Appendix A Mathematica® Examples

These computer examples are written in Mathematica. If you have Mathematica available, you should try some of them on your computer. If Mathematica is not available, it is still possible to read the examples. They provide examples for several of the concepts of this book. For information on getting started with Mathematica, see Section A.1. To download a Mathematica notebook that contains these commands, go to

```
bit.ly/2u5R7dW
```

A.1 Getting Started with Mathematica

1. Download the Mathematica notebook crypto.nb that you find using the links starting at `bit.ly/2u5R7dW`

2. Open Mathematica, and then open crypto.nb using the menu options under File on the command bar at the top of the Mathematica window. (Perhaps this is done automatically when you download it; it depends on your computer settings.)

3. With crypto.nb in the foreground, click (left button) on Evaluation on the command bar. A menu will appear. Move the arrow down to the line Evaluate Notebook and click (left button). This evaluates the notebook and loads the necessary functions. Ignore any warning messages about spelling. They occur because a few functions have similar names.

4. Go to the command bar at the top and click on File. Move the arrow down to New and left click. Then left click on Notebook. A new notebook will appear on top of crypto.nb. However, all the commands of crypto.nb will still be working.

5. If you want to give the new notebook a name, use the File command and scroll down to Save As.... Then save under some name with a .nb at the end.

6. You are now ready to use Mathematica. If you want to try something easy, type $1 + 2*3 + 4\hat{}5$ and then press the Shift and Enter keys simultaneously. Or, if your keyboard has a number pad with Enter, probably on the right side of the keyboard, you can press that (without the Shift). The result 1031 should appear (it's $1 + 2 \cdot 3 + 4^5$).

7. Turn to the Computer Examples Section A.3. Try typing in some of the commands there. The outputs should be the same as that in the examples. Remember to press Shift Enter (or the numeric Enter) to make Mathematica evaluate an expression.

8. If you want to delete part of your notebook, simply move the arrow to the line at the right edge of the window and click the left button. The highlighted part can be deleted by clicking on Edit on the top command bar, then clicking on Cut on the menu that appears.

9. Save your notebook by clicking on File on the command bar, then clicking on Save on the menu that appears.

10. Print your notebook by clicking on File on the command bar, then clicking on Print on the menu that appears. (You will see the advantage of opening a new notebook in Step 4; if you didn't open one, then all the commands in crypto.nb will also be printed.)

11. If you make a mistake in typing in a command and get an error message, you can edit the command and hit Shift Enter to try again. You don't need to retype everything.

12. Look at the commands available through the command bar at the top. For example, Format then Style allows you to change the type font on any cell that has been highlighted (by clicking on its bar on the right side).

13. If you are looking for help or a command to do something, try the Help command. Note that the commands that are built into Mathematica always start with capital letters. The commands that are coming from crypto.nb start with small letters and will not be found via Help.

A.2 Some Commands

The following are some Mathematica commands that are used in the Computer Examples. The commands that start with capital letters, such as `EulerPhi`, are built into Mathematica. The ones that start with small letters, such as `addell`, have been written specially for this text and are in the Mathematica notebook available at

```
bit.ly/2u5R7dW
```

`addell[{x,y}, {u,v}, b, c, n]` finds the sum of the points $\{x, y\}$ and $\{u, v\}$ on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod{n}$, where n is odd.

`affinecrypt[txt,m,n]` affine encryption of txt using $mx + n$.

`allshifts[txt]` gives all 26 shifts of txt.

`ChineseRemainder[{a,b,...},{m,n,...}]` gives a solution to the simultaneous congruences $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$, ...

`choose[txt,m,n]` lists the characters in txt in positions congruent to $n \pmod{m}$.

`coinc[txt,n]` the number of matches between txt and txt displaced by n.

`corr[v]` the dot product of the vector v with the 26 shifts of the alphabet frequency vector.

`EulerPhi[n]` computes $\phi(n)$ (don't try very large values of n).

`ExtendedGCD[m,n]` computes the gcd of m and n along with a solution of $mx + ny = \gcd$.

`FactorInteger[n]` factors n.

`frequency[txt]` lists the number of occurrences of each letter a through z in txt.

`GCD[m,n]` is the gcd of m and n.

`Inverse[M]` finds the inverse of the matrix M.

`lfsr[c,k,n]` gives the sequence of n bits produced by the recurrence that has coefficients given by the vector c. The initial values of the bits are given by the vector k.

`lfsrlength[v,n]` tests the vector v of bits to see if it is generated by a recurrence of length at most n.

`lfsrsolve[v,n]` given a guess n for the length of the recurrence that generates the binary vector v, it computes the coefficients of the recurrence.

`Max[v]` is the largest element of the vector v.

`Mod[a,n]` is the value of $a \pmod n$.

`multell[{x,y}, m, b, c, n]` computes m times the point $\{x, y\}$ on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod n$.

`multsell[{x,y}, m, b, c, n]` lists the first m multiples of the point $\{x, y\}$ on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod n$.

`NextPrime[x]` gives the $\text{next prime} > x$.

`num2text0[n]` changes a number n to letters. The successive pairs of digits must each be at most 25; a is 00, z is 25.

`num2text[n]` changes a number n to letters. The successive pairs of digits must each be at most 26; *space* is 00, a is 01, z is 26.

`PowerMod[a,b,n]` computes $a^b \pmod{n}$.

`PrimitiveRoot[p]` finds a primitive root for the prime p.

`shift[txt,n]` shifts txt by n.

`txt2num0[txt]` changes txt to numbers, with $a = 00, \ \ldots, z = 25$.

`txt2num[txt]` changes txt to numbers, with *space* = 00, $a = 01, \ \ldots, z = 26$.

`vigenere[txt,v]` gives the Vigenère encryption of txt using the vector v.

`vigvec[txt,m,n]` gives the frequencies of the letters a through z in positions congruent to $n \pmod{m}$.

A.3 Examples for Chapter 2

Example 1

A shift cipher was used to obtain the ciphertext `kddmu`. Decrypt it by trying all possibilities.

In[1]:= `allshifts["kddkmu"]`

```
kddkmu
leelnv
mffmow
nggnpx
ohhoqy
piiprz
qjjqsa
rkkrtb
sllsuc
tmmtvd
unnuwe
voovxf
wppwyg
xqqxzh
yrryai
zsszbj
attack
buubdl
cvvcem
dwwdfn
exxego
fyyfhp
gzzgiq
haahjr
ibbiks
jccjlt
```

As you can see, *attack* is the only word that occurs on this list, so that was the plaintext.

Example 2

Encrypt the plaintext message *cleopatra* using the affine function $7x + 8$:

```
In[2]:=affinecrypt["cleopatra", 7, 8]

Out[2]=whkcjilxi
```

Example 3

The ciphertext *mzdvezc* was encrypted using the affine function $5x + 12$. Decrypt it.

SOLUTION

First, solve $y \equiv 5x + 12 \pmod{26}$ for x to obtain $x \equiv 5^{-1}(y - 12)$. We need to find the inverse of $5 \pmod{26}$:

```
In[3]:= PowerMod[5, -1, 26]

Out[3]= 21
```

Therefore, $x \equiv 21(y - 12) \equiv 21y - 12 \cdot 21$. To change $-12 \cdot 21$ to standard form:

```
In[4]:= Mod[-12*21, 26]

Out[4]= 8
```

Therefore, the decryption function is $x \equiv 21y + 8$. To decrypt the message:

```
In[5]:= affinecrypt["mzdvezc", 21, 8]

Out[5]= anthony
```

In case you were wondering, the plaintext was encrypted as follows:

In[6]:= `affinecrypt["anthony", 5, 12]`

Out[6]= `mzdvezc`

Example 4

Here is the example of a Vigenère cipher from the text. Let's see how to produce the data that was used in Section 2.3 to decrypt it. For convenience, we've already stored the ciphertext under the name *vvhq*.

In[7]:= `vvhq`

Out[7]=

```
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

Find the frequencies of the letters in the ciphertext:

In[8]:= `frequency[vvhq]`

Out[8]=

```
{{a, 8}, {b, 5}, {c, 12}, {d, 4}, {e, 15}, {f,
10}, {g, 27}, {h, 16}, {i, 13}, {j, 14}, {k, 17},
{l, 25}, {m, 7}, {n, 7}, {o, 5}, {p, 9}, {q, 14},
{r, 17}, {s, 24}, {t, 8}, {u, 12}, {v, 22}, {w,
22}, {x, 5}, {y, 8}, {z, 5}}
```

Let's compute the coincidences for displacements of 1, 2, 3, 4, 5, 6:

In[9]:= `coinc[vvhq, 1]`

Out[9]= 14

In[10]:= `coinc[vvhq, 2]`

Out[10]= 14

In[11]:= `coinc[vvhq, 3]`

Out[11]= 16

In[12]:= `coinc[vvhq, 4]`

Out[12]= 14

In[13]:= `coinc[vvhq, 5]`

Out[13]= 24

In[14]:= `coinc[vvhq, 6]`

Out[14]= 12

We conclude that the key length is probably 5. Let's look at the 1st, 6th, 11th, ... letters (namely, the letters in positions congruent to 1 mod 5):

In[15]:= `choose[vvhq, 5, 1]`

Out[15]=
`vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqu`
`dukvpkvggjjivgjggpfncwuce`

In[16]:= `frequency[%]`

Out[16]= {{a, 0}, {b, 0}, {c, 7}, {d, 1},
{e, 1}, {f, 2}, {g, 9}, {h, 0}, {i, 1},
{j, 8}, {k, 8}, {l, 0}, {m, 0}, {n, 3},
{o, 0}, {p, 4}, {q, 5}, {r, 2}, {s, 0},
{t, 3}, {u, 6}, {v, 5}, {w, 1}, {x, 0},
{y, 1}, {z, 0}}

To express this as a vector of frequencies:

In[17]:= vigvec[vvhq, 5, 1]

Out[17]= {0, 0, 0.104478, 0.0149254,
0.0149254, 0.0298507, 0.134328, 0,
0.0149254, 0.119403, 0.119403, 0, 0,
0.0447761, 0, 0.0597015, 0.0746269,
0.0298507, 0, 0.0447761, 0.0895522,
0.0746269, 0.0149254, 0, 0.0149254, 0}

The dot products of this vector with the displacements of the alphabet frequency vector are computed as follows:

In[18]:= corr[%]

Out[18]=

```
{0.0250149, 0.0391045, 0.0713284, 0.0388209,
0.0274925, 0.0380149, 0.051209, 0.0301493,
0.0324776, 0.0430299, 0.0337761, 0.0298507,
0.0342687, 0.0445672, 0.0355522, 0.0402239,
0.0434328, 0.0501791, 0.0391791, 0.0295821,
0.0326269, 0.0391791, 0.0365522, 0.0316119,
0.0488358, 0.0349403}
```

The third entry is the maximum, but sometimes the largest entry is hard to locate. One way to find it is

In[19]:= Max[%]

Out[19]= 0.0713284

Now it is easy to look through the list and find this number (it usually occurs only once). Since it occurs in the third position, the first shift for this Vigenère cipher is by 2, corresponding to the letter *c*. A procedure similar to the one just used (using *vigvec[vvhq, 5,2],..., vigvec[vvhq,5,5]*) shows that the other shifts are probably 14, 3, 4, 18. Let's check that we have the correct key by decrypting.

In[20]:= `vigenere[vvhq, -{2, 14, 3, 4, 18}]`

Out[20]=

```
themethodusedforthepreparationandreadingofcodemes
sagesissimpleinthe
extremeandatthesametimeimpossibleoftranslationunl
essthekeyisknownth
eeasewithwhichthekeymaybechangedisanotherpointinf
avoroftheadoptiono
fthiscodebythosedesiringtotransmitimportantmessag
eswithouttheslight
estdangeroftheirmessagesbeingreadbypoliticalorbus
inessrivalsetc
```

For the record, the plaintext was originally encrypted by the command

In[21]:= `vigenere[%, {2, 14, 3, 4, 18}]`

Out[21]=

```
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

A.4 Examples for Chapter 3

Example 5

Find $\gcd\,(23456, 987654)$.

In[1]:= `GCD[23456, 987654]`

Out[1]= 2

Example 6

Solve $23456x + 987654y = 2$ in integers x, y.

In[2]:= `ExtendedGCD[23456, 987654]`

Out[2]= `{2, {-3158, 75}}`

This means that 2 is the gcd and
$23456 \cdot (-3158) + 987654 \cdot 75 = 2$.

Example 7

Compute $234 \cdot 456 \pmod{789}$.

In[3]:= `Mod[234*456, 789]`

Out[3]= 189

Example 8

Compute $234567^{876543} \pmod{565656565}$.

In[4]:= `PowerMod[234567, 876543, 565656565]`

Out[4]= 473011223

Example 9

Find the multiplicative inverse of
$87878787 \pmod{9191919191}$.

In[5]:= `PowerMod[87878787, -1, 9191919191]`

Out[5]= `7079995354`

Example 10

Solve $7654x \equiv 2389 \pmod{65537}$.

SOLUTION

Here is one way. It corresponds to the method in Section 3.3. We calculate 7654^{-1} and then multiply it by 2389:

In[6]:= `PowerMod[7654, -1, 65537]`

Out[6]= `54637`

In[7]:= `Mod[%*2389, 65537]`

Out[7]= `43626`

Example 11

Find x with

$$x \equiv 2 \pmod{78}, x \equiv 5 \pmod{97}, x \equiv 1 \pmod{119}.$$

SOLUTION

In[8]:= `ChineseRemainder[{2, 5, 1}, {78, 97, 119}]`

Out[8]= `647480`

We can check the answer:

In[9]:= `Mod[647480, {78, 97, 119}]`

Out[9]= `{2, 5, 1}`

Example 12

Factor 123450 into primes.

In[10]:= `FactorInteger[123450]`

Out[10]= `{{2, 1}, {3, 1}, {5, 2}, {823, 1}}`

This means that $123450 = 2^1 3^1 5^2 823^1$.

Example 13

Evaluate $\phi(12345)$.

In[11]:= `EulerPhi[12345]`

Out[11]= `6576`

Example 14

Find a primitive root for the prime 65537.

In[12]:= `PrimitiveRoot[65537]`

Out[12]= `3`

Therefore, 3 is a primitive root for 65537.

Example 15

Find the inverse of the matrix

$$\begin{pmatrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{pmatrix} \pmod{999}.$$

SOLUTION

First, invert the matrix without the mod:

In[13]:= `Inverse[{{13, 12, 35}, {41, 53,`
`62}, {71, 68, 10}}]`

Out[13]=

$$\left\{\left\{\frac{3686}{34139}, -\frac{2260}{34139}, \frac{1111}{34139}\right\}, \left\{-\frac{3992}{34139}, \frac{2355}{34139}, -\frac{629}{34139}\right\}, \left\{\frac{975}{34139}, \frac{32}{34139}, -\frac{197}{34139}\right\}\right\}$$

We need to clear the 34139 out of the denominator, so we evaluate 1/34139 mod 999:

```
In[14]:= PowerMod[34139, -1, 999]

Out[14]= 410
```

Since $410 \cdot 34139 \equiv 1 \pmod{999}$, we multiply the inverse matrix by $410 \cdot 34139$ and reduce mod 999 in order to remove the denominators without changing anything mod 999:

```
In[15]:= Mod[410*34139*%%, 999]

Out[15]= {{772, 472, 965}, {641, 516, 851},
{150, 133, 149}}
```

Therefore, the inverse matrix mod 999 is
$\begin{pmatrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{pmatrix}$.

In many cases, it is possible to determine by inspection the common denominator that must be removed. When this is not the case, note that the determinant of the original matrix will always work as a common denominator.

Example 16

Find a square root of 26951623672 mod the prime $p = 98573007539$.

SOLUTION

Since $p \equiv 3 \pmod{4}$, we can use the Proposition of Section 3.9:

```
In[16]:= PowerMod[26951623672, (98573007539
+ 1)/4, 98573007539]

Out[16]= 98338017685
```

The other square root is minus this one:

```
In[17]:= Mod[-%, 98573007539]

Out[17]= 234989854
```

Example 17

Let $n = 34222273 = 9803 \cdot 3491$. Find all four solutions of $x^2 \equiv 19101358 \pmod{34222273}$.

SOLUTION

First, find a square root mod each of the two prime factors, both of which are congruent to $3 \pmod 4$:

```
In[18]:= PowerMod[19101358, (9803 + 1)/4,
9803]
```

```
Out[18]= 3998
```

```
In[19]:= PowerMod[19101358, (3491 + 1)/4,
3491]
```

```
Out[19]= 1318
```

Therefore, the square roots are congruent to $\pm 3998 \pmod{9803}$ and are congruent to $\pm 1318 \pmod{3491}$. There are four ways to combine these using the Chinese remainder theorem:

```
In[20]:= ChineseRemainder[ {3998, 1318 },
{9803, 3491 }]
```

```
Out[20]= 43210
```

```
In[21]:= ChineseRemainder[ {-3998, 1318 },
{9803, 3491 }]
```

```
Out[21]= 8397173
```

```
In[22]:= ChineseRemainder[ {3998, -1318 },
{9803, 3491 }]
```

```
Out[22]= 25825100
```

```
In[23]:= ChineseRemainder[ {-3998, -1318},
{9803, 3491}]
```

```
Out[23]= 34179063
```

These are the four desired square roots.

A.5 Examples for Chapter 5

Example 18

Compute the first 50 terms of the recurrence

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2.$$

The initial values are $0, 1, 0, 0, 0$.

SOLUTION

The vector of coefficients is $\{1, 0, 1, 0, 0\}$ and the initial values are given by the vector $\{0, 1, 0, 0, 0\}$. Type

```
In[1]:= lfsr[{1, 0, 1, 0, 0}, {0, 1, 0, 0,
0}, 50]
```

```
Out[1]= {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1}
```

Example 19

Suppose the first 20 terms of an LFSR sequence are 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1. Find a recurrence that generates this sequence.

SOLUTION

First, we find the length of the recurrence. The command *lfsrlength[v, n]* calculates the determinants mod 2 of the

first n matrices that appear in the procedure in Section 5.2:

In[2]:=

```
lfsrlength[{1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
1, 1, 0, 1, 0, 1, 0, 1}, 10]
     {1, 1}
     {2, 1}
     {3, 0}
     {4, 1}
     {5, 0}
     {6, 1}
     {7, 0}
     {8, 0}
     {9, 0}
     {10, 0}
```

The last nonzero determinant is the sixth one, so we guess that the recurrence has length 6. To find the coefficients:

In[3]:= `lfsrsolve[{1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1}, 6]`

Out[3]= `{1, 0, 1, 1, 1, 0}`

This gives the recurrence as

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2.$$

Example 20

The ciphertext 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0 was produced by adding the output of a LFSR onto the plaintext mod 2 (i.e., XOR the plaintext with the LFSR output). Suppose you know that the plaintext starts 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0. Find the rest of the plaintext.

SOLUTION

XOR the ciphertext with the known part of the plaintext to obtain the beginning of the LFSR output:

```
In[4]:= Mod[{1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 0, 0} + {0, 1, 1, 0, 1, 0,
1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1}, 2]

Out[4]= {1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1}
```

This is the beginning of the LFSR output. Now let's find the length of the recurrence:

```
In[5]:= lfsrlength[%, 8]
```

```
{1, 1}
{2, 0}
{3, 1}
{4, 0}
{5, 1}
{6, 0}
{7, 0}
{8, 0}
```

We guess the length is 5. To find the coefficients of the recurrence:

```
In[6]:= lfsrsolve[%%, 5]

Out[6]= {1, 1, 0, 0, 1}
```

Now we can generate the full output of the LFSR using the coefficients we just found plus the first five terms of the LFSR output:

```
In[7]:= lfsr[{1, 1, 0, 0, 1}, {1, 0, 0, 1,
0}, 40]
```

```
Out[7]={1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0}
```

When we XOR the LFSR output with the ciphertext, we get back the plaintext:

```
In[8]:= Mod[% + {0, 1, 1, 0, 1, 0, 1, 0, 1,
0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
1, 1, 0}, 2]
```

```
Out[8]= {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0}
```

This is the plaintext.

A.6 Examples for Chapter 6

Example 21

The ciphertext

$$22, 09, 00, 12, 03, 01, 10, 03, 04, 08, 01, 17$$

was encrypted using a Hill cipher with matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}.$$

Decrypt it.

