \\ 8c+14d=6 \\ 13a+19b=4 \\ 13c+19d=1 \end{cases},$$

which has a solution $\mathbf{a}=10$, $\mathbf{b}=18$, $\mathbf{c}=1$, $\mathbf{d}=24$. Using the corresponding matrix K to decipher the message we get

$$gocqqcamowqgmckuasqgaukeikiguekc$$

a text without meaning (even in Spanish!).

Of course, it may happen that after traversing the whole ciphertext, we still do not get a solution. Then it is time to try with the *second* most frequent 4−gram, repeating the process just described. In this case, we would try with *dela*. After repeating the process four times, we arrive at

$$\begin{array}{c|c|c} d & e\ \ l & a \\ EB & I\ \ L & EC \end{array}$$

Completing the left block of two letters, we use *sd*, because this is the most frequent bi-gram with d in its second place. The corresponding system is

$$\begin{cases} 18a + 3b = 4 \\ 18c + 3d = 1 \\ 4a + 11b = 8 \\ 4c + 11d = 11 \end{cases}, \tag{1}$$

which has the solution $\mathbf{a} = 5$, $\mathbf{b} = 6$, $\mathbf{c} = 1$, $\mathbf{d} = 3$. Using the corresponding matrix

$$K = \begin{pmatrix} 5 & 6 \\ 1 & 3 \end{pmatrix},$$

we get the message

$$despuesdelastiniblasesperolaluz$$

('After the dark, I wait for the light', it is a *motto* written in the cover of the first edition of 'Don Quixote', by M. de Cervantes (1616)).

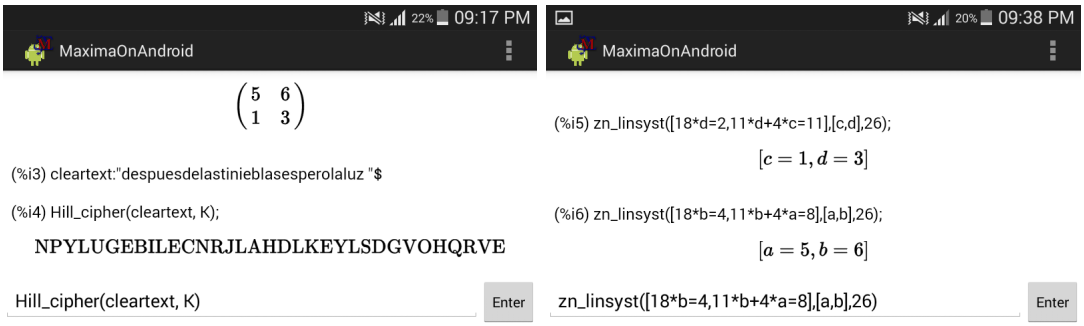
Here is a program for solving systems of 2 linear congruences, by Gaussian elimination (it is very specific to the 2×2 case, using arithmetic modulo 2, but it is given here to show how to work in $\mathbb{Z}_2$):

```
zn_linsyst(listec,listvar,n):=
    block([c,tmp,divs,L,i0,j0,cinv,A,B,a,b,d,q,mprime,u,x,y],
    c:coefmatrix(listec,listvar),
    tmp:list_matrix_entries(c),
    for i:1 thru 4 do divs[i]:gcd(tmp[i],n),
    L:makelist(divs[i],i,1,4),
    if not(member(1,L)) then
      return(
      "There are no solutions, no coefficient is coprime with the modulus"
      )
    else
        (
         if is(equal(L[1],1)) then (i0:0,j0:0),
         if is(equal(L[2],1)) then (i0:0,j0:1),
         if is(equal(L[3],1)) then (i0:1,j0:0),
         if is(equal(L[4],1)) then (i0:1,j0:1),
         cinv:inv_mod(c[i0+1,j0+1],n),
         A:c[mod(i0+1,2)+1,mod(j0+1,2)+1]-
           cinv*c[mod(i0+1,2)+1,j0+1]*c[i0+1,mod(j0+1,2)+1],
```

```
        B:rhs(listec[mod(i0+1,2)+1])-
          inv_mod(c[i0+1,j0+1],n)*rhs(listec[i0+1])*c[mod(i0+1,2)+1,j0+1],
        a:mod(A,n),
        b:mod(B,n),
        d:gcd(a,n),
        if elementp(d,divisors(b))
          then (
                  q:b/d,
                  mprime:n/d,
                  u:first(gcdex(a,n)),
                  x:mod(mod(q*u,n),mprime)
                )
      else
      return(
      "There are no solutions: in ax=b mod n, gcd(a,n) does not divide b"
      ),
   [
   listvar[mod(j0+1,2)+1] =x,
   listvar[j0+1]=mod(cinv*(rhs(listec[i0+1])-c[i0+1,mod(j0+1,2)+1]*x),n)
   ]
    )
)$
```