SOLUTION

A matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is entered as $\{\{a, b\}, \{c, d\}\}$. Type $M.N$ to multiply matrices M and N. Type $v.M$ to multiply a vector v on the right by a matrix M.

First, we need to invert the matrix mod 26:

```
In[1]:= Inverse[{{ 1,2,3},{ 4,5,6},
{7,8,10}}]
```

Out[1]= $\{\{-\frac{2}{3}, -\frac{4}{3}, 1\}, \{\frac{2}{3}, \frac{11}{3}, -2\}, \{1, -2, 1\}\}$

Since we are working mod 26, we can't stop with numbers like 2/3. We need to get rid of the denominators and reduce mod 26. To do so, we multiply by 3 to extract the numerators of the fractions, then

multiply by the inverse of 3 mod 26 to put the "denominators" back in (see Section 3.3):

```
In[2]:= %*3

Out[2]= {{-2, -4, 3}, {-2, 11, -6}, {3, -6,
3}}

In[3]:= Mod[PowerMod[3, -1, 26]*%, 26]

Out[3]= {{8,16,1}, {8,21,24}, {1,24,1}}
```

This is the inverse of the matrix mod 26. We can check this as follows:

```
In[4]:= Mod[%.{{1, 2, 3}, {4, 5, 6}, {7, 8,
10}}, 26]

Out[4]= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

To decrypt, we break the ciphertext into blocks of three numbers and multiply each block on the right by the inverse matrix we just calculated:

```
In[5]:= Mod[{22, 09, 00}.%%, 26]

Out[5]= {14, 21, 4}

In[6]:= Mod[{12, 03, 01}.%%%, 26]

Out[6]= {17, 19, 7}

In[7]:= Mod[{10, 03, 04}.%%%%, 26]

Out[7]= {4, 7, 8}

In[8]:= Mod[{08, 01, 17}.%%%%%, 26]

Out[8]= {11, 11, 23}
```

Therefore, the plaintext is 14, 21, 4, 17, 19, 7, 4, 7, 8, 11, 11, 23. This can be changed back to letters:

```
In[9]:= num2txt0[142104171907040708111123]

Out[9]= overthehillx
```

Note that the final `x` was appended to the plaintext in order to complete a block of three letters.

A.7 Examples for Chapter 9

Example 22

Suppose you need to find a large random prime of 50 digits. Here is one way. The function *NextPrime[x]* finds the next prime greater than x. The function *Random[Integer,{a,b}]* gives a random integer between a and b. Combining these, we can find a prime:

```
In[1]:= NextPrime[Random[Integer, {10^49,
10^50 }]]

Out[1]=
73050570031667109175215303340488313456708913284291
```

If we repeat this procedure, we should get another prime:

```
In[2]:= NextPrime[Random[Integer, {10^49,
10^50 }]]

Out[2]=
97476407694931303255724326040586144145341054568331
```

Example 23

Suppose you want to change the text *hellohowareyou* to numbers:

```
In[3]:= txt2num1["hellohowareyou"]

Out[3]= 805121215081523011805251521
```

Note that we are now using $a = 1, b = 2, \ldots, z = 26$, since otherwise *a*'s at the beginnings of messages would disappear. (A more efficient procedure would be to work in base 27, so the numerical form of the message would be $8 + 5 \cdot 27 + 12 \cdot 27^2 + \ldots + 21 \cdot 27^{13} = 87495221502384554951$. Note that this uses fewer digits.)

Now suppose you want to change it back to letters:

In[4]:=
```
num2txt1[805121215081523011805251521]
```

Out[4]= `hellohowareyou`

Example 24

Encrypt the message *hi* using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the message to numbers:

In[5]:= `txt2num1["hi"]`

Out[5]= `809`

Now, raise it to the eth power mod n:

In[6]:= `PowerMod[%, 17, 823091]`

Out[6]= `596912`

Example 25

Decrypt the ciphertext in the previous problem.

SOLUTION

First, we need to find the decryption exponent d. To do this, we need to find $\phi(823091)$. One way is as follows:

```
In[7]:= EulerPhi[823091]

Out[7]= 821184
```

Another way is to factor n as $p \cdot q$ and then compute $(p-1)(q-1)$:

```
In[8]:= FactorInteger[823091]

Out[8]= { {659, 1 }, {1249, 1 } }

In[9]:= 658*1248

Out[9]= 821184
```

Since $de \equiv 1 \pmod{\phi(n)}$, we compute the following (note that we are finding the inverse of $e \bmod \phi(n)$, not $\bmod n$):

```
In[10]:= PowerMod[17, -1, 821184]

Out[10]= 48305
```

Therefore, $d = 48305$. To decrypt, raise the ciphertext to the dth power $\bmod n$:

```
In[11]:= PowerMod[596912, 48305, 823091]

Out[11]= 809
```

Finally, change back to letters:

```
In[12]:= num2txt1[809]

Out[12]= hi
```

Example 26

Encrypt *hellohowareyou* using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the plaintext to numbers:

```
In[13]:= txt2num1["hellohowareyou"]

Out[13]= 805121215081523011805251521
```

Suppose we simply raised this to the eth power mod n :

```
In[14]:= PowerMod[%, 17, 823091]

Out[14]= 447613
```

If we decrypt (we know d from Example 25), we obtain

```
In[15]:= PowerMod[%, 48305, 823091]

Out[15]= 628883
```

This is not the original plaintext. The reason is that the plaintext is larger than n, so we have obtained the plaintext mod n:

```
In[16]:= Mod[805121215081523011805251521,
823091]

Out[16]= 628883
```

We need to break the plaintext into blocks, each less than n. In our case, we use three letters at a time:

$$80512 \quad 121508 \quad 152301 \quad 180525 \quad 1521$$

```
In[17]:= PowerMod[80512, 17, 823091]
```

```
Out[17]= 757396

In[18]:= PowerMod[121508, 17, 823091]

Out[18]= 164513

In[19]:= PowerMod[152301, 17, 823091]

Out[19]= 121217

In[20]:= PowerMod[180525, 17, 823091]

Out[20]= 594220

In[21]:= PowerMod[1521, 17, 823091]

Out[21]= 442163
```

The ciphertext is therefore 757396164513121217594220442163. Note that there is no reason to change this back to letters. In fact, it doesn't correspond to any text with letters.

Decrypt each block individually:

```
In[22]:= PowerMod[757396, 48305, 823091]

Out[22]= 80512

In[23]:= PowerMod[164513, 48305, 823091]

Out[23]= 121508
```

Etc.

We'll now do some examples with large numbers, namely the numbers in the RSA Challenge discussed in Section 9.5. These are stored under the names *rsan, rsae, rsap, rsaq*:

```
In[24]:= rsan
```

Out[24]=

```
1143816257578888676692357799761466120102182967212
42362562561842935
7069352457338978305971235639587050589890751475992
90026879543541
```

In[25]:= `rsae`

Out[25]= 9007

Example 27

Encrypt each of the messages *b, ba, bar, bard* using *rsan* and *rsae*.

In[26]:= `PowerMod[num1["b"], rsae, rsan]`

Out[26]=

```
7094675846761266859837016499155078618287633106068
52354105647041144
8678226171649720012215533234846201405328798758089
9263765142534
```

In[27]:= `PowerMod[txt2num1["ba"], rsae, rsan]`

Out[27]=

```
3504513060897510032501170944987195427378820475394
85930603136976982
2762175980602796227053803156556477335203367178226
1305796158951
```

In[28]:= `PowerMod[txt2num1["bar"], rsae, rsan]`

Out[28]=

```
448145128638551010760045308594921093424295316066074090703605434080
008436459868804059531028183128225863625802987844411519226064 24
```

In[29]:= `PowerMod[txt2num1["bard"], rsae, rsan]`

Out[29]=

```
242380777851116664232028625120903173934852129590562707831349916142
560543232971798049289580734457526630264498739868779893299094 98
```

Observe that the ciphertexts are all the same length. There seems to be no easy way to determine the length of the corresponding plaintext.

Example 28

Using the factorization $rsan = rsap \cdot rsaq$, find the decryption exponent for the RSA Challenge, and decrypt the ciphertext (see Section 9.5).

SOLUTION

First we find the decryption exponent:

In[30]:=`rsad=PowerMod[rsae,-1,(rsap-1)*(rsaq-1)];`

Note that we use the final semicolon to avoid printing out the value. If you want to see the value of *rsad*, see Section 9.5, or don't use the semicolon. To decrypt the ciphertext, which is stored as *rsaci*, and change to letters:

In[31]:=num2txt1[PowerMod[rsaci, rsad, rsan]]

Out[31]= the magic words are squeamish ossifrage

Example 29

Encrypt the message *rsaencryptsmessageswell* using *rsan* and *rsae*.

In[32]:= PowerMod[txt2num1["rsaencryptsmessageswell"], rsae, rsan]

Out[32]=

```
9463942034900225931630582353924949641464096993400
17097214043524182
7195065425436558490601396632881775353928311265319
7553130781884
```

Example 30

Decrypt the preceding ciphertext.

SOLUTION

Fortunately, we know the decryption exponent *rsad*. Therefore, we compute

In[33]:= PowerMod[%, rsad, rsan]

Out[33]= 1819010514031825162019130519190107051923051212

In[34]:= `num2txt1[%]`

Out[34]= `rsaencryptsmessageswell`

Suppose we lose the final 4 of the ciphertext in transmission. Let's try to decrypt what's left (subtracting 4 and dividing by 10 is a mathematical way to remove the 4):

In[35]:= `PowerMod[(%%% - 4)/10, rsad, rsan]`

Out[35]=

```
479529991731959886649023526295254864091136338943756298468549079705
884123003734879696577942541171589569212679126284614944756828 06
```

If we try to change this to letters, we get a long error message. A small error in the plaintext completely changes the decrypted message and usually produces garbage.

Example 31

Suppose we are told that $n = 11313771275590312567$ is the product of two primes and that $\phi(n) = 11313771187608744400$. Factor n.

SOLUTION

We know (see Section 9.1) that p and q are the roots of $X^2 - (n - \phi(n) + 1)X + n$. Therefore, we compute

In[36]:= `Roots[Xˆ2 - (11313771275590312567 - 11313771187608744400 + 1)*X + 11313771275590312567 == 0, X]`

Out[36]= $X == 128781017 \mid\mid X == 87852787151$

Therefore, $n = 128781017 \cdot 87852787151$. We also could have used the quadratic formula to find the roots.

Example 32

Suppose we know *rsae* and *rsad*. Use these to factor *rsan*.

SOLUTION

We use the $a^r \equiv 1$ factorization method from Section 9.4. First write $rsae \cdot rsad - 1 = 2^s m$ with m odd. One way to do this is first to compute $rsae \cdot rsad - 1$, then keep dividing by 2 until we get an odd number:

In[37]:=
```
rsae*rsad - 1
```

Out[37]=

```
961034419617782266156919023359583834109854129051878330250644604041
155985575087352659156174898557342995131594680431086921245830097664
```

In[38]:=
```
%/2
```

Out[38]=

```
480517209808891133078459511679791917054927064525939165125322302020
577992787543676329578087449278671497565797340215543460622915048832
```

In[39]:=
```
%/2
```

Out[39]=

```
240258604904445566539229755839895958527463532262969582562661151010
288996393771838164789043724639335748782898670107771730311457524416
```

We continue this way for six more steps until we get

Out[45]=

```
375404070163196197717546493499837435199161769160889972754158048453
576556865268497132482880819748962107473279172043393328611652381 9
```

This number is m. Now choose a random integer a. Hoping to be lucky, we choose 13. As in the $a^r \equiv 1$ factorization method, we compute

In[46]:= `PowerMod[13, %, rsan]`

Out[46]=

```
275743685070065305922434948688471611984230957073078056905698396470301831098398623708005293380929847954901926435879608598705512 39
```

Since this is not $\pm 1 \pmod{rsan}$, we successively square it until we get ± 1:

In[47]:= `PowerMod[%, 2, rsan]`

Out[47]=

```
483189603219285155801384764187230345541040990699408462254947027766549964125829556360352661561086864311942985740758540375122772 92
```

In[48]:= `PowerMod[%, 2, rsan]`

Out[48]=

```
781728141548773565791419280587540000219487870564838209179306251152
151818397420560132755219134875609447320735164877222273875579363
```

In[49]:= `PowerMod[%, 2, rsan]`

Out[49]=

```
428361912025087287421992990405829002029762229160177671675518702165
094445182394621863794705694420551013929922930822596017382287 02
```

In[50]:= `PowerMod[%, 2, rsan]`

Out[50]= 1

Since the last number before the 1 was not $\pm 1 \pmod{rsan}$, we have an example of $x \not\equiv \pm 1 \pmod{rsan}$ with $x^2 \equiv 1$. Therefore, $\gcd(x-1, rsan)$ is a nontrivial factor of *rsan*:

In[51]:= `GCD[%% - 1, rsan]`

Out[51]=

```
327691329932667095499619881908344614131776429679929425395798288533
```

This is *rsaq*. The other factor is obtained by computing *rsan/rsaq*:

In[52]:= `rsan/%`

Out[52]=

```
34905295108476509491478496199038981334177646384933
87843990820577
```

This is *rsap*.

Example 33

Suppose you know that

$$150883475569451^2 \equiv 16887570532858^2 \pmod{205611444308117}.$$

Factor 205611444308117.

SOLUTION

We use the Basic Principle of Section 9.4.

In[53]:= `GCD[150883475569451-16887570532858,205611444308117]`

Out[53]= `23495881`

This gives one factor. The other is

In[54]:= `205611444308117/%`

Out[54]= `8750957`

We can check that these factors are actually primes, so we can't factor any further:

In[55]:= `PrimeQ[%%]`

Out[55]= `True`

In[56]:= `PrimeQ[%%]`

Out[56]= `True`

Example 34

Factor
$n = 37687557542639485559998999289787 3239$
by the $p - 1$ method.

SOLUTION

Let's choose our bound as $B = 100$, and let's take
$a = 2$, so we compute $2^{100!} \pmod{n}$:

In[57]:=
PowerMod[2,Factorial[100],3768755754263948 5559998999289787 3239]

Out[57]=
`3696766783019563319394221062511995 12`

Then we compute the gcd of $2^{100!} - 1$ and n:

In[58]:= `GCD[% - 1, 37687557542639485559998999289787 3239]`

Out[58]= `430553161739796481`

This is a factor p. The other factor q is

In[59]:=
`37687557542639485559998999289787 3239/%`

Out[59]= `875328783798732119`

Let's see why this worked. The factorizations of $p - 1$
and $q - 1$ are

In[60]:= `FactorInteger[430553161739796481 - 1]`

```
Out[60]= {{2, 18 }, {3, 7 }, {5, 1 }, {7, 4
}, {11, 3 }, {47, 1 }}

In[61]:= FactorInteger[875328783798732119 -
1]

Out[61]= {{2, 1 }, {61, 1 }, {20357, 1 },
{39301, 1 }, {8967967, 1 }}
```

We see that $100!$ is a multiple of $p-1$, so $2^{100!} \equiv 1 \pmod{p}$. However, $100!$ is not a multiple of $q-1$, so it is likely that $2^{100!} \not\equiv 1 \pmod{q}$. Therefore, both $2^{100!}-1$ and pq have p as a factor, but only pq has q as a factor. It follows that the gcd is p.

A.8 Examples for Chapter 10

Example 35

Let's solve the discrete log problem $2^x \equiv 71 \pmod{131}$ by the Baby Step-Giant Step method of Subsection 10.2.2. We take $N = 12$ since $N^2 > p - 1 = 130$ and we form two lists. The first is $2^j \pmod{1}31$ for $0 \leq j \leq 11$:

```
In[1]:=Do[Print[n, " " , PowerMod[2, n,
131]], {n, 0, 11}]
```

Out[1]= 0 1

```
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 125
9 119
10 107
11 83
```

The second is $71 \cdot 2^{-j} \pmod{1}31$ for $0 \leq j \leq 11$:

```
In[2]:=Do[Print[n, " " , Mod[71*PowerMod[2,
-12*n, 131], 131]], {n, 0, 11}]
```

Out[2]= > 0 71

```
1 17
2 124
3 26
4 128
```

```
5 86
6 111
7 93
8 85
9 96
10 130
11 116
```

The number 128 is on both lists, so we see that $2^7 \equiv 71 \cdot 2^{-12*4} \pmod{131}$. Therefore,

$$71 \equiv 2^{7+4*12} \equiv 2^{55} \pmod{131}.$$

A.9 Examples for Chapter 12

Example 36

Suppose there are 23 people in a room. What is the probability that at least two have the same birthday?

SOLUTION

The probability that no two have the same birthday is $\prod_{i=1}^{22}(1 - i/365)$ (note that the product stops at $i = 22$, not $i = 23$). Subtracting from 1 gives the probability that at least two have the same birthday:

```
In[1]:= 1 - Product[1. - i/365, {i, 22}]

Out[1]= 0.507297
```

Note that we used 1. in the product instead of 1 without the decimal point. If we had omitted the decimal point, the product would have been evaluated as a rational number (try it, you'll see).

Example 37

Suppose a lazy phone company employee assigns telephone numbers by choosing random seven-digit numbers. In a town with 10,000 phones, what is the probability that two people receive the same number?

```
In[2]:= 1 - Product[1. - i/10^7, {i, 9999}]

Out[2]= 0.99327
```

Note that the number of phones is about three times the square root of the number of possibilities. This means that we expect the probability to be high, which it is. From Section 12.1, we have the estimate that if there are around $\sqrt{2(\ln 2)10^7} \approx 3723$ phones, there should be a 50% chance of a match. Let's see how accurate this is:

```
In[3]:= 1 - Product[1. - i/10^7, i, 3722]

Out[3]= 0.499895
```

A.10 Examples for Chapter 17

Example 38

Suppose we have a (5, 8) Shamir secret sharing scheme. Everything is mod the prime $p = 987541$. Five of the shares are

(9853, 853), (4421, 4387), (6543, 1234), (93293, 78428), (12398, 7563).

Find the secret.

SOLUTION

One way: First, find the Lagrange interpolating polynomial through the five points:

In[1]:=
```
InterpolatingPolynomial[ { {9853,
853 }, {4421, 4387 }, {6543, 1234 },
{93293, 78428 }, {12398, 7563 } }, x]
```

Out[1]=

$$853 + (-\frac{1767}{2716} + (+\frac{2406987}{9538347560} + (-\frac{8464915920541}{3130587195363428640000}) - \frac{4959003720134640533754 7(-93293 + x)}{13378864151099487659488222 6797600000})(-6543 + x))(-4421 + x))(-9853 + x)$$

Now evaluate at $x = 0$ to find the constant term (use $/.x -> 0$ to evaluate at $x = 0$):

In[2]:=
```
%/. x - > 0
```

Out[2]=

$$\frac{204484326154044983230114592433944282591}{22298106918499146099147037799600000}$$

We need to change this to an integer mod 987541, so we find the multiplicative inverse of the denominator:

```
In[3]:= PowerMod[Denominator[%], -1, 987541]

Out[3]= 509495
```

Now, multiply times the numerator to get the desired integer:

```
In[4]:= Mod[Numerator[%%]*%, 987541]

Out[4]= 678987
```

Therefore, 678987 is the secret.

A.11 Examples for Chapter 18

Example 39

Here is a game you can play. It is essentially the simplified version of poker over the telephone from Section 18.2. There are five cards: ten, jack, queen, king, ace. They are shuffled and disguised by raising their numbers to a random exponent mod the prime 24691313099. You are supposed to guess which one is the ace. To start, pick a random exponent. We use the semicolon after *khide* so that we cannot cheat and see what value of k is being used.

```
In[1]:= k = khide;
```

Now, shuffle the disguised cards (their numbers are raised to the kth power mod p and then randomly permuted):

```
In[2]:= shuffle

Out[2]= {14001090567, 16098641856,
23340023892, 20919427041, 7768690848}
```

These are the five cards (yours will differ from these because the k and the random shuffle will be different). None looks like the ace; that's because their numbers have been raised to powers mod the prime. Make a guess anyway. Let's see if you're correct.

```
In[3]:= reveal[%]

Out[3]= {ten, ace, queen, jack, king}
```

Let's play again:

```
In[4]:= k = khide;

In[5]:= shuffle

Out[5]= {13015921305, 14788966861,
23855418969, 22566749952, 8361552666}
```

Make your guess (note that the numbers are different because a different random exponent was used). Were you lucky?

```
In[6]:= reveal[%]

Out[6]= {ten, queen, ace, king, jack}
```

Perhaps you need some help. Let's play one more time:

```
In[7]:= k = khide;

In[8]:= shuffle

Out[8]= {13471751030, 20108480083,
8636729758, 14735216549, 11884022059}
```

We now ask for advice:

```
In[9]:= advise[%]

Out[9]= 3
```

We are advised that the third card is the ace. Let's see (note that %% is used to refer to the next to last output):

```
In[10]:= reveal[%%]

Out[10]= {jack, ten, ace, queen, king}
```

How does this work? Read the part on "How to Cheat" in Section 18.2. Note that if we raise the numbers for the

cards to the $(p-1)/2$ power $\bmod p$, we get

```
In[11]:= PowerMod[{200514, 10010311,
1721050514, 11091407, 10305}, (24691313099
- 1)/ 2, 24691313099]

Out[11]= {1, 1, 1, 1, 24691313098}
```

Therefore, only the ace is a quadratic nonresidue $\bmod p$.

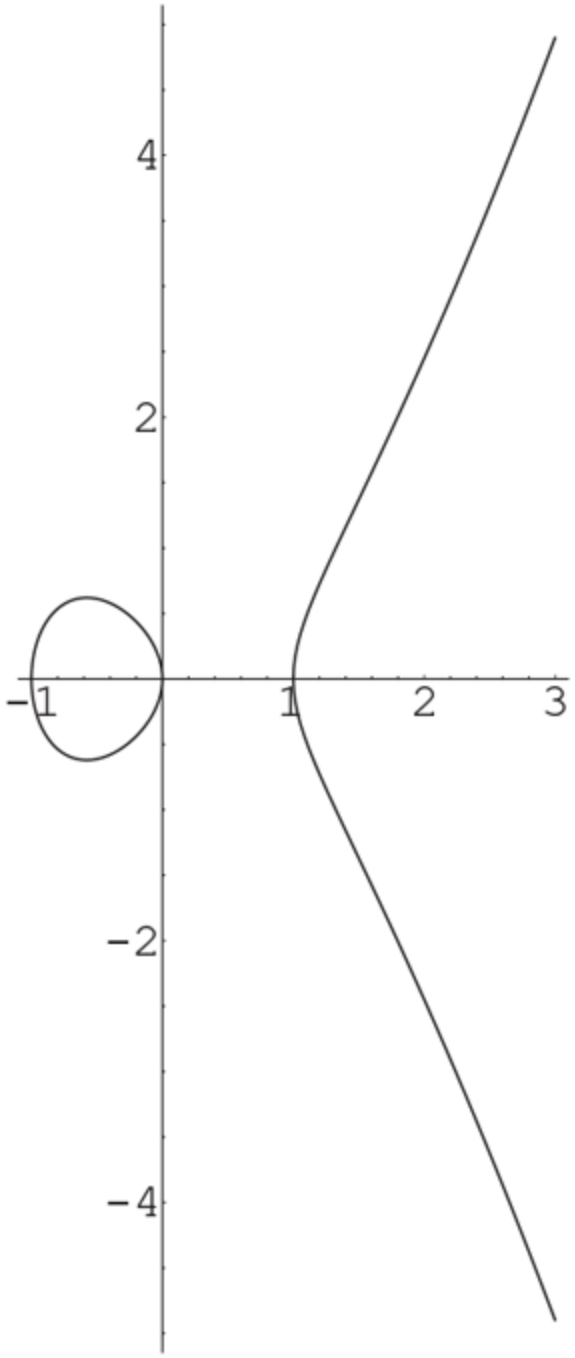
A.12 Examples for Chapter 21

Example 40

All of the elliptic curves we work with in this chapter are elliptic curves $\mod n$. However, it is helpful to use the graphs of elliptic curves with real numbers in order to visualize what is happening with the addition law, for example, even though such pictures do not exist $\mod n$. Therefore, let's graph the elliptic curve $y^2 = x(x-1)(x+1)$. We'll specify that $-1 \leq x \leq 3$ and $-y \leq y \leq 5$:

```
In[1]:= ContourPlot[y^2 == x*(x - 1)*(x +
1), {x, -1, 3 }, {y, -5, 5 }]
```

Graphics



Full Alternative Text

Example 41

Add the points (1, 3) and (3, 5) on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$.

```
In[2]:= addell[ {1, 3 }, {3, 5 }, 24, 13,
29]

Out[2]= {26, 1 }
```

You can check that the point (26, 1) is on the curve: $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$.

Example 42

Add (1, 3) to the point at infinity on the curve of the previous example.

```
In[3]:= addell[ {1, 3 }, {"infinity",
"infinity" }, 24, 13, 29]

Out[3]= {1, 3 }
```

As expected, adding the point at infinity to a point P returns the point P.

Example 43

Let $P = (1, 3)$ be a point on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$. Find $7P$.

```
In[4]:= multell[ {1, 3 }, 7, 24, 13, 29]

Out[4]= {15, 6 }
```

Example 44

Find $k(1, 3)$ for $k = 1, 2, 3, \ldots, 40$ on the curve of the previous example.

```
In[5]:= multsell[ {1, 3 }, 40, 24, 13, 29]
```

```
Out[5]= {1,{1,3},2,{11,10},3,{23,28},4,
{0,10},5,{19,7},6,{18,19},7, {15,6},8,
{20,24},9,{4,12},10,{4,17},11,{20,5},12,
{15,23},13,{18,10}, 14,{19,22},15,{0,19},
16,{23,1},17,{11,19},18,{1,26},19,
{infinity,infinity},20,{1,3},21,{11,10},
22,{23,28},23,{0,10}, 24,{19,7}, 25,
{18,19},26,{15,6},27,{20,24},28,{4,12},29,
{4,17}, 30,{20,5},31,{15,23},32,
{18,10},33,{19,22}, 34,{0,19},35,
{23,1},36, {11,19},37,{1,26}, 38,
{infinity,infinity},39,{1,3},40,{11,10}}
```

Notice how the points repeat after every 19 multiples.

Example 45

The previous four examples worked mod the prime 29. If we work mod a composite number, the situation at infinity becomes more complicated since we could be at infinity mod both factors or we could be at infinity mod one of the factors but not mod the other. Therefore, we stop the calculation if this last situation happens and we exhibit a factor. For example, let's try to compute $12P$, where $P = (1, 3)$ is on the elliptic curve $y^2 \equiv x^3 - 5x + 13 \pmod{209}$:

```
In[6]:= multell[ {1, 3 }, 12, -5, 13, 11*19]

Out[6]= {factor=, 19 }
```

Now let's compute the successive multiples to see what happened along the way:

```
In[7]:= multsell[ {1, 3 }, 12, -5, 13,
11*19]
```

```
Out[7]= 1,{{1,3},2,{91,27},3,{118,133},4,
{148,182},5,{20,35}, 6,{factor=, 19}}
```

When we computed $6P$, we ended up at infinity mod 19. Let's see what is happening mod the two prime factors of 209, namely 19 and 11:

```
In[8]:= multsell[{1,3}, 12, -5, 13, 19]

Out[8]= 1,{{1,3},2,{15,8},3,{4,0},4,
{15,11},5,{1,16}, 6,{infinity,infinity},
7,{1,3},8,{15,8},9,{4,0},10,{15,11}, 11,
{1,16}, 12,{infinity,infinity}}

In[9]:= multsell[ {1, 3 }, 20, -5, 13, 11]

Out[9]= 1,{{1,3},2,{3,5},3,{8,1},4,{5,6},5,
{9,2},6,{6,10},7,{2,0}, 8,{6,1},9,
{9,9},10,{5,5},11,{8,10},12,{3,6},13,
{1,8}, 14,{infinity,infinity},15,{1,3},
16,{3,5},17,{8,1},18,{5,6}, 19,{9,2},20,
{6,10}}
```

After six steps, we were at infinity mod 19, but it takes 14 steps to reach infinity mod 11. To find $6P$, we needed to invert a number that was 0 mod 19 and nonzero mod 11. This couldn't be done, but it yielded the factor 19. This is the basis of the elliptic curve factorization method.

Example 46

Factor 193279 using elliptic curves.

SOLUTION

First, we need to choose some random elliptic curves and a point on each curve. For example, let's take $P = (2, 4)$ and the elliptic curve

$$y^2 \equiv x^3 - 10x + b \pmod{193279}.$$

For P to lie on the curve, we take $b = 28$. We'll also take

$$\begin{aligned} y^2 &\equiv x^3 + 11x - 11, \quad P = (1, 1) \\ y^2 &\equiv x^3 + 17x - 14, \quad P = (1, 2). \end{aligned}$$

Now we compute multiples of the point P. We do the analog of the $p - 1$ method, so we choose a bound B, say $B = 12$, and compute $B!P$.

```
In[10]:= multell[{2,4}, Factorial[12], -10,
28, 193279]

Out[10]= {factor=, 347}

In[11]:= multell[{1,1}, Factorial[12], 11,
-11, 193279]

Out[11]= {13862, 35249}

In[12]:= multell[{1, 2}, Factorial[12], 17,
-14, 193279]

Out[12]= {factor=, 557}
```

Let's analyze in more detail what happened in these examples.

On the first curve, $266P$ ends up at infinity mod 557 and $35P$ is infinity mod 347. Since $266 = 2 \cdot 7 \cdot 19$, it has a prime factor larger than $B = 12$, so $B!P$ is not infinity mod 557. But 35 divides $B!$, so $B!P$ is infinity mod 347.

On the second curve, $356P = \text{infinity}$ mod 347 and $561P = \text{infinity}$ mod 557. Since $356 = 4 \cdot 89$ and $561 = 3 \cdot 11 \cdot 17$, we don't expect to find the factorization with this curve.

The third curve is a surprise. We have $331P = \text{infinity}$ mod 347 and $272P = \text{infinity}$ mod 557. Since 331 is prime and $272 = 16 \cdot 17$, we don't expect to find the

factorization with this curve. However, by chance, an intermediate step in the calculation of $B!P$ yielded the factorization. Here's what happened. At one step, the program required adding the points (184993, 13462) and (20678, 150484). These two points are congruent mod 557 but not mod 347. Therefore, the slope of the line through these two points is defined mod 347 but is 0/0 mod 557. When we tried to find the multiplicative inverse of the denominator mod 193279, the gcd algorithm yielded the factor 557. This phenomenon is fairly rare.

Example 47

Here is how to produce the example of an elliptic curve ElGamal cryptosystem from Section 21.5. For more details, see the text. The elliptic curve is $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$ and the point is $G = (4, 11)$. Alice's message is the point $P_m = (5, 1743)$.

Bob has chosen his secret random number $a_B = 3$ and has computed $a_B G$:

```
In[13]:= multell[{4, 11}, 3, 3, 45, 8831]

Out[13]= {413, 1808}
```

Bob publishes this point. Alice chooses the random number $k = 8$ and computes kG and $P_m + k(a_B G)$:

```
In[14]:= multell[{4, 11}, 8, 3, 45, 8831]

Out[14]= {5415, 6321}

In[15]:= addell[{5, 1743}, multell[{413,
1808}, 8, 3, 45, 8831], 3, 45, 8831]

Out[15]= {6626, 3576}
```

Alice sends (5415, 6321) and (6626, 3576) to Bob, who multiplies the first of these points by a_B:

```
In[16]:= multell[{5415, 6321}, 3, 3, 45,
8831]

Out[16]= {673, 146}
```

Bob then subtracts the result from the last point Alice sends him. Note that he subtracts by adding the point with the second coordinate negated:

```
In[17]:= addell[{6626, 3576}, {673, -146},
3, 45, 8831]

Out[17]= {5, 1743}
```

Bob has therefore received Alice's message.

Example 48

Let's reproduce the numbers in the example of a Diffie-Hellman key exchange from Section 21.5: The elliptic curve is $y^2 \equiv x^3 + x + 7206 \pmod{7211}$ and the point is $G = (3, 5)$. Alice chooses her secret $N_A = 12$ and Bob chooses his secret $N_B = 23$. Alice calculates

```
In[18]:= multell[{3, 5}, 12, 1, 7206, 7211]

Out[18]= {1794, 6375}
```

She sends (1794,6375) to Bob. Meanwhile, Bob calculates

```
In[19]:= multell[{3, 5}, 23, 1, 7206, 7211]

Out[19]= {3861, 1242}
```

and sends (3861,1242) to Alice. Alice multiplies what she receives by N_A and Bob multiplies what he receives by

N_B:

```
In[20]:= multell[{3861, 1242}, 12, 1, 7206,
7211]

Out[20]= {1472, 2098}

In[21]:= multell[{1794, 6375}, 23, 1, 7206,
7211]

Out[21]= {1472, 2098}
```

Therefore, Alice and Bob have produced the same key.

Appendix B Maple® Examples

These computer examples are written in Maple. If you have Maple available, you should try some of them on your computer. If Maple is not available, it is still possible to read the examples. They provide examples for several of the concepts of this book. For information on getting started with Maple, see Section B.1. To download a Maple notebook that contains the necessary commands, go to

```
bit.ly/2TzKFec
```

B.1 Getting Started with Maple

1. Download the Maple notebook math.mws that you find using the links starting at `bit.ly/2TzKFec`

2. Open Maple (on a Linux machine, use the command `xmaple`; on most other systems, click on the Maple icon)), then open math.mws using the menu options under File on the command bar at the top of the Maple window. (Perhaps this is done automatically when you download it; it depends on your computer settings.)

3. With math.mws in the foreground, press the Enter or Return key on your keyboard. This will load the functions and packages needed for the following examples.

4. You are now ready to use Maple. If you want to try something easy, type $1 + 2^*3 + 4\ \hat{}\ 5$; and then press the Return/Enter key. The result 1031 should appear (it's $1 + 2 \cdot 3 + 4^5$).

5. Go to the Computer Examples in Section B.3. Try typing in some of the commands there. The outputs should be the same as those in the examples. Press the Return or Enter key to make Maple evaluate an expression.

6. If you make a mistake in typing in a command and get an error message, you can edit the command and hit Return or Enter to try again. You don't need to retype everything.

7. If you are looking for help or a command to do something, try the Help menu on the command bar at the top. If you can guess the name of a function, there is another way. For example, to obtain information on gcd, type `?gcd` and Return or Enter.

B.2 Some Commands

The following are some Maple commands that are used in the examples. Some, such as `phi`, are built into Maple. Others, such as `addell`, are in the Maple notebook available at

```
bit.ly/2TzKFec
```

If you want to suppress the output, use a colon instead.

The argument of a function is enclosed in round parentheses. Vectors are enclosed in square brackets. Entering `matrix(m,n,[a,b,c,...,z])` gives the $m \times n$ matrix with first row `a,b, ...` and last row `...z`. To multiply two matrices A and B, type `evalm(A&*B)`.

If you want to refer to the previous output, use %. The next-to-last output is %%, etc. Note that % refers to the most recent output, not to the last displayed line. If you will be referring to an output frequently, it might be better to name it. For example, `g:=phi(12345)` defines `g` to be the value of $\phi(12345)$. Note that when you are assigning a value to a variable in this way, you should use a colon before the equality sign. Leaving out the colon is a common cause of hard-to-find errors.

Exponentiation is written as $\mathtt{a \hat{} b}$. However, we will need to use modular exponentiation with very large exponents. In that case, use $\mathtt{a\&\hat{}\,b} \bmod \mathtt{n}$. For modular exponentiation, you might need to use a $\backslash$ between & and ^. Use the right arrow to escape from the exponent.

Some of the following commands require certain Maple packages to be loaded via the commands

```
with(numtheory), with(linalg), with(plots),
with(combinat)
```

These are loaded when the math.mws notebook is loaded. However, if you want to use a command such as `nextprime` without loading the notebook, first type `with(numtheory):` to load the package (once for the whole session). Then you can use functions such as `nextprime`, `isprime`, etc. If you type `with(numtheory)` without the colon, you'll get a list of the functions in the package, too.

The following are some of the commands used in the examples. We list them here for easy reference. To see how to use them, look at the examples. We have used `txt` to refer to a string of letters. Such strings should be enclosed in quotes (`"string"`).

`addell([x,y], [u,v], b, c, n)` finds the sum of the points (x, y) and (u, v) on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod{n}$. The integer n should be odd.

`affinecrypt(txt,m,n)` is the affine encryption of `txt` using $mx + n$.

`allshifts(txt)` gives all 26 shifts of `txt`.

`chrem([a,b,...], [m,n,...])` gives a solution to the simultaneous congruences $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$, $\ldots$.

`choose(txt,m,n)` lists the characters in `txt` in positions that are congruent to $n \pmod{m}$.

`coinc(txt,n)` is the number of matches between `txt` and `txt` displaced by n.

`corr(v)` is the dot product of the vector v with the 26 shifts of the alphabet frequency vector.

`phi(n)` computes $\phi(n)$ (don't try very large values of n).

`igcdex(m,n,'x','y')` computes the gcd of m and n along with a solution of $mx + ny = \gcd$. To get `x` and `y`, type `x;y` on this or a subsequent command line.

`ifactor(n)` factors n.

`frequency(txt)` lists the number of occurrences of each letter `a` through `z` in `txt`.

`gcd(m,n)` is the gcd of m and n.

`inverse(M)` finds the inverse of the matrix M.

`lfsr(c,k,n)` gives the sequence of n bits produced by the recurrence that has coefficients given by the vector c. The initial values of the bits are given by the vector k.

`lfsrlength(v,n)` tests the vector v of bits to see if it is generated by a recurrence of length at most n.

`lfsrsolve(v,n)` computes the coefficients of a recurrence, given a guess n for the length of the recurrence that generates the binary vector v.

`max(v)` is the largest element of the list v.

`a mod n` is the value of $a \pmod n$.

`multell([x,y], m, b, c, n)` computes m times the point (x, y) on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod n$.

`multsell([x,y], m, b, c, n)` lists the first m multiples of the point (x, y) on the elliptic curve $y^2 \equiv x^3 + bx + c \pmod n$.

`nextprime(x)` gives the next prime $> x$.

`num2text(n)` changes a number n to letters. The successive pairs of digits must each be at most 26 *space* is 00, a is 01, z is 26.

`primroot(p)` finds a primitive root for the prime p.

`shift(txt,n)` shifts `txt` by n.

`text2num(txt)` changes `txt` to numbers, with `space=00, a=01, ..., z=25`.

`vigenere(txt,v)` gives the Vigenère encryption of `txt` using the vector v as the key.

`vigvec(txt,m,n)` gives the frequencies of the letters `a` through `z` in positions congruent to $n \pmod m$.

B.3 Examples for Chapter 2

Example 1

A shift cipher was used to obtain the ciphertext `kddkmu`. Decrypt it by trying all possibilities.

```
> allshifts("kddkmu")
                        "kddkmu"
                        "leelnv"
                        "mffmow"
                        "nggnpx"
                        "ohhoqy"
                        "piiprz"
                        "qjjqsa"
                        "rkkrtb"
                        "sllsuc"
                        "tmmtvd"
                        "unnuwe"
                        "voovxf"
                        "wppwyg"
                        "xqqxzh"
                        "yrryai"
                        "zsszbj"
                        "attack"
                        "buubdl"
                        "cvvcem"
                        "dwwdfn"
                        "exxego"
                        "fyyfhp"
                        "gzzgiq"
                        "haahjr"
                        "ibbiks"
                        "jccjlt"
```

As you can see, `attack` is the only word that occurs on this list, so that was the plaintext.

Example 2

Encrypt the plaintext message `cleopatra` using the affine function $7x + 8$:

```
> affinecrypt("cleopatra", 7, 8)

                              "whkcjilxi"
```

Example 3

The ciphertext `mzdvezc` was encrypted using the affine function $5x + 12$. Decrypt it.

SOLUTION

First, solve $y \equiv 5x + 12 \pmod{26}$ for x to obtain $x \equiv 5^{-1}(y - 12)$. We need to find the inverse of $5 \pmod{26}$:

```
> 5& ^      (−1) mod 26

                                21
```

(On some computers, the `^` doesn't work. Instead, type a backslash \ and then `^`. Use the right arrow key to escape from the exponent before typing `mod`. For some reason, a space is needed before a parenthesis in an exponent.)

Therefore, $x \equiv 21(y - 12) \equiv 21y - 12 \cdot 21$. To change $-12 \cdot 21$ to standard form:

```
> −12*21   mod 26

                                8
```

Therefore, the decryption function is $x \equiv 21y + 8$. To decrypt the message:

```
> affinecrypt("mzdvezc", 21, 8)

                              "anthony"
```

In case you were wondering, the plaintext was encrypted as follows:

```
> affinecrypt("anthony", 5, 12)

                              "mzdvezc"
```

Example 4

Here is the example of a Vigenère cipher from the text. Let's see how to produce the data that was used in Section 2.3 to decrypt it. For convenience, we've already stored the ciphertext under the name vvhq.