Some examples (the solution to (1)):

```
(%i4) zn_linsyst([18*d=2,11*d+4*c=11],[c,d],26);
(%o4) [c=1,d=3]
(%i5) zn_linsyst([18*b=4,11*b+4*a=8],[a,b],26);
(%o5) [a=5,b=6]
```



(a) (b)

Figure 4: Hill cryptosystem

Finally, we list a function to compute the inverse of a matrix A modulo n, to be used in the deciphering process:

```
matrix_mod_inv(A,n):=block([det,mdet],
det:determinant(A),
    if equal(gcd(det,n),1) then
       (mdet:inv_mod(det,n),
       mod(mdet*adjoint(A),n))
    else
       return("The matrix is not invertible mod n")
)$
```

6 Conclusions

Of course, our conclusions are completely biased, but we are firmly convinced that smartphones and tablets can be efficiently used in the classroom, even at the level of university courses. Among the many advantages we have found, we can count the following:

1. Use can be made of modern technology without the need for an internet connection, only local WIFI or, in the worst scenario, a mobile phone network is needed.

2. To avoid the use of cell phones in the classroom for other purposes than learning.

3. To catch students' interest, they are prone to put attention on those topics they consider 'hot', and nothing seems hotter to post-teenagers than using their smartphone in an advanced way.

4. No need for a computer lab, any room will do the job. The tables will not be clustered with laptops, cables, chargers and so on.

5. The teacher can feel again what is like to be a teenager. Just create a group in your favorite social network to send their homework and some challenges in the form of encrypted messages, wait a couple of minutes and be prepared to learn the exciting *Neolingua* of IM clients :).

Regarding the technical contents, number theory is an ideal course for introducing the student to the basics of functional programming, developing her analytic skills and learning some useful mathematics, used in everyday life. The availability of both, great textbooks and software, allows the teacher to develop a variety of approaches to this topic, easily adaptable to the pedagogical trend of the moment.

Acknowledgement. This work has been supported by a Mexican National Council of Science and Technology Research Project, code CB-2012-179115. It is partly based on the second's author B. Sc. Thesis.

References

[1] *Frontiers in Number Theory, Physics, and Geometry I.* Editors: P. Cartier et al. Springer Verlag, 2006.

[2] Embeddable Common-Lisp: https://common-lisp.net/project/ecl/index.html.

[3] G. H. Hardy: *A Mathematician's Apology*, Cambridge University Press, 1940. There is a freely available edition at the URL https://www.math.ualberta.ca/mss/misc/AMathematician'sApology.pdf, the quotation here can be found in page 44.

[4] Y. Honda: *Maxima on Android* (Version 2.8), 2015, https://play.google.com/store/apps/details?id=jp.yhonda&feature=search_result#?t=W251bGwsMSwxLDEsImpwLnlob25kYSJd.

[5] G. A. Jones and J. Mary Jones: *Elementary number theory*, Springer Verlag, 1998.

[6] N. Koblitz: *A course on Number Theory and Cryptography*. Springer Verlag, 1994.

[7] J. S. Kraft, L. C. Washington: *An Introduction to Number Theory with Cryptography*. CRC Press, 2013.

[8] M. Marcolli: *Number theory in Physics*. In Encyclopaedia of Mathematical Physics, Elsevier, 2006. Editors: J. P. Francoise, G. L. Naber, and T. S. Tsun.

[9] Maxima.sourceforge.net: *Maxima, a Computer Algebra System*. Version 5.38.0 (2016). http://maxima.sourceforge.net.

[10] A. McAndrew: *Using the Hill cipher to teach cryptographic principles*. International Journal of Mathematical Education in Science and Technology **39** 7 (2008) 967–969

[11] A. McAndrew: *Introduction to cryptography with open-source software*, CRC Press, 2011.

[12] The Sage Developers: *SageMath, the Sage Mathematics Software System* (Version 7.3), 2016, http://www.sagemath.org.

[13] The Sage Developers: *SageMath for Android* (Version 1.0.1), 2014, https://play.google.com/store/apps/details?id=org.sagemath.droid.

[14] M. A. Skillen: *Mobile Learning: Impacts on Mathematics Education*, Proc. of the XX Asian Technology Conference in Mathematics (Leshan, China), 2015. Mathematics and Technology, LLC.

[15] D. R. Stinson: *Cryptography: Theory and practice*, CRC Press, 1995.

[16] W. S. von Waltershausen: *Gauss zum Gedchtniss*, S. Hirzel, Leipzig, 1856. A freely available edition can be found at the URL https://archive.org/details/gauss00waltgoog. The quotation mentioned in the text is in page 64 of this edition.

[17] S. Y. Yang: *Number Theory for Computing*, Springer Verlag, 2002.