```
> vvhq

vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

Find the frequencies of the letters in the ciphertext:

```
> frequency(vvhq)
```

```
 [ 8, 5, 12, 4, 15, 10, 27, 16, 13, 14, 17, 25,
7, 7, 5, 9, 14, 17,
                  24, 8, 12, 22, 22, 5, 8, 5]
```

Let's compute the coincidences for displacements of 1, 2, 3, 4, 5, 6:

```
> coinc(vvhq,1)
                    14
> coinc(vvhq,2)
                    14
> coinc(vvhq,3)
                    16
> coinc(vvhq,4)
                    14
> coinc(vvhq,5)
                    24
> coinc(vvhq,6)
                    12
```

We conclude that the key length is probably 5. Let's look at the 1st, 6th, 11th, ... letters (namely, the letters in positions congruent to 1 mod 5):

```
> choose(vvhq, 5, 1)

"vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqudukvpk
vggjjivgjggpfncwuce"

> frequency(%)

[0, 0, 7, 1, 1, 2, 9, 0, 1, 8, 8, 0, 0, 3, 0, 4,
5, 2, 0, 3, 6, 5, 1, 0, 1, 0]
```

To express this as a vector of frequencies:

```
> vigvec(vvhq, 5, 1)

[0., 0., .1044776119, .01492537313, .01492537313,
.02985074627, .1343283582, 0., .01492537313,
.1194029851,
.1194029851, 0., 0., .04477611940, 0.,
.05970149254,
```

```
   .07462686567, .02985074627, 0., .04477611940,
   .08955223881,
   .07462686567, .01492537313, 0., .01492537313, 0.]
```

The dot products of this vector with the displacements of the alphabet frequency vector are computed as follows:

```
> corr(%)

   .02501492539, .03910447762, .07132835821,
.03882089552,
   .02749253732, .03801492538, .05120895523,
.03014925374,
   .03247761194, .04302985074, .03377611940,
.02985074628,
   .03426865672, .04456716420, .03555223882,
.04022388058,
   .04343283582, .05017910450, .03917910447,
.02958208957,
   .03262686569, .03917910448, .03655223881,
.03161194031,
   .04883582088, .03494029848
```

The third entry is the maximum, but sometimes the largest entry is hard to locate. One way to find it is

```
> max(%)

                    .07132835821
```

Now it is easy to look through the list and find this number (it usually occurs only once). Since it occurs in the third position, the first shift for this Vigenère cipher is by 2, corresponding to the letter *c*. A procedure similar to the one just used (using `vigvec(vvhq, 5,2),...,vigvec(vvhq,5,5)`) shows that the other shifts are probably 14, 3, 4, 18. Let's check that we have the correct key by decrypting.

```
> vigenere(vvhq, -[2, 14, 3, 4, 18])
```

```
themethodusedforthepreparationandreadingofcodemes
sagesissimpleinthe
extremeandatthesametimeimpossibleoftranslationunl
essthekeyisknownth
eeasewithwhichthekeymaybechangedisanotherpointinf
avoroftheadoptiono
fthiscodebythosedesiringtotransmitimportantmessag
eswithouttheslight
estdangeroftheirmessagesbeingreadbypoliticalorbus
inessrivalsetc
```

For the record, the plaintext was originally encrypted by the command

```
> vigenere(%, [2, 14, 3, 4, 18])

vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

B.4 Examples for Chapter 3

Example 5

Find $\gcd\,(23456, 987654)$.

```
> gcd(23456, 987654)

                          2
```

Example 6

Solve $23456x + 987654y = 2$ in integers x, y.

```
igcdex(23456, 987654,'x','y')

                               2

> x;y

                            -3158
                              75
```

This means that `2` is the gcd and $23456 \cdot (-3158) + 987654 \cdot 75 = 2$. (The command `igcdex` is for *integer gcd extended*. Maple also calculates gcd's for polynomials.) Variable names other than `'x'` and `'y'` can be used if these letters are going to be used elsewhere, for example, in a polynomial. We can also clear the value of `x` as follows:

```
> x:='x'
```

```
x:=x
```

Example 7

Compute $234 \cdot 456 \pmod{789}$.

```
> 234*456 mod 789
                                    189
```

Example 8

Compute $234567^{876543} \pmod{565656565}$.

```
> 234567&^876543 mod 565656565
                              473011223
```

You might need a $\backslash$ before the $\hat{}$. Use the right arrow to escape from the exponent mode.

Example 9

Find the multiplicative inverse of $87878787 \pmod{9191919191}$.

```
> 87878787&^(-1) mod 9191919191
                           7079995354
```

You might need a space before the exponent (-1). (The command `1/87878787 mod 9191919191` also works).

Example 10

Solve $7654x \equiv 2389 \pmod{65537}$.

SOLUTION

Here is one way.

```
> solve(7654*x=2389,x) mod 65537
                                43626
```

Here is another way.

```
> 2389/7654 mod 65537
                                43626
```

The fraction `2389/7654` will appear as a vertically set fraction $\frac{2389}{7654}$. Use the right arrow key to escape from the fraction mode.

Example 11

Find x with

$$x \equiv 2 \pmod{78}, \quad x \equiv 5 \pmod{97}, \quad x \equiv 1 \pmod{119}.$$

```
> chrem([2, 5, 1],[78, 97, 119])
                               647480
```

We can check the answer:

```
> 647480 mod 78; 647480 mod 97; 647480 mod 119
```

```
2
5
1
```

Example 12

Factor 123450 into primes.

```
> ifactor(123450)
```

$$(2)\,(3)(5)^2\,(823)$$

This means that $123450 = 2^1 3^1 5^2 823^1$.

Example 13

Evaluate $\phi(12345)$.

```
> phi(12345)
```

```
6576
```

Example 14

Find a primitive root for the prime 65537.

```
> primroot(65537)
```

```
3
```

Therefore, 3 is a primitive root for 65537.

Example 15

Find the inverse of the matrix

$$\begin{matrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{matrix} \pmod{999}.$$

SOLUTION

First, invert the matrix without the mod, and then reduce the matrix mod 999:

```
> inverse(matrix(3,3,[13, 12, 35, 41, 53, 62, 71,
68, 10]))
```

$$\begin{matrix} \frac{3686}{34139} & -\frac{2260}{34139} & \frac{1111}{34139} \\ -\frac{3992}{34139} & \frac{2355}{34139} & -\frac{629}{34139} \\ \frac{975}{34139} & \frac{32}{34139} & -\frac{197}{34139} \end{matrix}$$

```
> map(x->x mod 999, %)
```

$$\begin{matrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{matrix}$$

This is the inverse matrix mod 999.

Example 16

Find a square root of 26951623672 mod the prime $p = 98573007539$.

SOLUTION

Since $p \equiv 3 \pmod 4$, we can use the proposition of Section 3.9:

```
> 26951623672&^((98573007539 + 1)/4) mod
98573007539

                            98338017685
```

(You need two right arrows to escape from the fraction mode and then the exponent mode.) The other square root is minus the preceding one:

```
> -% mod 98573007539

                           234989854
```

Example 17

Let $n = 34222273 = 9803 \cdot 3491$. Find all four solutions of $x^2 \equiv 19101358 \pmod{34222273}$.

SOLUTION

First, find a square root mod each of the two prime factors, both of which are congruent to $3 \pmod 4$:

```
> 19101358&^((9803 + 1)/4) mod 9803

                                3998

> 19101358&^((3491 + 1)/4) mod 3491

                                1318
```

Therefore, the square roots are congruent to $\pm 3998 \pmod{9803}$ and are congruent to ± 1318

$\pmod{3491}$. There are four ways to combine these using the Chinese remainder theorem:

```
> chrem([3998, 1318],[9803, 3491])
                              43210
> chrem([-3998, 1318],[9803, 3491])
                             8397173
> chrem([3998, -1318],[9803, 3491])
                            25825100
> chrem([-3998, -1318],[9803, 3491])
                            34179063
```

These are the four desired square roots.

B.5 Examples for Chapter 5

Example 18

Compute the first 50 terms of the recurrence

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2.$$

The initial values are $0, 1, 0, 0, 0$.

SOLUTION

The vector of coefficients is $[1, 0, 1, 0, 0]$ and the initial values are given by the vector $[0, 1, 0, 0, 0]$. Type

```
> lfsr([1, 0, 1, 0, 0], [0, 1, 0, 0, 0], 50)
[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
1, 1]
```

Example 19

Suppose the first 20 terms of an LFSR sequence are 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1. Find a recurrence that generates this sequence.

SOLUTION

First, we need to find the length of the recurrence. The command `lfsrlength(v, n)` calculates the determinants mod 2 of the first n matrices that appear in the procedure in Section 5.2:

```
> lfsrlength([1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
1, 1, 0, 1, 0, 1, 0, 1], 10)

                              [1, 1]
                              [2, 1]
                              [3, 0]
                              [4, 1]
                              [5, 0]
                              [6, 1]
                              [7, 0]
                              [8, 0]
                              [9, 0]
                              [10, 0]
```

The last nonzero determinant is the sixth one, so we guess that the recurrence has length 6. To find the coefficients:

```
> lfsrsolve([1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
1, 1, 0, 1, 0, 1, 0, 1], 6)

                        [1, 0, 1, 1, 1, 0]
```

This gives the recurrence as

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2.$$

Example 20

The ciphertext 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0 was produced by adding the output of a LFSR onto the plaintext mod 2 (i.e., XOR the plaintext with the LFSR output). Suppose you know that the plaintext starts 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0. Find the rest of the plaintext.

SOLUTION

XOR the ciphertext with the known part of the plaintext to obtain the beginning of the LFSR output:

```
> [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
0, 0]
  + [0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
0, 1] mod 2

         [1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 1, 0, 1]
```

This is the beginning of the LFSR output. Now let's find the length of the recurrence.

```
> lfsrlength(%, 8)

                              [1, 1]
                              [2, 0]
                              [3, 1]
                              [4, 0]
                              [5, 1]
                              [6, 0]
                              [7, 0]
                              [8, 0]
```

We guess the length is 5. To find the coefficients of the recurrence:

```
> lfsrsolve(%%, 5)

                       [1, 1, 0, 0, 1]
```

Now we can generate the full output of the LFSR using the coefficients we just found plus the first five terms of the LFSR output:

```
> lfsr([1, 1, 0, 0, 1], [1, 0, 0, 1, 0], 40)

[1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
```

```
1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
1, 0, 0, 1, 0, 1, 1, 0]
```

When we XOR the LFSR output with the ciphertext, we get back the plaintext:

```
> % + [0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
0, 1, 0, 0, 0, 1, 0, 1, 1, 0] mod 2

[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 0, 0, 0, 0]
```

This is the plaintext.

B.6 Examples for Chapter 6

Example 21

The ciphertext

$$22, 09, 00, 12, 03, 01, 10, 03, 04, 08, 01, 17$$

was encrypted using a Hill cipher with matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}.$$

Decrypt it.

SOLUTION

There are several ways to input a matrix. One way is the following. A 2×2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ can be entered as `matrix(2,2,[a,b,c,d])`. Type `evalm(M&*N)` to multiply matrices M and N. Type `evalm(v&*M)` to multiply a vector v on the right by a matrix M.

Here is the encryption matrix.

```
> M:=matrix(3,3,[1,2,3,4,5,6,7,8,10])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

We need to invert the matrix mod 26:

```
> invM:=map(x->x mod 26, inverse(M))
```

$$\begin{bmatrix} 8 & 16 & 1 \\ 8 & 21 & 24 \\ 1 & 24 & 1 \end{bmatrix}$$

The command `map(x->x mod 26, E)` takes each number in an expression `E` and reduces it mod 26.

This is the inverse of the matrix mod 26. We can check this as follows:

```
> evalm(M&*invM)
```

$$\begin{bmatrix} 27 & 130 & 52 \\ 78 & 313 & 130 \\ 130 & 520 & 209 \end{bmatrix}$$

```
> map(x->x mod 26, %)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To decrypt, we break the ciphertext into blocks of three numbers and multiply each block on the right by the inverse matrix we just calculated:

```
> map(x->x mod 26, evalm([22,09,00]&*invM))

                          [14, 21, 4]

> map(x->x mod 26, evalm([12,03,01]&*invM))

                          [17, 19, 7]

> map(x->x mod 26, evalm([10,03,04]&*invM))

                           [4, 7, 8]

> map(x->x mod 26, evalm([08,01,17]&*invM))

                          [11, 11, 23]
```

Therefore, the plaintext is 14, 21, 4, 17, 19, 7, 4, 7, 8, 11, 11, 23. Changing this back to letters, we obtain `overthehillx`. Note that the final `x` was appended to the plaintext in order to complete a block of three letters.

B.7 Examples for Chapter 9

Example 22

Suppose you need to find a large random prime of 50 digits. Here is one way. The function `nextprime` finds the next prime greater than x. The function `rand(a..b)()` gives a random integer between a and b. Combining these, we can find a prime:

```
> nextprime(rand(10 ^ 49..10 ^ 50)())

730505700316671091752153033404883134567089132842
91
```

If we repeat this procedure, we should get another prime:

```
> nextprime(rand(10 ^ 49..10 ^ 50)())

974764076949313032557243260405861441453410545683
31
```

Example 23

Suppose you want to change the text *hellohowareyou* to numbers:

```
> text2num("hellohowareyou")

                    805121215081523011805251521
```

Note that we are now using *a = 1, b = 2, ..., z = 26*, since otherwise *a*'s at the beginnings of messages would disappear. (A more efficient procedure would be to work in base 27, so the numerical form of the message would be
$8 + 5 \cdot 27 + 12 \cdot 27^2 + \cdots + 21 \cdot 27^{13} = 87495221502384554951.$
Note that this uses fewer digits.)

Now suppose you want to change it back to letters:

```
> num2text(805121215081523011805251521)

                          "hellohowareyou"
```

Example 24

Encrypt the message `hi` using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the message to numbers:

```
> text2num("hi")

                    809
```

Now, raise it to the *e*th power mod n:

```
> %&^17 mod 823091

                           596912
```

You might need a $\backslash$ before the $\hat{}$. Use the right arrow to escape from the exponent mode.

Example 25

Decrypt the ciphertext in the previous problem.

SOLUTION

First, we need to find the decryption exponent d. To do this, we need to find $\phi(823091)$. One way is

```
> phi(823091)

                  821184
```

Another way is to factor n as $p \cdot q$ and then compute $(p-1)(q-1)$:

```
> ifactor(823091)

                   (659)(1249)

> 658*1248

                     821184
```

Since $de \equiv 1 \pmod{\phi(n)}$, we compute the following (note that we are finding the inverse of e mod $\phi(n)$, not mod n):

```
> 17&^ (-1) mod 821184

                            48305
```

Therefore, $d = 48305$. To decrypt, raise the ciphertext to the dth power mod n:

```
> 596912&^48305 mod 823091
                              809
```

Finally, change back to letters:

```
> num2text(809)

                        "hi"
```

Example 26

Encrypt `hellohowareyou` using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the plaintext to numbers:

```
> text2num("hellohowareyou")

                    805121215081523011805251521
```

Suppose we simply raised this to the eth power mod n:

```
> %&^17 mod 823091

                          447613
```

If we decrypt (we know d from Example 25), we obtain

```
> %&^48305 mod 823091

                         628883
```

This is not the original plaintext. The reason is that the plaintext is larger than n, so we have obtained the plaintext mod n:

```
> 805121215081523011805251521 mod 823091
                                628883
```

We need to break the plaintext into blocks, each less than n. In our case, we use three letters at a time:

$$80512 \quad 121508 \quad 152301 \quad 180525 \quad 1521$$

```
> 80512&^17 mod 823091
                              757396
> 121508&^17 mod 823091
                              164513
> 152301&^17 mod 823091
                              121217
> 180525&^17 mod 823091
                              594220
> 1521&^17 mod 823091
                              442163
```

The ciphertext is therefore 757396164513121217594220442163. Note that there is no reason to change this back to letters. In fact, it doesn't correspond to any text with letters.

Decrypt each block individually:

```
> 757396&^48305 mod 823091
                                80512
> 164513&^48305 mod 823091
                                121508
```

etc.

We'll now do some examples with large numbers, namely the numbers in the RSA Challenge discussed in Section 9.5. These are stored under the names *rsan, rsae, rsap, rsaq*:

```
> rsan

114381625757888867669235779976146612010218296721242362562561842935
706935245733897830597123563958705058989075147599290026879543541

> rsae
                          9007
```

Example 27

Encrypt each of the messages `b, ba, bar, bard` using *rsan* and *rsae*.

```
> text2num("b")&^rsae mod rsan

709467584676126685983701649915507861828763310606852354105647041144
867822617164972001221553323484620140532879875808992631765142534

> text2num("ba")&^rsae mod rsan

350451306089751003250117094498719542737882047539485930603136976982
276217598060279622705380315655647733520336717822613057961581951

> text2num("bar")&^rsae mod rsan

448145128638551010760045308594921093424295316066074090703605434080
008436459868804059531028183128225863625802987844411519226064 24
```

```
> text2num("bard")&^rsae mod rsan

242380777851116664232028625120903173934852129590562707831349916142
560543232971798049289580734457526630264498739868779893299094 98
```

Observe that the ciphertexts are all the same length. There seems to be no easy way to determine the length of the corresponding plaintext.

Example 28

Using the factorization $rsap = rsap \cdot rsap$, find the decryption exponent for the RSA Challenge, and decrypt the ciphertext (see Section 9.5).

First we find the decryption exponent:

```
> rsad:=rsae&^(-1) mod (rsap-1)*(rsaq-1):
```

Note that we use the final colon to avoid printing out the value. If you want to see the value of *rsad*, see Section 9.5, or don't use the colon. To decrypt the ciphertext, which is stored as *rsaci*, and change to letters:

```
> num2text(rsaci&^rsad mod rsan)

           "the magic words are squeamish
ossifrage"
```

Example 29

Encrypt the message `rsaencryptsmessageswell` using *rsan* and *rsae*.

```
> text2num("rsaencryptsmessageswell")&^rsae mod
rsan

946394203490022593163058235392494964146409699340
017097214043524182
719506542543655849060139663288177535392831126531
97553130781884
```

Example 30

Decrypt the preceding ciphertext.

SOLUTION

Fortunately, we know the decryption exponent *rsad*. Therefore, we compute

```
> %&^ rsad mod rsan


1819010514031825162019130519190107051923051212

> num2text(%)

                      "rsaencryptsmessageswell"
```

Suppose we lose the final digit 4 of the ciphertext in transmission. Let's try to decrypt what's left (subtracting 4 and dividing by 10 is a mathematical way to remove the 4):

```
> (%%% - 4)/10)&^rsad mod rsan

479529991731959886649023526295254864091136338943
756298468549079705
884123003734879696577942541171589569212679126284
61494475682806
```

If we try to change this to letters, we do not get anything resembling the message. A small error in the plaintext completely changes the decrypted message and usually produces garbage.

Example 31

Suppose we are told that $n = 11313771275590312567$ is the product of two primes and that $\phi(n) = 11313771187608744400$. Factor n.

SOLUTION

We know (see Section 9.1) that p and q are the roots of $X^2 - (n - \phi(n) + 1)X + n$. Therefore, we compute

```
> solve(x^2 -
(11313771275590312567 - 11313771187608744400 +
1)*x +
11313771275590312567, x)

                      87852787151, 128781017
```

Therefore, $n = 128781017 \cdot 87852787151$. We also could have used the quadratic formula to find the roots.

Example 32

Suppose we know *rsae* and *rsad*. Use these to factor *rsan*.

SOLUTION

We use the $a^r \equiv 1$ factorization method from Section 9.4. First write $rsae \cdot rsad - 1 = 2^s m$ with m odd.

One way to do this is first to compute $rsae \cdot rsad - 1$, and then keep dividing by 2 until we get an odd number:

```
> rsae*rsad - 1

961034419617782266156919023359583834109854129051878330250644604041
155985575087352659156174898557342995131594680431086921245830097664

> %/2

480517209808891133078459511679791917054927064525939165125322302020
577992787543676329578087449278671497565797340215543460622915048832

> %/2
240258604904445566539229755839895958527463532262969582562661151010
288996393771838164789043724639335748782898670107771730311457524416
```

We continue this way for six more steps until we get

```
375404070163196197717546493499837435199161769160889972754158048453
576556865268497132482880819748962107473279172043393328611652381 9
```

This number is m. Now choose a random integer a. Hoping to be lucky, we choose 13. As in the $a^r \equiv 1$ factorization method, we compute

```
> 13&^% mod rsan

275743685070065305922434948688471611984230957073078056905698396470
301831098398623708005293380929847954901926435879608594870551239
```

Since this is not $\pm 1 \pmod{rsan}$, we successively square it until we get ± 1:

```
> %&^2 mod rsan

48318960321928515580138476418723034554104099069940
8462254947027766
54996412582955636035266156108686431194298574075854
037512277292

> %&^2 mod rsan

78172814154877356579141928058754000021948787056483
8209179306251152
15181839742056013275521913487560944732073516487722
273875579363

> %&^2 mod rsan

42836191202508728742199299040582900202976222916017
7671675518702165
09444518239462186379470569442055101392992293082259
601738228702

> %&^2 mod rsan

                                1
```

Since the last number before the 1 was not $\pm 1 \pmod{rsan}$, we have an example of $x \not\equiv \pm 1 \pmod{rsan}$ with $x^2 \equiv 1$. Therefore, $\gcd(x - 1, rsan)$ is a nontrivial factor of *rsan*:

```
> gcd(%% - 1, rsan)

32769132993266709549961988190834461413177642967992942539798288533
```

This is *rsaq*. The other factor is obtained by computing *rsan/rsaq*:

```
rsan/%
```

```
3490529510847650949147849619903898133417764638493
387843990820577
```

This is *rsap*.

Example 33

Suppose you know that

$$150883475569451^2 \equiv 16887570532858^2 \pmod{205611444308117}.$$

Factor 205611444308117.

SOLUTION

We use the Basic Principle of Section 9.4:

```
> gcd(150883475569451-
16887570532858,205611444308117)

                              23495881
```

This gives one factor. The other is

```
> 205611444308117/%

                        8750957
```

We can check that these factors are actually primes, so we can't factor any further:

```
> isprime(%%)

                      true

> isprime(%%)

                      true
```

Example 34

Factor
$n = 376875575426394855599989992897873239$
by the $p - 1$ method.

SOLUTION

Let's choose our bound as $B = 100,$ and let's take
$a = 2,$ so we compute $2^{100!} \pmod{n}$:

```
> 2&^factorial(100)
mod 376875575426394855599989992897873239


369676678301956331939422106251199512
```

Then we compute the gcd of $2^{100!} - 1$ and n:

```
> gcd(% - 1,
376875575426394855599989992897873239)

                        430553161739796481
```

This is a factor p. The other factor q is

```
> 376875575426394855599989992897873239/%

                        875328783798732119
```

Let's see why this worked. The factorizations of $p - 1$
and $q - 1$ are

```
> ifactor(430553161739796481 - 1)

                    (2)^18(3)^7(5)(7)^4(11)^3(47)

> ifactor(875328783798732119 - 1)
```

```
(2)(61)(8967967)(20357)(39301)
```

We see that $100!$ is a multiple of $p-1$, so $2^{100!} \equiv 1 \pmod{p}$. However, $100!$ is not a multiple of $q-1$, so it is likely that $2^{100!} \not\equiv 1 \pmod{q}$. Therefore, both $2^{100!}-1$ and pq have p as a factor, but only pq has q as a factor. It follows that the gcd is p.

B.8 Examples for Chapter 10

Example 35

Let's solve the discrete log problem $2^x \equiv 71 \pmod{131}$ by the Baby Step-Giant Step method of Subsection 10.2.2. We take $N = 12$ since $N^2 > p - 1 = 130$ and we form two lists. The first is $2^j \pmod{131}$ for $0 \leq j \leq 11$:

```
>for j from 0 while j <= 11 do; (j, 2&j mod 131);
end do;

                              0, 1
                              1, 2
                              2, 4
                              3, 8
                              4, 16
                              5, 32
                              6, 64
                              7, 128
                              8, 125
                              9, 119
                              10, 107
                              11, 83
```

The second is $71 \cdot 2^{-j} \pmod{1}31$ for $0 \leq j \leq 11$:

```
> for j from 0 while j <= 11 do; (j, 71*2&\^\:
(-12*j) mod 131); end do;

                               0, 71
                               1, 17
                               2, 124
                               3, 26
                               4, 128
                               5, 86
                               6, 111
                               7, 93
```

```
8, 85
9, 96
10, 130
11, 116
```

The number 128 is on both lists, so we see that $2^7 \equiv 71 \cdot 2^{-12*4} \pmod{131}$. Therefore,

$$71 \equiv 2^{7+4*12} \equiv 2^{55} \pmod{131}.$$

B.9 Examples for Chapter 12

Example 36

Suppose there are 23 people in a room. What is the probability that at least two have the same birthday?

SOLUTION

The probability that no two have the same birthday is $\prod_{i=1}^{22}(1-i/365)$ (note that the product stops at $i=22,$ not $i=23$). Subtracting from 1 gives the probability that at least two have the same birthday:

```
> 1-mul(1.-i/365, i=1..22)
                                .5072972344
```

Note that we used $1.$ in the product instead of 1 without the decimal point. If we had omitted the decimal point, the product would have been evaluated as a rational number (try it, you'll see).

Example 37

Suppose a lazy phone company employee assigns telephone numbers by choosing random seven-digit numbers. In a town with 10,000 phones, what is the probability that two people receive the same number?

```
> 1-mul(1.-i/10^7, i=1..9999)
                                .9932699133
```

Note that the number of phones is about three times the square root of the number of possibilities. This means that we expect the probability to be high, which it is. From Section 12.1, we have the estimate that if there are around $\sqrt{2(\ln 2)10^7} \approx 3723$ phones, there should be a 50% chance of a match. Let's see how accurate this is:

```
> 1-mul(1.-i/10^7, i=1..3722)

                                    .4998945441
```

B.10 Examples for Chapter 17

Example 38

Suppose we have a (5, 8) Shamir secret sharing scheme. Everything is mod the prime $p = 987541$. Five of the shares are

$$(9853, 853),\ (4421, 4387),\ (6543, 1234),\ (93293, 78428),\ (12398, 7563).$$

Find the secret.

SOLUTION

One way: First, find the Lagrange interpolating polynomial through the five points:

```
> interp([9853,4421,6543,93293,12398],
[853,4387,1234,78428,7563],x)
```

$$-\frac{4959003720134640533754 7}{13378864151099487659488222679760000 0}X^4 + \frac{3531308571691925577790733 07}{8919242767399658439658815119840000}X^3 - \frac{8829628978321139771076837361481}{19112663072999268084983175256800000}X^2 + \frac{9749049230474450716950803519811081}{44596213836998292198294075599200000}X + \frac{2044843261540449832301145924339442825 91}{22298106918499146099147037799600000}$$

Now evaluate at $x = 0$ to find the constant term:

```
> eval(%,x=0)
```

$$\frac{2044843261540449832301145924339442825 91}{22298106918499146099147037799600000}$$

We need to change this to an integer mod 987541:

```
> % mod 987541

                    678987
```

Therefore, 678987 is the secret.

Here is another way. Set up the matrix equations as in the text and then solve for the coefficients of the polynomial mod 987541:

```
> map(x->x mod 987541,evalm(inverse(matrix(5,5,
[1,9853,9853^2,9853^3,9853^4,
1,4421,4421^2,4421^3,4421^4,
1,6543,6543^2,6543^3, 6543^4,
1, 93293, 93293^2,93293^3, 93293^4,
1, 12398, 12398^2,12398^3,12398^4]))
&*matrix(5,1,[853,4387,1234,78428,7563])))
```

$$\begin{matrix} 678987 \\ 14728 \\ 1651 \\ 574413 \\ 456741 \end{matrix}$$

The constant term is 678987, which is the secret.

B.11 Examples for Chapter 18

Example 39

Here is a game you can play. It is essentially the simplified version of poker over the telephone from Section 18.2. There are five cards: ten, jack, queen, king, ace. They are shuffled and disguised by raising their numbers to a random exponent mod the prime `24691313099`. You are supposed to guess which one is the ace.

To start, pick a random exponent. We use the colon after `khide()` so that we cannot cheat and see what value of k is being used.

```
> k:= khide():
```

Now, shuffle the disguised cards (their numbers are raised to the kth power mod p and then randomly permuted):

```
> shuffle(k)

   [14001090567, 16098641856, 23340023892,
20919427041, 7768690848]
```

These are the five cards. None looks like the ace that's because their numbers have been raised to powers mod the prime. Make a guess anyway. Let's see if you're correct.

```
> reveal(%)

             ["ten", "ace", "queen", "jack",
"king"]
```

Let's play again:

```
> k:= khide():

> shuffle(k)

   [13015921305, 14788966861, 23855418969,
22566749952, 8361552666]
```

Make your guess (note that the numbers are different because a different random exponent was used). Were you lucky?

```
> reveal(%)

                 ["ten", "queen", "ace", "king",
"jack"]
```

Perhaps you need some help. Let's play one more time:

```
> k:= khide():
> shuffle(k)

   [13471751030, 20108480083, 8636729758,
14735216549, 11884022059]
```

We now ask for advice:

```
> advise(%)

                3
```

We are advised that the third card is the ace. Let's see (recall that %% is used to refer to the next to last output):

```
> reveal(%%)

               ["jack", "ten", "ace", "queen",
"king"]
```

How does this work? Read the part on "How to Cheat" in Section 18.2. Note that if we raise the numbers for the cards to the $(p-1)/2$ power mod p, we get

```
> map(x->x&^((24691313099-1)/2) mod 24691313099,
[200514, 10010311, 1721050514, 11091407, 10305])

                    [1, 1, 1, 1, 24691313098]
```

Therefore, only the ace is a quadratic nonresidue mod p.

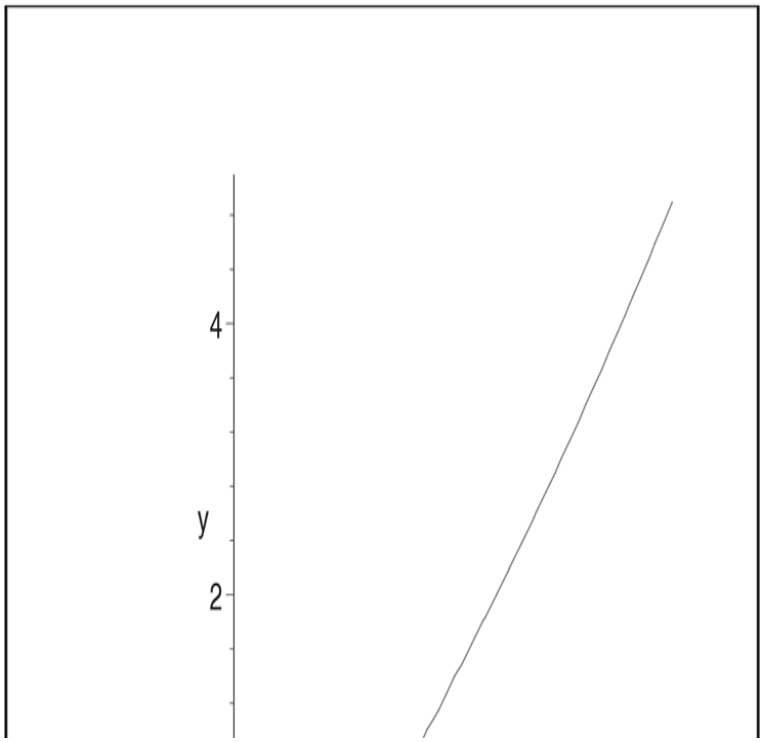
B.12 Examples for Chapter 21

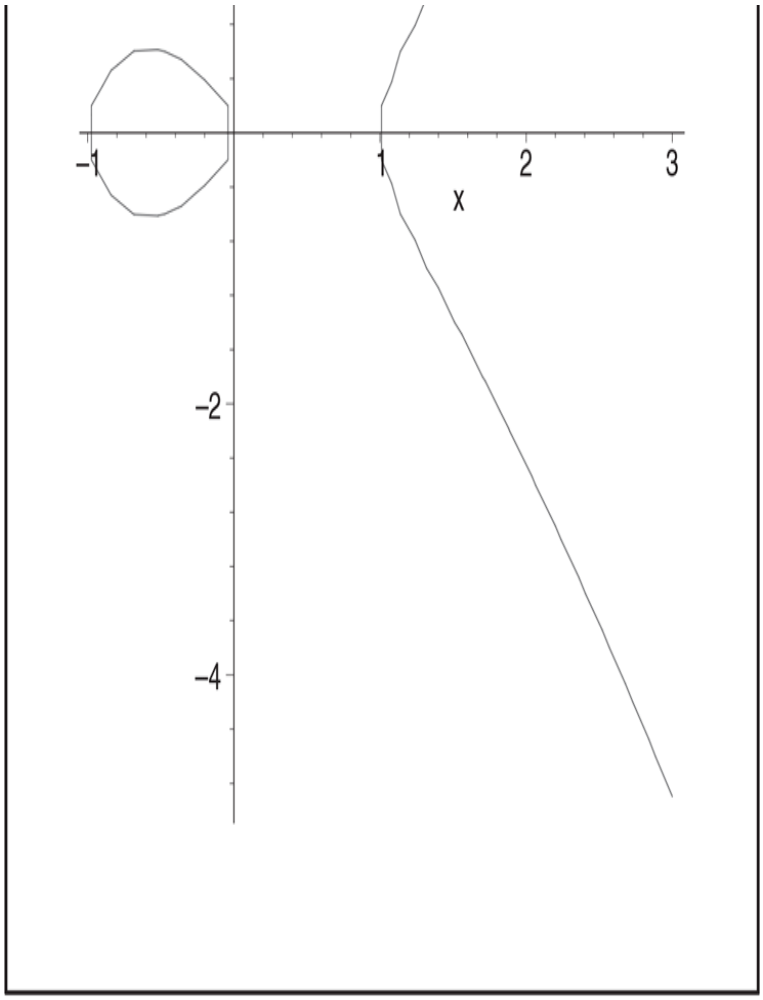
Example 40

All of the elliptic curves we work with in this chapter are elliptic curves mod n. However, it is helpful use the graphs of elliptic curves with real numbers in order to visualize what is happening with the addition law, for example, even though such pictures do not exist mod n.

Let's graph the elliptic curve $y^2 = x(x-1)(x+1)$. We'll specify that $-1 \leq x \leq 3$ and $-5 \leq y \leq 5,$ and make sure that x and y are cleared of previous values.

```
> x:='x';y:='y';implicitplot(y^2=x*(x-1)*(x+1),
x=-1..3,y=-5..5)
```





B.12-1 Full Alternative Text

Example 41

Add the points (1, 3) and (3, 5) on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$.

```
> addell([1,3], [3,5], 24, 13, 29)

                              [26,1]
```

You can check that the point (26, 1) is on the curve: $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$.

Example 42

Add (1, 3) to the point at infinity on the curve of the previous example.

```
> addell([1,3], ["infinity","infinity" ], 24, 13,
29)

                                [1,3]
```

As expected, adding the point at infinity to a point P returns the point P.

Example 43

Let $P = (1, 3)$ be a point on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$. Find $7P$.

```
> multell([1,3], 7, 24, 13, 29)

                                [15,6]
```

Example 44

Find $k(1, 3)$ for $k = 1, 2, 3, \ \ldots\ , 40$ on the curve of the previous example.

```
> multsell([1,3], 40, 24, 13, 29)

[[1,[1,3]],[2,[11,10]],[3,[23,28]],[4,[0,10]],[5,
[19,7]],[6,[18,19]],
[7,[15,6]],[8,[20,24]],[9,[4,12]],[10,[4,17]],
[11,[20,5]],
[12,[15,23]],[13,[18,10]],[14,[19,22]],[15,
[0,19]],[16,[23,1]],
[17,[11,19]],[18,[1,26]],[19,
["infinity","infinity"]],[20,[1,3]],
```

```
[21,[11,10]],[22,[23,28]],[23,[0,10]],[24,
[19,7]],[25,[18,19]],
[26,[15,6]],[27,[20,24]],[28,[4,12]],[29,[4,17]],
[30,[20,5]],
[31,[15,23]],[32,[18,10]],[33,[19,22]],[34,
[0,19]],[35,[23,1]],
[36,[11,19]],[37,[1,26]],[38,
["infinity","infinity"]],[39,[1,3]],
[40,[11,10]]]
```

Notice how the points repeat after every 19 multiples.

Example 45

The previous four examples worked mod the prime 29. If we work mod a composite number, the situation at infinity becomes more complicated since we could be at infinity mod both factors or we could be at infinity mod one of the factors but not mod the other. Therefore, we stop the calculation if this last situation happens and we exhibit a factor. For example, let's try to compute $12P$, where $P = (1, 3)$ is on the elliptic curve $y^2 \equiv x^3 - 5x + 13 \pmod{209}$:

```
> multell([1,3], 12, -5, 13, 11*19)

                          ["factor=",19]
```

Now let's compute the successive multiples to see what happened along the way:

```
> multsell([1,3], 12, -5, 13, 11*19)

[[1,[1,3]],[2,[91,27]],[3,[118,133]],[4,
[148,182]],[5,[20,35]],
                     [6,["factor=",19]]]
```

When we computed $6P$, we ended up at infinity mod 19. Let's see what is happening mod the two prime factors of

209, namely 19 and 11:

```
> multsell([1,3], 12, -5, 13, 19)

[[1,[1,3]],[2,[15,8]],[3,[4,0]],[4,[15,11]],[5,
[1,16]],
[6,["infinity","infinity"]],[7,[1,3]],[8,[15,8]],
[9,[4,0]],
[10,[15,11]],[11,[1,16]],[12,
["infinity","infinity"]]]

> multsell([1,3], 24, -5, 13, 11)

[[1,[1,3]],[2,[3,5]],[3,[8,1]],[4,[5,6]],[5,
[9,2]],[6,[6,10]],
[7,[2,0]],[8,[6,1]],[9,[9,9]],[10,[5,5]],[11,
[8,10]],[12,[3,6]],
[13,[1,8]],[14,["infinity","infinity"]],[15,
[1,3]],[16,[3,5]],
[17,[8,1]],[18,[5, 6]],[19,[9, 2]],[20,[6,10]],
[21,[2,0]],
[22,[6,1]],[23,[9,9]],[24,[5,5]]]
```

After six steps, we were at infinity mod 19, but it takes 14 steps to reach infinity mod 11. To find $6P$, we needed to invert a number that was 0 mod 19 and nonzero mod 11. This couldn't be done, but it yielded the factor 19. This is the basis of the elliptic curve factorization method.

Example 46

Factor 193279 using elliptic curves.

SOLUTION

First, we need to choose some random elliptic curves and a point on each curve. For example, let's take $P = (2, 4)$ and the elliptic curve

$$y^2 \equiv x^3 - 10x + b \pmod{193279}.$$

For P to lie on the curve, we take $b = 28$. We'll also take

$$y^2 \equiv x^3 + 11x - 11, \quad P = (1, 1)$$
$$y^2 \equiv x^3 + 17x - 14, \quad P = (1, 2).$$

Now we compute multiples of the point P. We do the analog of the $p - 1$ method, so we choose a bound B, say $B = 12,$ and compute $B!P$.

```
> multell([2,4], factorial(12), -10, 28, 193279)

                          ["factor=",347]

> multell([1,1], factorial(12), 11, -11, 193279)

                           [13862,35249]

> multell([1,2], factorial(12), 17, -14, 193279)

                          ["factor=",557]
```

Let's analyze in more detail what happened in these examples.

On the first curve, $266P$ ends up at infinity mod 557 and $35P$ is infinity mod 347. Since $266 = 2 \cdot 7 \cdot 19,$ it has a prime factor larger than $B = 12,$ so $B!P$ is not infinity mod 557. But 35 divides $B!,$ so $B!P$ is infinity mod 347.

On the second curve, $356P = \text{infinity}$ mod 347 and $561P = \text{infinity}$ mod 557. Since $356 = 4 \cdot 89$ and $561 = 3 \cdot 11 \cdot 17,$ we don't expect to find the factorization with this curve.

The third curve is a surprise. We have $331P = \text{infinity}$ mod 347 and $272P = \text{infinity}$ mod 557. Since 331 is prime and $272 = 16 \cdot 17,$ we don't expect to find the factorization with this curve. However, by chance, an intermediate step in the calculation of $B!P$ yielded the factorization. Here's what happened. At one step, the program required adding the points (184993, 13462) and (20678, 150484). These two points are congruent mod 557 but not mod 347. Therefore, the slope of the line

through these two points is defined mod 347 but is 0/0 mod 557. When we tried to find the multiplicative inverse of the denominator mod 193279, the gcd algorithm yielded the factor 557. This phenomenon is fairly rare.

Example 47

Here is how to produce the example of an elliptic curve ElGamal cryptosystem from Section 21.5. For more details, see the text. The elliptic curve is $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$ and the point is $G = (4, 11)$. Alice's message is the point $P_m = (5, 1743)$.

Bob has chosen his secret random number $a_B = 3$ and has computed $a_b G$:

```
> multell([4,11], 3, 3, 45, 8831)

                    [413,1808]
```

Bob publishes this point. Alice chooses the random number $k = 8$ and computes kG and $P_m + k(a_B G)$:

```
> multell([4,11], 8, 3, 45, 8831)

                           [5415,6321]

>
addell([5,1743],multell([413,1808],8,3,45,8831),3
,45,8831)

                           [6626,3576]
```

Alice sends (5415,6321) and (6626,3576) to Bob, who multiplies the first of these point by a_B:

```
> multell([5415,6321], 3, 3, 45, 8831)
                                [673,146]
```

Bob then subtracts the result from the last point Alice sends him. Note that he subtracts by adding the point with the second coordinate negated:

```
> addell([6626,3576], [673,-146], 3, 45, 8831)
                                [5,1743]
```

Bob has therefore received Alice's message.

Example 48

Let's reproduce the numbers in the example of a Diffie-Hellman key exchange from Section 21.5: The elliptic curve is $y^2 \equiv x^3 + x + 7206 \pmod{7211}$ and the point is $G = (3, 5)$. Alice chooses her secret $N_A = 12$ and Bob chooses his secret $N_B = 23$. Alice calculates

```
> multell([3,5], 12, 1, 7206, 7211)
                              [1794,6375]
```

She sends (1794,6375) to Bob. Meanwhile, Bob calculates

```
> multell([3,5], 23, 1, 7206, 7211)
                              [3861, 1242]
```

and sends (3861,1242) to Alice. Alice multiplies what she receives by N_A and Bob multiplies what he receives by N_B:

```
> multell([3861,1242], 12, 1, 7206, 7211)
                            [1472,2098]
> multell([1794,6375], 23, 1, 7206, 7211)
                            [1472,2098]
```

Therefore, Alice and Bob have produced the same key.

Appendix C MATLAB® Examples

These computer examples are written for MATLAB. If you have MATLAB available, you should try some of them on your computer. For information on getting started with MATLAB, see Section C.1. Several functions have been written to allow for experimentation with MATLAB. The MATLAB functions associated with this book are available at

```
bit.ly/2HyvR8n
```

We recommend that you create a directory or folder to store these files and download them to that directory or folder. One method for using these functions is to launch MATLAB from the directory where the files are stored, or launch MATLAB and change the current directory to where the files are stored. In some versions of MATLAB the working directory can be changed by changing the current directory on the command bar. Alternatively, one can add the path to that directory in the MATLAB path by using the *path* function or the Set Path option from the File menu on the command bar.

If MATLAB is not available, it is still possible to read the examples. They provide examples for several of the concepts presented in the book. Most of the examples used in the MATLAB appendix are similar to the examples in the Mathematica and Maple appendices. MATLAB, however, is limited in the size of the numbers it can handle. The maximum number that MATLAB can represent in its default mode is roughly 16 digits and larger numbers are approximated. Therefore, it is

necessary to use the symbolic mode in MATLAB for some of the examples used in this book.

A final note before we begin. It may be useful when doing the MATLAB exercises to change the formatting of your display. The command

```
>> format rat
```

sets the formatting to represent numbers using a fractional representation. The conventional *short* format represents large numbers in scientific notation, which often doesn't display some of the least significant digits. However, in both formats, the calculations, when not in symbolic mode, are done in floating point decimals, and then the rational format changes the answers to rational numbers approximating these decimals.

C.1 Getting Started with MATLAB

MATLAB is a programming language for performing technical computations. It is a powerful language that has become very popular and is rapidly becoming a standard instructional language for courses in mathematics, science, and engineering. MATLAB is available on most campuses, and many universities have site licenses allowing MATLAB to be installed on any machine on campus.

In order to launch MATLAB on a PC, double click on the MATLAB icon. If you want to run MATLAB on a Unix system, type *matlab* at the prompt. Upon launching MATLAB, you will see the MATLAB prompt:

```
>>
```

which indicates that MATLAB is waiting for a command for you to type in. When you wish to quit MATLAB, type *quit* at the command prompt.

MATLAB is able to do the basic arithmetic operations such as addition, subtraction, multiplication, and division. These can be accomplished by the operators +, -, *, and /, respectively. In order to raise a number to a power, we use the operator ^ . Let us look at an example:

If we type $2 \hat{} 7 + 125/5$ at the prompt and press the *Enter* key

```
>> 2^7 + 125/5
```

then MATLAB will return the answer:

```
ans =
   153
```

Notice that in this example, MATLAB performed the exponentiation first, the division next, and then added the two results. The order of operations used in MATLAB is the one that we have grown up using. We can also use parentheses to change the order in which MATLAB calculates its quantities. The following example exhibits this:

```
>> 11*( (128/(9+7) - 2^(72/12)))

ans =
   -616
```

In these examples, MATLAB has called the result of the calculations *ans*, which is a variable that is used by MATLAB to store the output of a computation. It is possible to assign the result of a computation to a specific variable. For example,

```
>> spot=17

spot =
   17
```

assigns the value of 17 to the variable *spot*. It is possible to use variables in computations:

```
>> dog=11

dog =
   11

>> cat=7

cat =
   7

>> animals=dog+cat
```

```
animals =
   18
```

MATLAB also operates like an advanced scientific calculator since it has many functions available to it. For example, we can do the standard operation of taking a square root by using the *sqrt* function, as in the following example:

```
>> sqrt(1024)

ans =
   32
```

There are many other functions available. Some functions that will be useful for this book are *mod, factorial, factor, prod*, and *size*.

Help is available in MATLAB. You may either type *help* at the prompt, or pull down the Help menu. MATLAB also provides help from the command line by typing *help commandname*. For example, to get help on the function *mod*, which we shall be using a lot, type the following:

```
>> help mod
```

MATLAB has a collection of toolboxes available. The toolboxes consist of collections of functions that implement many application-specific tasks. For example, the Optimization toolbox provides a collection of functions that do linear and nonlinear optimization. Generally, not all toolboxes are available. However, for our purposes, this is not a problem since we will only need general MATLAB functions and have built our own functions to explore the number theory behind cryptography.

The basic data type used in MATLAB is the matrix. The MATLAB programming language has been written to use

matrices and vectors as the most fundamental data type. This is natural since many mathematical and scientific problems lend themselves to using matrices and vectors.

Let us start by giving an example of how one enters a matrix in MATLAB. Suppose we wish to enter the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}$$

into MATLAB. To do this we type:

```
>> A = [1 1 1 1; 1 2 4 8; 1 3 9 27; 1 4 16 64]
```

at the prompt. MATLAB returns

```
A =
     1  1    1     1
     1  2    4     8
     1  3    9    27
     1  4   16    64
```

There are a few basic rules that are used when entering matrices or vectors. First, a vector or matrix is started by using a square bracket [and ended using a square bracket]. Next, blanks or commas separate the elements of a row. A semicolon is used to end each row. Finally, we may place a semicolon at the very end to prevent MATLAB from displaying the output of the command.

To define a row vector, use blanks or commas. For example,

```
>> x = [2, 4, 6, 8, 10, 12]

x =
     2   4   6   8   10   12
```

To define a column vector, use semicolons. For example,

```
>> y=[1;3;5;7]

y =
   1
   3
   5
   7
```

In order to access a particular element of y, put the desired index in parentheses. For example, y(1) = 1, y(2) = 3, and so on.

MATLAB provides a useful notation for addressing multiple elements at the same time. For example, to access the third, fourth, and fifth elements of x, we would type

```
>> x(3:5)

ans =
    6   8   10
```

The 3:5 tells MATLAB to start at 3 and count up to 5. To access every second element of x, you can do this by

```
>> x(1:2:6)

ans =
    2   6   10
```

We may do this for the array also. For example,

```
>> A(1:2:4,2:2:4)

ans =
    1   1
    3  27
```

The notation 1:n may also be used to assign to a variable. For example,

```
>> x=1:7
```

returns

```
x =
    1   2   3   4   5   6   7
```

MATLAB provides the *size* function to determine the dimensions of a vector or matrix variable. For example, if we want the dimensions of the matrix A that we entered earlier, then we would do

```
>> size(A)

ans =
    4   4
```

It is often necessary to display numbers in different formats. MATLAB provides several output formats for displaying the result of a computation. To find a list of formats available, type

```
>> help format
```

The *short* format is the default format and is very convenient for doing many computations. However, in this book, we will be representing long whole numbers, and the *short* format will cut off some of the trailing digits in a number. For example,

```
>> a=1234567899

a =
   1.2346e+009
```

Instead of using the *short* format, we shall use the *rational* format. To switch MATLAB to using the rational format, type

```
>> format rat
```

As an example, if we do the same example as before, we now get different results:

```
>> a=1234567899

a =
   1234567899
```

This format is also useful because it allows us to represent fractions in their fractional form, for example,

```
>> 111/323

ans =
   111/323
```

In many situations, it will be convenient to suppress the results of a computation. In order to have MATLAB suppress printing out the results of a command, a semicolon must follow the command. Also, multiple commands may be entered on the same line by separating them with commas. For example,

```
>> dogs=11, cats=7; elephants=3, zebras=19;

dogs =
   11

elephants =
   3
```

returns the values for the variables *dogs* and *elephants* but does not display the values for *cats* and *zebras*.

MATLAB can also handle variables that are made of text. A string is treated as an array of characters. To assign a string to a variable, enclose the text with single quotes. For example,

```
>> txt='How are you today?'
```

returns

```
txt =
   How are you today?
```

A string has size much like a vector does. For example, the size of the variable txt is given by

```
>> size(txt)
ans =
    1   18
```

It is possible to edit the characters one by one. For example, the following command changes the first word of txt:

```
>> txt(1)='W'; txt(2)='h';txt(3)='o'

txt =
   Who are you today?
```

As you work in MATLAB, it will remember the commands you have entered as well as the values of the variables you have created. To scroll through your previous commands, press the up-arrow and down-arrow. In order to see the variables you have created, type *who* at the prompt. A similar command *whos* gives the variables, their size, and their type information.

Notes. 1. To use the commands that have been written for the examples, you should run MATLAB in the

directory into which you have downloaded the file from the Web site `bit.ly/2HyvR8n`

2. Some of the examples and computer problems use long ciphertexts, etc. For convenience, these have been stored in the file `ciphertexts.m`, which can be loaded by typing *ciphertexts* at the prompt. The ciphertexts can then be referred to by their names. For example, see Computer Example 4 for Chapter 2.

C.2 Examples for Chapter 2

Example 1

A shift cipher was used to obtain the ciphertext `kddkmu`.

Decrypt it by trying all possibilities.

```
>> allshift('kddkmu')


kddkmu
leelnv
mffmow
nggnpx
ohhoqy
piiprz
qjjqsa
rkkrtb
sllsuc
tmmtvd
unnuwe
voovxf
wppwyg
xqqxzh
yrryai
zsszbj
attack
buubdl
cvvcem
dwwdfn
exxego
fyyfhp
gzzgiq
haahjr
ibbiks
jccjlt
```

As you can see, `attack` is the only word that occurs on this list, so that was the plaintext.

Example 2

Encrypt the plaintext message `cleopatra` using the affine function $7x + 8$:

```
>> affinecrypt('cleopatra',7,8)

ans =


'whkcjilxi'
```

Example 3

The ciphertext `mzdvezc` was encrypted using the affine function $5x + 12$. Decrypt it.

SOLUTION

First, solve $y \equiv 5x + 12 \pmod{26}$ for x to obtain $x \equiv 5^{-1}(y - 12)$. We need to find the inverse of $5 \pmod{26}$:

```
>> powermod(5,-1,26)

ans =
   21
```

Therefore, $x \equiv 21(y - 12) \equiv 21y - 12 \cdot 21$. To change $-12 \cdot 21$ to standard form:

```
>> mod(-12*21,26)

ans =
   8
```

Therefore, the decryption function is $x \equiv 21y + 8$. To decrypt the message:

```
>> affinecrypt('mzdvezc',21,8)

ans =


'anthony'
```

In case you were wondering, the plaintext was encrypted as follows:

```
>> affinecrypt('anthony',5,12)

ans =


'mzdvezc'
```

Example 4

Here is the example of a Vigenère cipher from the text. Let's see how to produce the data that was used in Section 2.3 to decrypt the ciphertext. In the file ciphertexts.m, the ciphertext is stored under the name *vvhq*. If you haven't already done so, load the file ciphertexts.m:

```
>> ciphertexts
```

Now we can use the variable vvhq to obtain the ciphertext:

```
>> vvhq

vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
```

```
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

We now find the frequencies of the letters in the ciphertext. We use the function *frequency*. The *frequency* command was written to display automatically the letter and the count next to it. We therefore have put a semicolon at the end of the command to prevent MATLAB from displaying the count twice.

```
>> fr=frequency(vvhq);
 a   8
 b   5
 c  12
 d   4
 e  15
 f  10
 g  27
 h  16
 i  13
 j  14
 k  17
 l  25
 m   7
 n   7
 o   5
 p   9
 q  14
 r  17
 s  24
 t   8
 u  12
 v  22
 w  22
 x   5
 y   8
 z   5
```

Let's compute the coincidences for displacements of 1, 2, 3, 4, 5, 6:

```
>> coinc(vvhq,1)

ans =
   14
```

```
>> coinc(vvhq,2)

ans =
   14

>> coinc(vvhq,3)

ans =
   16

>> coinc(vvhq,4)

ans =
   14

>> coinc(vvhq,5)

ans =
   24

>> coinc(vvhq,6)

ans =
   12
```

We conclude that the key length is probably 5. Let's look at the 1st, 6th, 11th, ... letters (namely, the letters in positions congruent to 1 mod 5). The function *choose* will do this for us. The function *choose(txt,m,n)* extracts every letter from the string txt that has positions congruent to n mod m.

```
>> choose(vvhq,5,1)

ans =

vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqudukvpkv
ggjjivgjgg pfncwuce
```

We now do a frequency count of the preceding substring. To do this, we use the *frequency* function and use ans as input. In MATLAB, if a command is issued without declaring a variable for the result, MATLAB will put the output in the variable ans.

```
>> frequency(ans);


 a   0
 b   0
 c   7
 d   1
 e   1
 f   2
 g   9
 h   0
 i   1
 j   8
 k   8
 l   0
 m   0
 n   3
 o   0
 p   4
 q   5
 r   2
 s   0
 t   3
 u   6
 v   5
 w   1
 x   0
 y   1
 z   0
```

To express this as a vector of frequencies, we use the *vigvec* function. The *vigvec* function will not only display the frequency counts just shown, but will return a vector that contains the frequencies. In the following output, we have suppressed the table of frequency counts since they appear above and have reported the results in the *short* format.

```
>> vigvec(vvhq,5,1)

ans =
   0
   0
   0.1045
   0.0149
   0.0149
   0.0299
   0.1343
```

```
    0
    0.0149
    0.1194
    0.1194
    0
    0
    0.0448
    0
    0.0597
    0.0746
    0.0299
    0
    0.0448
    0.0896
    0.0746
    0.0149
    0
    0.0149
    0
```

(If we are working in rational format, these numbers are displayed as rationals.) The dot products of this vector with the displacements of the alphabet frequency vector are computed as follows:

```
>> corr(ans)

ans =
   0.0250
   0.0391
   0.0713
   0.0388
   0.0275
   0.0380
   0.0512
   0.0301
   0.0325
   0.0430
   0.0338
   0.0299
   0.0343
   0.0446
   0.0356
   0.0402
   0.0434
   0.0502
   0.0392
   0.0296
   0.0326
   0.0392
```

```
    0.0366
    0.0316
    0.0488
    0.0349
```

The third entry is the maximum, but sometimes the largest entry is hard to locate. One way to find it is

```
>> max(ans)

ans =
   0.0713
```

Now it is easy to look through the list and find this number (it usually occurs only once). Since it occurs in the third position, the first shift for this Vigenère cipher is by 2, corresponding to the letter *c*. A procedure similar to the one just used (using *vigvec(vvhq, 5,2), . . . , vigvec(vvhq,5,5)*) shows that the other shifts are probably 14, 3, 4, 18. Let's check that we have the correct key by decrypting.

```
>> vigenere(vvhq,-[2,14,3,4,18])

ans =

themethodusedforthepreparationandreadingofcodemes
sagesissimpleinthe
extremeandatthesametimeimpossibleoftranslationunl
essthekeyisknownth
eeasewithwhichthekeymaybechangedisanotherpointinf
avoroftheadoptiono
fthiscodebythosedesiringtotransmitimportantmessag
eswithouttheslight
estdangeroftheirmessagesbeingreadbypoliticalorbus
inessrivalsetc
```

For the record, the plaintext was originally encrypted by the command

```
>> vigenere(ans,[2,14,3,4,18])

ans =
```

```
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahw
kcuhwauglqhnslrljs
hbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfn
svwljsniqkgnrgybwl
wgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrclrlkbi
enqfrjljsdhgrhlsfq
twlauqrhwdmwlgusgikkflryvcwvspgpmlkassjvoqxeggvey
ggzmljcxxljsvpaivw
ikvrdrygfrjljslveggveyggeiapuuisfpbtgnwwmuczrvtwg
lrwugumnczvile
```

C.3 Examples for Chapter 3

Example 5

Find $\gcd(23456; 987654)$.

```
>> gcd(23456,987654)

ans =
   2
```

If larger integers are used, they should be expressed in symbolic mode; otherwise, only the first 16 digits of the entries are used accurately. The present calculation could have been done as

```
>> gcd(sym('23456'),sym('987654'))

ans =
   2
```

Example 6

Solve $23456x + 987654y = 2$ in integers x, y.

```
>> [a,b,c]=gcd(23456,987654)

a =
   2

b =
   -3158

c =
   75
```

This means that 2 is the gcd and
$23456 \cdot (-3158) + 987654 \cdot 75 = 2$.

Example 7

Compute $234 \cdot 456 \pmod{789}$.

```
>> mod(234*456,789)

ans =
   189
```

Example 8

Compute $234567^{876543} \pmod{565656565}$.

```
>>
powermod(sym('234567'),sym('876543'),sym('5656565
65'))

ans =
   5334
```

Example 9

Find the multiplicative inverse of
$87878787 \pmod{9191919191}$.

```
>> invmodn(sym('87878787'),sym('9191919191'))

ans =
   7079995354
```

Example 10

Solve $7654x \equiv 2389 \pmod{65537}$.

SOLUTION

To solve this problem, we follow the method described in Section 3.3. We calculate 7654^{-1} and then multiply it by 2389:

```
>> invmodn(7654,65537)

ans =
   54637

>> mod(ans*2389,65537)

ans =
   43626
```

Example 11

Find x with

$$x \equiv 2 \pmod{78}, \quad x \equiv 5 \pmod{97}, \quad x \equiv 1 \pmod{119}.$$

SOLUTION

To solve the problem we use the function *crt*.

```
>> crt([2 5 1],[78 97 119])

ans =
   647480
```

We can check the answer:

```
>> mod(647480,[78 97 119])

ans =
    2   5   1
```

Example 12

Factor 123450 into primes.

```
>> factor(123450)

ans =
    2    3    5    5   823
```

This means that $123450 = 2^1 3^1 5^2 823^1$.

Example 13

Evaluate $\phi(12345)$.

```
>> eulerphi(12345)

ans =
   6576
```

Example 14

Find a primitive root for the prime 65537.

```
>> primitiveroot(65537)

ans =
   3
```

Therefore, 3 is a primitive root for 65537.

Example 15

Find the inverse of the matrix

$$\begin{pmatrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{pmatrix} \pmod{999}.$$

SOLUTION

First, we enter the matrix as M.

```
>> M=[13 12 35; 41 53 62; 71 68 10];
```

Next, invert the matrix without the mod:

```
>> Minv=inv(M)

Minv =
     233/2158     -539/8142    103/3165
     -270/2309    139/2015     -40/2171
     209/7318     32/34139    -197/34139
```

We need to multiply by the determinant of M in order to clear the fractions out of the numbers in $Minv$. Then we need to multiply by the inverse of the determinant mod 999.

```
>> Mdet=det(M)

Mdet =
   -34139

>> invmodn(Mdet,999)

ans =
   589
```

The answer is given by

```
>> mod(Minv*589*Mdet,999)

ans =
    772    472    965
```

```
      641     516     851
      150     133     149
```

Therefore, the inverse matrix mod 999 is

$$\begin{pmatrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{pmatrix}.$$

In many cases, it is possible to determine by inspection the common denominator that must be removed. When this is not the case, note that the determinant of the original matrix will always work as a common denominator.

Example 16

Find a square root of 26951623672 mod the prime $p = 98573007539$.

SOLUTION

Since $p \equiv 3 \pmod 4$, we can use the proposition of Section 3.9:

```
>> powermod(sym('26951623672'),
(sym('98573007539')+1)/4,sym('98573007539'))

ans =
   98338017685
```

The other square root is minus this one:

```
>> mod(-ans,32579)

ans =
   234989854
```

Example 17

Let $n = 34222273 = 9803 \cdot 3491$. Find all four solutions of $x^2 \equiv 19101358 \pmod{34222273}$.

SOLUTION

First, find a square root mod each of the two prime factors, both of which are congruent to $3 \pmod 4$:

```
>> powermod(19101358,(9803+1)/4,9803)

ans =
   3998
>> powermod(19101358,(3491+1)/4,3491)

ans =
   1318
```

Therefore, the square roots are congruent to $\pm 3998 \pmod{9803}$ and are congruent to $\pm 1318 \pmod{3491}$. There are four ways to combine these using the Chinese remainder theorem:

```
>> crt([3998 1318],[9803 3491])

ans =
   43210

>> crt([-3998 1318],[9803 3491])

ans =
   8397173

>> crt([3998 -1318],[9803 3491])

ans =
   25825100

>> crt([-3998 -1318],[9803 3491])

ans =
   34179063
```

These are the four desired square roots.

C.4 Examples for Chapter 5

Example 18

Compute the first 50 terms of the recurrence

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2.$$

The initial values are $0, 1, 0, 0, 0$.

SOLUTION

The vector of coefficients is $[1, 0, 1, 0, 0]$ and the initial values are given by the vector $[0, 1, 0, 0, 0]$. Type

```
>> lfsr([1 0 1 0 0],[0 1 0 0 0],50)

ans =
   Columns 1 through 12
    0   1   0   0   0   0   1   0   0   1   0   1
   Columns 13 through 24
    1   0   0   1   1   1   1   1   0   0   0   1
   Columns 25 through 36
    1   0   1   1   1   0   1   0   1   0   0   0
   Columns 37 through 48
    0   1   0   0   1   0   1   1   0   0   1   1
   Columns 49 through 50
    1   1
```

Example 19

Suppose the first 20 terms of an LFSR sequence are 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1. Find a recursion that generates this sequence.

SOLUTION

First, we find a candidate for the length of the recurrence. The command *lfsrlength(v, n)* calculates the determinants mod 2 of the first n matrices that appear in the procedure described in Section 5.2 for the sequence v. Recall that the last nonzero determinant gives the length of the recurrence.

```
>> lfsrlength([1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0
1 0 1],10)
    Order Determinant
       1        1
       2        1
       3        0
       4        1
       5        0
       6        1
       7        0
       8        0
       9        0
      10        0
```

The last nonzero determinant is the sixth one, so we guess that the recurrence has length 6. To find the coefficients:

```
>> lfsrsolve([1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1
0 1],6)

ans =
    1   0   1   1   1   0
```

This gives the recurrence as

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2.$$

Example 20

The ciphertext 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0 was produced by adding the output of a LFSR onto the plaintext mod 2 (i.e., XOR the plaintext with the LFSR

output). Suppose you know that the plaintext starts 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0. Find the rest of the plaintext.

SOLUTION

XOR the ciphertext with the known part of the plaintext to obtain the beginning of the LFSR output:

```
>> x=mod([1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0]+[0 1
1 0 1 0 1 0 1 0 0 1 1 0 0 0 1],2)

x =
   Columns 1 through 12
    1   0   0   1   0   1   1   0   1   0   0   1
   Columns 13 through 17
    0   1   1   0   1
```

This is the beginning of the LFSR output. Let's find the length of the recurrence:

```
>> lfsrlength(x,8)
    Order  Determinant
      1         1
      2         0
      3         1
      4         0
      5         1
      6         0
      7         0
      8         0
```

We guess the length is 5. To find the coefficients of the recurrence:

```
>> lfsrsolve(x,5)

ans =
    1   1   0   0   1
```

Now we can generate the full output of the LFSR using the coefficients we just found plus the first five terms of the LFSR output:

```
>> lfsr([1 1 0 0 1],[1 0 0 1 0],40)

ans =
   Columns 1 through 12
    1   0   0   1   0   1   1   0   1   0   0   1
   Columns 13 through 24
    0   1   1   0   1   0   0   1   0   1   1   0
   Columns 25 through 36
    1   0   0   1   0   1   1   0   1   0   0   1
   Columns 37 through 40
    0   1   1   0
```

When we XOR the LFSR output with the ciphertext, we get back the plaintext:

```
>> mod(ans+[0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0],2)

ans =
   Columns 1 through 12
    1   1   1   1   1   1   0   0   0   0   0   0
   Columns 13 through 24
    1   1   1   0   0   0   1   1   1   1   0   0
   Columns 25 through 36
    0   0   1   1   1   1   1   1   1   0   0   0
   Columns 37 through 40
    0   0   0   0
```

This is the plaintext.

C.5 Examples for Chapter 6

Example 21

The ciphertext

$$22, 09, 00, 12, 03, 01, 10, 03, 04, 08, 01, 17$$

was encrypted using a Hill cipher with matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}.$$

Decrypt it.

SOLUTION

A matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is entered as $[a, b; c, d]$. Type $M * N$ to multiply matrices M and N. Type $v * M$ to multiply a vector v on the right by a matrix M.

First, we put the above matrix in the variable M.

```
>> M=[1 2 3; 4 5 6; 7 8 10]

M =
```

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{matrix}$$

Next, we need to invert the matrix mod 26:

```
>> Minv=inv(M)

   Minv =
```

$$\begin{matrix} -2/3 & -4/3 & 1 \\ -2/3 & 11/3 & -2 \\ 1 & -2 & 1 \end{matrix}$$

Since we are working mod 26, we can't stop with numbers like $2/3$. We need to get rid of the denominators and reduce mod 26. To do so, we multiply by 3 to extract the numerators of the fractions, then multiply by the inverse of 3 mod 26 to put the "denominators" back in (see Section 3.3):

```
>> M1=Minv*3

M1 =
```

$$\begin{matrix} -2 & -4 & 3 \\ -2 & 11 & -6 \\ 3 & -6 & 3 \end{matrix}$$

```
>> M2=mod(round(M1*9,26))

M2 =
```

$$\begin{matrix} 8 & 16 & 1 \\ 8 & 21 & 24 \\ 1 & 24 & 1 \end{matrix}$$

Note that we used the function *round* in calculating $M2$. This was done since MATLAB performs its calculations in floating point and calculating the inverse matrix $Minv$ produces numbers that are slightly different from whole numbers. For example, consider the following:

```
>> a=1.99999999;display([a, mod(a,2),
mod(round(a),2)])
```

```
2.0000 2.0000 0
```

The matrix $M2$ is the inverse of the matrix M mod 26. We can check this as follows:

```
>> mod(M2*M,26)
ans =
    1   0   0
    0   1   0
    0   0   1
```

To decrypt, we break the ciphertext into blocks of three numbers and multiply each block on the right by the inverse matrix we just calculated:

```
>> mod([22,9,0]*M2,26)

ans =
    14   21   4

>> mod([12,3,1]*M2,26)

ans =
    17   19   7

>> mod([10,3,4]*M2,26)

ans =
    4   7   8

>> mod([8,1,17]*M2,26)

ans =
    11   11   23
```

Therefore, the plaintext is 14, 21, 4, 17, 19, 7, 4, 7, 8, 11, 11, 23. This can be changed back to letters:

```
>> int2text([14 21 4 17 19 7 4 7 8 11 11 23])

ans =

overthehillx
```

Note that the final x was appended to the plaintext in order to complete a block of three letters.

C.6 Examples for Chapter 9

Example 22

Two functions, *nextprime* and *randprime*, can be used to generate prime numbers. The function *nextprime* takes a number n as input and attempts to find the next prime after n. The function *randprime* takes a number n as input and attempts to find a random prime between 1 and n. It uses the Miller-Rabin test described in Chapter 9.

```
>> nextprime(346735)

ans =
   346739

>> randprime(888888)

ans =
   737309
```

For larger inputs, use symbolic mode:

```
>> nextprime(10^sym(60))

ans =

1000000000000000000000000000000000000000000000000000000000007

>> randprime(10^sym(50))

ans =

582325165358256624514865507080685347318649291992 19
```

It is interesting to note the difference that the `' '` makes when entering a large integer:

```
>>
nextprime(sym('12345678901234567890123456789O'))

ans =
   12345678901234567890123456907

>> nextprime(sym(123456789012345678901234567890))

ans =
   12345678901234567787771959707 1
```

In the second case, the input was a number, so only the first 16 digits of the input were used correctly when changing to symbolic mode, while the first case regarded the entire input as a string and therefore used all of the digits.

Example 23

Suppose you want to change the text `hellohowareyou` to numbers:

```
>> text2int1('hellohowareyou')

ans =
   8051212150815230118052515 21
```

Note that we are now using *a = 1, b = 2, ..., z = 26*, since otherwise *a*'s at the beginnings of messages would disappear. (A more efficient procedure would be to work in base 27, so the numerical form of the message would be $8 + 5 \cdot 27 + 12 \cdot 27^2 + \cdots + 21 \cdot 27^{13} = 87495221502384554951$. Note that this uses fewer digits.)

Now suppose you want to change it back to letters:

```
>> int2text1(805121215081523011805251521)

ans =
   'hellohowareyou'
```

Example 24

Encrypt the message `hi` using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the message to numbers:

```
>> text2int1('hi')

ans =
   809
```

Now, raise it to the eth power mod n:

```
>> powermod(ans,17,823091)

ans =
   596912
```

Example 25

Decrypt the ciphertext in the previous problem.

SOLUTION

First, we need to find the decryption exponent d. To do this, we need to find $\phi(823091)$. One way is

```
>> eulerphi(823091)
```

```
ans =
   821184
```

Another way is to factor n as $p \cdot q$ and then compute $(p-1)(q-1)$:

```
>> factor(823091)

ans =
    659   1249

>> 658*1248

ans =
   821184
```

Since $de \equiv 1 \pmod{\phi(n)}$, we compute the following (note that we are finding the inverse of e mod $\phi(n)$, not mod n):

```
>> invmodn(17,821184)

ans =
   48305
```

Therefore, $d = 48305$. To decrypt, raise the ciphertext to the dth power mod n:

```
>> powermod(596912,48305,823091)

ans =
   809
```

Finally, change back to letters:

```
>> int2text1(ans)

ans =
   hi
```

Example 26

Encrypt `hellohowareyou` using RSA with $n = 823091$ and $e = 17$.

SOLUTION

First, change the plaintext to numbers:

```
>> text2int1('hellohowareyou')

ans =
   805121215081523011805251521
```

Suppose we simply raised this to the eth power mod n:

```
>> powermod(ans,17,823091)

ans =
   447613
```

If we decrypt (we know d from Example 25), we obtain

```
>> powermod(ans,48305,823091)

ans =
   628883
```

This is not the original plaintext. The reason is that the plaintext is larger than n, so we have obtained the plaintext mod n:

```
>> mod(text2int1('hellohowareyou'),823091)

ans =
   628883
```

We need to break the plaintext into blocks, each less than n. In our case, we use three letters at a time:

$$80512 \quad 121508 \quad 152301 \quad 180525 \quad 1521$$

```
>> powermod(80512,17,823091)

ans =
   757396

>> powermod(121508,17,823091)

ans =
   164513

>> powermod(152301,17,823091)

ans =
   121217

>> powermod(180525,17,823091)

ans =
   594220

>> powermod(1521,17,823091)

ans =
   442163
```

The ciphertext is therefore 757396164513121217594220442163. Note that there is no reason to change this back to letters. In fact, it doesn't correspond to any text with letters.

Decrypt each block individually:

```
>> powermod(757396,48305,823091)

ans =
   80512

>> powermod(164513,48305,823091)

ans =
   121508
```

Etc.

We'll now do some examples with large numbers, namely the numbers in the RSA Challenge discussed in Section

9.5. These are stored under the names *rsan, rsae, rsap, rsaq*:

```
>> rsan

ans =

114381625757888867669235779976146612010218296721
42362562561842935
706935245733897830597123563958705058989075147599
290026879543541

>> rsae

ans =
   9007
```

Example 27

Encrypt each of the messages *b, ba, bar, bard* using *rsan* and *rsae*.

```
>> powermod(text2int1('b'), rsae, rsan)

ans =

709467584676126685983701649915507861828763310606
85235410564704114
4
867822617164972001221553323484620140532879875808
99263765142534

>> powermod(text2int1('ba'), rsae, rsan)

ans =

350451306089751003250117094498719542737882047539
485930603136976982
276217598060279622705380315655647733520336717822
61305796158951

>> powermod(txt2int1('bar'), rsae, rsan)

ans =

448145128638551010760045308594921093424295316066
074090703605434080
008436459868804059531028183128225863625802987844
```

```
1151922606424

>> powermod(text2int1('bard'), rsae, rsan)

ans =

24238077785111666423202862512090317393485212959056270783134991614256054323297179804928958073445752663026449873986877989329909498
```

Observe that the ciphertexts are all the same length. There seems to be no easy way to determine the length of the corresponding plaintext.

Example 28

Using the factorization $rsan = rsap \cdot rsaq$, find the decryption exponent for the RSA Challenge, and decrypt the ciphertext (see Section 9.5).

SOLUTION

First, we find the decryption exponent:

```
>> rsad=invmodn(rsae,-1,(rsap-1)*(rsaq-1));
```

Note that we use the final semicolon to avoid printing out the value. If you want to see the value of *rsad*, see Section 9.5, or don't use the semicolon. To decrypt the ciphertext, which is stored as *rsaci*, and change to letters:

```
>> int2text1(powermod(rsaci, rsad, rsan))

ans =
   the magic words are squeamish ossifrage
```

Example 29

Encrypt the message *rsaencryptsmessageswell* using *rsan* and *rsae*.

```
>> ci =
powermod(text2int1('rsaencryptsmessageswell'),
rsae, rsan)
   ci =

946394203490022593163058235392494964141640969934001709721404352418271950654254365584906013966328817753539283112653197553130781884
```

We called the ciphertext `ci` because we need it in Example 30.

Example 30

Decrypt the preceding ciphertext.

SOLUTION

Fortunately, we know the decryption exponent *rsad*. Therefore, we compute

```
>> powermod(ans, rsad, rsan)

ans =
   1819010514031825162019130519190107051923051212

>> int2text1(ans)

ans =
   rsaencryptsmessageswell
```

Suppose we lose the final 4 of the ciphertext in transmission. Let's try to decrypt what's left (subtracting 4 and dividing by 10 is a mathematical way to remove the 4): `>> powermod((ci - 4)/10, rsad, rsan)`

```
ans =

47952999173195988664902352629525486409113633894375629846854907970588412300373487969657794254117158956921267912628461494475682806
```

If we try to change this to letters, we get a weird-looking answer. A small error in the plaintext completely changes the decrypted message and usually produces garbage.

Example 31

Suppose we are told that $n = 11313771275590312567$ is the product of two primes and that $\phi(n) = 11313771187608744400$. Factor n.

SOLUTION

We know (see Section 9.1) that p and q are the roots of $X^2 - (n - \phi(n) + 1)X + n$. Therefore, we compute (vpa is for variable precision arithmetic)

```
>> digits(50); syms y; vpasolve(y^2-(sym('11313771275590312567') - sym('11313771187608744400')+1)*y+sym('11313771275590312567'),y)

ans =
   128781017.0 87852787151.0
```

Therefore, $n = 128781017 \cdot 87852787151$. We also could have used the quadratic formula to find the roots.

Example 32

Suppose we know *rsae* and *rsad*. Use these to factor *rsan*.

SOLUTION

We use the $a^r \equiv 1$ factorization method from Section 9.4. First write $rsae \cdot rsad - 1 = 2^s m$ with m odd. One way to do this is first to compute

```
>> rsae*rsad - 1

ans =

961034419617782266156919023359583834109854129051878330250644604041155985575087352659156174898557342995131594680431089621245830097664

>> ans/2

ans =

480517209808891133078459511679791917054927064525939165125322302020577992787543676329578087449278671497565797340215543460622915048832

>> ans/2

ans =

240258604904445566539229755839895958527463532262969582562661151010288996393771838164789043724639335748782898670107771730311457524416
```

We continue this way for six more steps until we get

```
ans =

375404070163196197717546493499837435199161769160889972754158048453576556865268497132482880819748962107473279172043393328611652381 9
```

This number is m. Now choose a random integer a. Hoping to be lucky, we choose 13. As in the $a^r \equiv 1$ factorization method, we compute

```
>>b0=powermod(13, ans, rsan)

b0 =

2757436850700653059224349486884716119842309570730
780569056983964
7030183109839862370800529338092984795490192643587
960859870551239
```

Since this is not $\pm 1 \pmod{rsan}$, we successively square it until we get ± 1:

```
>> b1=powermod(b0,2,rsan)

b1 =

4831896032192851558013847641872303455410409906994
084622549470277
6654996412582955636035266156108686431194298574075
854037512277292

>> b2=powermod(b1,2,rsan)

b2 =

7817281415487735657914192805875400002194878705648
382091793062511
5215181839742056013275521913487560944732073516487
722273875579363

>> b3=powermod(b2, 2, rsan)

b3 =

4283619120250872874219929904058290020297622291601
776716755187021
6509444518239462186379470569442055101392992293082
259601738228702

>> b4=powermod(b3, 2, rsan)

b4 =
   1
```

Since the last number before the 1 was not $\pm 1 \pmod{rsan}$, we have an example of $x \not\equiv \pm 1 \pmod{rsan}$ with $x^2 \equiv 1$. Therefore, $\gcd(x-1,\ rsan)$ is a nontrivial factor of *rsan*:

```
>> gcd(b3 - 1, rsan)

ans =

3276913299326670954996198819083446141317764296799
2942539798288533
```

This is *rsaq*. The other factor is obtained by computing *rsan/rsaq*:

```
>> rsan/ans

ans =

3490529510847650949147849619903898133417764638493
387843990820577
```

This is *rsap*.

Example 33

Suppose you know that

$$150883475569451^2 \equiv 16887570532858^2 \pmod{205611444308117}.$$

Factor 205611444308117.

SOLUTION

We use the Basic Principle of Section 9.4.

```
>> g= gcd(150883475569451-
16887570532858,205611444308117)

g =
   23495881
```

This gives one factor. The other is

```
>> 205611444308117/g

ans =
   8750957
```

We can check that these factors are actually primes, so we can't factor any further:

```
>> primetest(ans)

ans =
   1

>> primetest(g)

ans =
   1
```

Example 34

Factor
$n = 37687557542639485559998999 2897873239$
by the $p - 1$ method.

SOLUTION

Let's choose our bound as $B = 100$, and let's take $a = 2$, so we compute $2^{100!} \pmod{n}$:

```
>>
powermod(2,factorial(100),sym('376875575426394855
59998999 2897873239')

ans =
   369676678301956331939422106251199512
```

Then we compute the gcd of $2^{100!} - 1$ and n:

```
>> gcd(ans - 1,
sym'(37687557542639485559998992897873239')

ans =
   430553161739796481
```

This is a factor p. The other factor q is

```
>>
sym('37687557542639485559998992897873239')/ans

ans =
   875328783798732119
```

Let's see why this worked. The factorizations of $p-1$ and $q-1$ are

```
>> factor(sym('430553161739796481') - 1)

ans =
   [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 5, 7, 7, 7, 7, 11,
11, 11, 47]

>> factor(sym('875328783798732119') - 1)

ans =
   [ 2, 61, 20357, 39301, 8967967]
```

We see that $100!$ is a multiple of $p-1$, so $2^{100!} \equiv 1 \pmod{p}$. However, $100!$ is not a multiple of $q-1$, so it is likely that $2^{100!} \not\equiv 1 \pmod{q}$. Therefore, both $2^{100!}-1$ and pq have p as a factor, but only pq has q as a factor. It follows that the gcd is p.

C.7 Examples for Chapter 10

Example 35

Let's solve the discrete log problem $2^x \equiv 71 \pmod{131}$ by the Baby Step-Giant Step method of Subsection 10.2.2. We take $N = 12$ since $N^2 > p - 1 = 130$ and we form two lists. The first is $2^j \pmod{131}$ for $0 \leq j \leq 11$:

```
>> for k=0:11;z=[k,
powermod(2,k,131)];disp(z);end;

 0  1
 1  2
 2  4
 3  8
 4  16
 5  32
 6  64
 7  128
 8  125
 9  119
 10  107
 11  83
```

The second is $71 \cdot 2^{-j} \pmod{131}$ for $0 \leq j \leq 11$:

```
>> for k=0:11;z=[k,
mod(71*invmodn(powermod(2,12*k,131),131),131)];
disp(z);end;

 0  71
 1  17
 2  124
 3  26
 4  128
 5  86
 6  111
 7  93
 8  85
```

```
9 96
10 130
11 116
```

The number 128 is on both lists, so we see that $2^7 \equiv 71 \cdot 2^{-12*4} \pmod{131}$. Therefore,

$$71 \equiv 2^{7+4*12} \equiv 2^{55} \pmod{131}.$$

C.8 Examples for Chapter 12

Example 36

Suppose there are 23 people in a room. What is the probability that at least two have the same birthday?

SOLUTION

The probability that no two have the same birthday is $\prod_{i=1}^{22}(1 - i/365)$ (note that the product stops at $i = 22$, not $i = 23$). Subtracting from 1 gives the probability that at least two have the same birthday:

```
>> 1-prod( 1 - (1:22)/365)

ans =
   0.5073
```

Example 37

Suppose a lazy phone company employee assigns telephone numbers by choosing random seven-digit numbers. In a town with 10,000 phones, what is the probability that two people receive the same number?

```
>> 1-prod( 1 - (1:9999)/10^7)

ans =
   0.9933
```

Note that the number of phones is about three times the square root of the number of possibilities. This means

that we expect the probability to be high, which it is. From Section 12.1, we have the estimate that if there are around $\sqrt{2(\ln 2)10^7} \approx 3723$ phones, there should be a 50% chance of a match. Let's see how accurate this is:

```
>> 1-prod( 1 - (1:3722)/10^7)

ans =
   0.4999
```

C.9 Examples for Chapter 17

Example 38

Suppose we have a (5, 8) Shamir secret sharing scheme. Everything is mod the prime $p = 987541$. Five of the shares are

(9853, 853), (4421, 4387), (6543, 1234), (93293, 78428), (12398, 7563).

Find the secret.

SOLUTION

The function *interppoly(x,f,m)* calculates the interpolating polynomial that passes through the points (x_j, f_j). The arithmetic is done mod m.

In order to use this function, we need to make a vector that contains the x values, and another vector that contains the share values. This can be done using the following two commands:

```
>> x=[9853 4421 6543 93293 12398];

>> s=[853 4387 1234 78428 7563];
```

Now we calculate the coefficients for the interpolating polynomial.

```
>> y=interppoly(x,s,987541)

y =
    678987   14728   1651   574413   456741
```

The first value corresponds to the constant term in the interpolating polynomial and is the secret value.
Therefore, 678987 is the secret.

C.10 Examples for Chapter 18

Example 39

Here is a game you can play. It is essentially the simplified version of poker over the telephone from Section 18.2. There are five cards: `ten, jack, queen, king, ace`. We have chosen to abbreviate them by the following: ten, ace, que, jac, kin. They are shuffled and disguised by raising their numbers to a random exponent mod the prime 300649. You are supposed to guess which one is the ace.

First, the cards are entered in and converted to numerical values by the following steps:

```
>> cards=['ten';'ace';'que';'jac';'kin'];

>> cvals=text2int1(cards)

cvals =

   200514
   10305
   172105
   100103
   110914
```

Next, we pick a random exponent k that will be used in the hiding operation. We use the semicolon after *khide* so that we cannot cheat and see what value of k is being used.

```
>> p=300649;
```

```
>> k=khide(p);
```

Now, shuffle the disguised cards (their numbers are raised to the kth power mod p and then randomly permuted):

```
>> shufvals=shuffle(cvals,k,p)

shufvals =
    226536
    226058
    241033
    281258
    116809
```

These are the five cards. None looks like the ace; that's because their numbers have been raised to powers mod the prime. Make a guess anyway. Let's see if you're correct.

```
>> reveal(shufvals,k,p)

ans =


jac
que
ten
kin
ace
```

Let's play again:

```
>> k=khide(p);

» shufvals=shuffle(cvals,k,p)

shufvals =
    117135
    144487
    108150
    266322
    264045
```

Make your guess (note that the numbers are different because a different random exponent was used). Were you lucky?

```
>> reveal(shufvals,k,p)

ans =

kin
jac
ten
que
ace
```

Perhaps you need some help. Let's play one more time:

```
>> k=khide(p);

» shufvals=shuffle(cvals,k,p)

shufvals =
    108150
    144487
    266322
    264045
    117135
```

We now ask for advice:

```
>> advise(shufvals,p);
```

```
Ace Index: 4
```

We are advised that the fourth card is the ace. Let's see:

```
>> reveal(shufvals,k,p)

ans =


ten
jac
que
```

```
ace
kin
```

How does this work? Read the part on “How to Cheat” in Section 18.2. Note that if we raise the numbers for the cards to the $(p-1)/2$ power mod p, we get

```
>> powermod(cvals,(p-1)/2,p)

ans =
        1
    300648
        1
        1
        1
```

Therefore, only the ace is a quadratic nonresidue mod p.

C.11 Examples for Chapter 21

Example 40

We want to graph the elliptic curve
$y^2 = x(x-1)(x+1)$.

First, we create a string v that contains the equation we wish to graph.

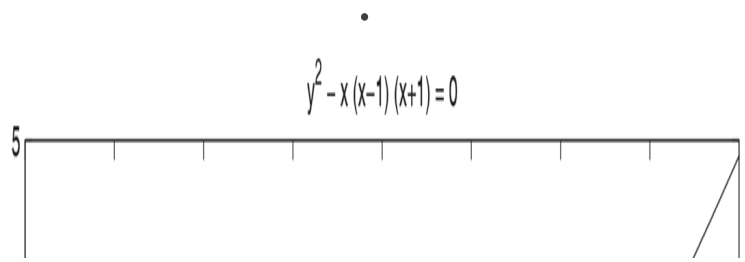
```
>> v='y^2 - x*(x-1)*(x+1)';
```

Next we use the *ezplot* command to plot the elliptic curve.

```
>> ezplot(v,[-1,3,-5,5])
```

The plot appears in Figure C.1. The use of $[-1, 3, -5, 5]$ in the preceding command is to define the limits of the x-axis and y-axis in the plot.

Figure C.1 Graph of the Elliptic Curve $y^2 = x(x-1)(x+1)$



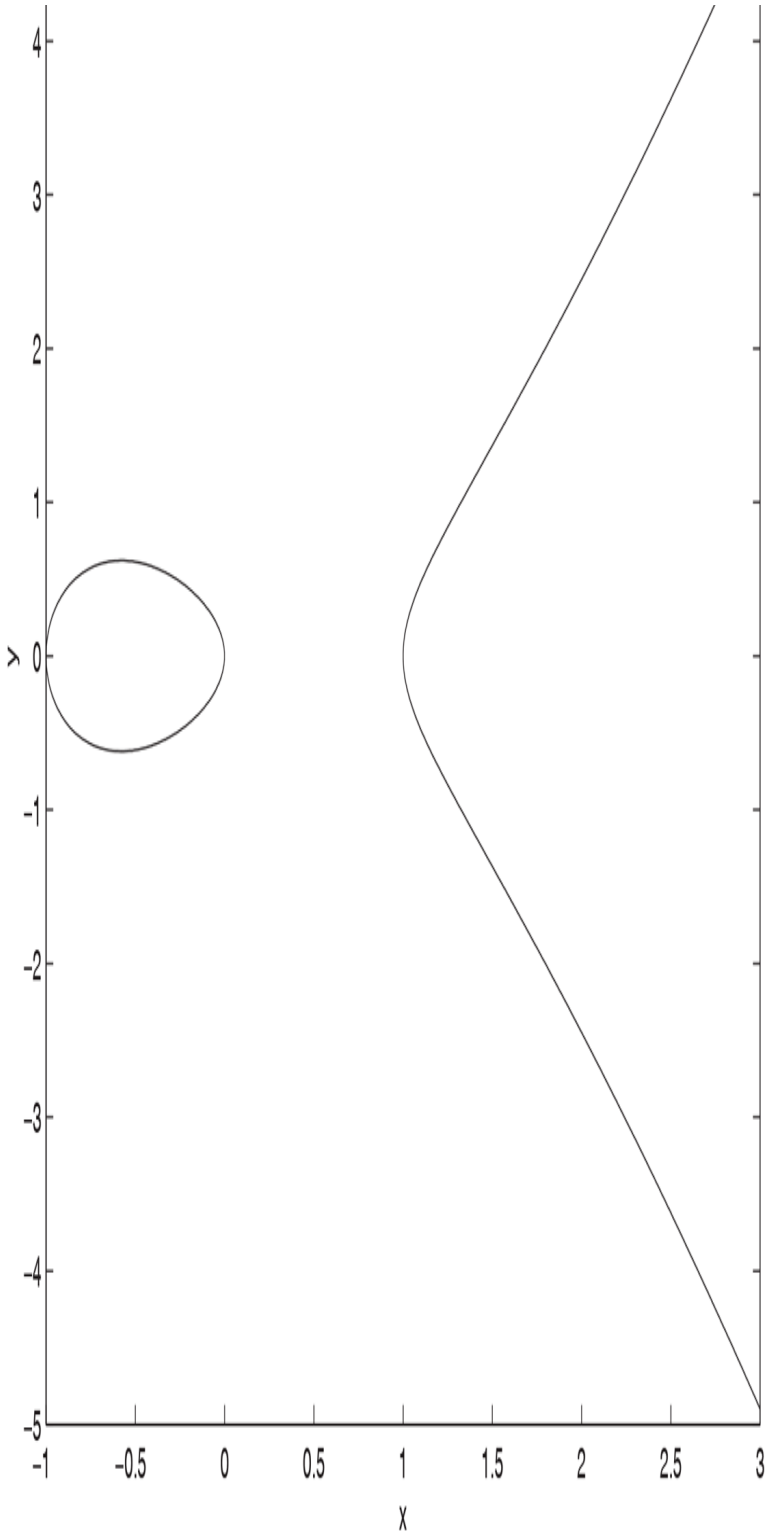


Figure C.1 Full Alternative Text

Example 41

Add the points (1,3) and (3,5) on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$.

```
>> addell([1,3],[3,5],24,13,29)

ans =
    26     1
```

You can check that the point (26,1) is on the curve: $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$. (*Note:* `addell([x,y],[u,v],b,c,n)` is only programmed to work for odd n.)

Example 42

Add (1,3) to the point at infinity on the curve of the previous example.

```
>> addell([1,3],[inf,inf],24,13,29)

ans =
    1     3
```

As expected, adding the point at infinity to a point P returns the point P.

Example 43

Let $P = (1, 3)$ be a point on the elliptic curve $y^2 \equiv x^3 + 24x + 13 \pmod{29}$. Find $7P$.

```
>> multell([1,3],7,24,13,29)

ans =
    15     6
```

Example 44

Find $k(1, 3)$ for $k = 1, 2, 3, \ldots, 40$ on the curve of the previous example.

```
>> multsell([1,3],40,24,13,29)

ans =
   1:    1     3
   2:    11    10
   3:    23    28
   4:    0     10
   5:    19    9
   6:    18    19
   7:    15    6
   8:    20    24
   9:    4     12
  10:    4     17
  11:    20    5
  12:    15    23
  13:    18    10
  14:    19    22
  15:    0     19
  16:    23    1
  17:    11    19
  18:    1     26
  19:   inf   Inf
  20:    1     3
  21:    10    10
  22:    23    28
  23:    0     10
  24:    19    7
  25:    18    19
  26:    15    6
  27:    20    24
  28:    4     12
  29:    4     17
  30:    20    5
  31:    15    23
  32:    18    10
  33:    19    22
  34:    0     19
  35:    23    1
  36:    11    19
  37:    1     26
  38:   inf   inf
  39:    1     3
  40:    10    10
```

Notice how the points repeat after every 19 multiples.

Example 45

The previous four examples worked mod the prime 29. If we work mod a composite number, the situation at infinity becomes more complicated since we could be at infinity mod both factors or we could be at infinity mod one of the factors but not mod the other. Therefore, we stop the calculation if this last situation happens and we exhibit a factor. For example, let's try to compute $12P$, where $P = (1, 3)$ is on the elliptic curve $y^2 \equiv x^3 - 5x + 13 \pmod{209}$:

```
>> multell([1,3],12,-5,13,11*19)

Elliptic Curve addition produced a factor of n,
factor= 19
Multell found a factor of n and exited

ans =
   []
```

Now let's compute the successive multiples to see what happened along the way:

```
>> multsell([1,3],12,-5,13,11*19)

Elliptic Curve addition produced a factor of n,
factor= 19
Multsell ended early since it found a factor

ans =
1:   1     3
2:   91    27
3:   118   133
4:   148   182
5:   20    35
```

When we computed $6P$, we ended up at infinity mod 19. Let's see what is happening mod the two prime factors of 209, namely 19 and 11:

```
>> multsell([1,3],20,-5,13,19)
```

```
ans =
 1:    1     3
 2:   15     8
 3:    4     0
 4:   15    11
 5:    1    16
 6:   Inf   Inf
 7:    1     3
 8:   15     8
 9:    4     0
10:   15    11
11:   15     8
12:   Inf   Inf
13:    1     3
14:   15     8
15:    4     0
16:   15    11
17:    1    16
18:   Inf   Inf
19:    1     3
20:   15     8
```

```
>> multsell([1,3],20,-5,13,11)

ans =
 1:    1     3
 2:    3     5
 3:    8     1
 4:    5     6
 5:    9     2
 6:    6    10
 7:    2     0
 8:    6     1
 9:    9     9
10:    5     5
11:    8    10
12:    3     6
13:    1     8
14:   Inf   Inf
15:    1     3
16:    3     5
17:    8     1
18:    5     6
19:    9     2
20:    6    10
```

After six steps, we were at infinity mod 19, but it takes 14 steps to reach infinity mod 11. To find $6P$, we needed to invert a number that was 0 mod 19 and nonzero mod 11.

This couldn't be done, but it yielded the factor 19. This is the basis of the elliptic curve factorization method.

Example 46

Factor 193279 using elliptic curves.

SOLUTION

First, we need to choose some random elliptic curves and a point on each curve. For example, let's take $P = (2, 4)$ and the elliptic curve

$$y^2 \equiv x^3 - 10x + b \pmod{193279}.$$

For P to lie on the curve, we take $b = 28$. We'll also take

$$\begin{aligned} y^2 &\equiv x^3 + 11x - 11, \quad P = (1, 1), \\ y^2 &\equiv x^3 + 17x - 14, \quad P = (1, 2). \end{aligned}$$

Now we compute multiples of the point P. We do the analog of the $p - 1$ method, so we choose a bound B, say $B = 12$, and compute $B!P$.

```
>> multell([2,4],factorial(12),-10,28,193279)

Elliptic Curve addition produced a factor of n,
factor= 347
Multell found a factor of n and exited

ans =
   []

>> multell([1,1],factorial(12),11,-11,193279)

ans =
    13862    35249

» multell([1,2],factorial(12),17,-14,193279)

Elliptic Curve addition produced a factor of n,
factor= 557
Multell found a factor of n and exited
```

```
ans =
   []
```

Let's analyze in more detail what happened in these examples.

On the first curve, $266P$ ends up at infinity mod 557 and $35P$ is infinity mod 347. Since $272 = 2 \cdot 7 \cdot 9$, it has a prime factor larger than $B = 12$, so $B!P$ is not infinity mod 557. But 35 divides $B!$, so $B!P$ is infinity mod 347.

On the second curve, $356P = \text{infinity}$ mod 347, and $561P = \text{infinity}$ mod 557. Since $356 = 4 \cdot 89$ and $561 = 3 \cdot 11 \cdot 17$, we don't expect to find the factorization with this curve.

The third curve is a surprise. We have $331P = \text{infinity}$ mod 347 and $272P = \text{infinity}$ mod 557. Since 331 is prime and $272 = 16 \cdot 17$, we don't expect to find the factorization with this curve. However, by chance, an intermediate step in the calculation of $B!P$ yielded the factorization. Here's what happened. At an intermediate step in the calculation, the program required adding the points $(184993, 13462)$ and $(20678, 150484)$. These two points are congruent mod 557 but not mod 347. Therefore, the slope of the line through these two points is defined mod 347 but is $0/0$ mod 557. When we tried to find the multiplicative inverse of the denominator mod 193279, the gcd algorithm yielded the factor 557. This phenomenon is fairly rare.

Example 47

Here is how to produce the example of an elliptic curve ElGamal cryptosystem from Section 21.5. For more details, see the text. The elliptic curve is $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$ and the point is

$G = (4, 11)$. Alice's message is the point $P_m = (5, 1743)$.

Bob has chosen his secret random number $a_B = 3$ and has computed $a_B G$:

```
>> multell([4,11],3,3,45,8831)

ans =
    413    1808
```

Bob publishes this point. Alice chooses the random number $k = 8$ and computes kG and $P_m + k(a_B G)$:

```
>> multell([4,11],8,3,45,8831)

ans =
    5415    6321

>>
addell([5,1743],multell([413,1808],8,3,45,8831),3
,45,8831)

ans =
    6626    3576
```

Alice sends (5415,6321) and (6626, 3576) to Bob, who multiplies the first of these point by a_B:

```
>> multell([5415,6321],3,3,45,8831) ans = 673  146
```

Bob then subtracts the result from the last point Alice sends him. Note that he subtracts by adding the point with the second coordinate negated:

```
>> addell([6626,3576],[673,-146],3,45,8831)

ans =
   5 1743
```

Bob has therefore received Alice's message.

Example 48

Let's reproduce the numbers in the example of a Diffie-Hellman key exchange from Section 21.5: The elliptic curve is $y^2 \equiv x^3 + x + 7206 \pmod{7211}$ and the point is $G = (3, 5)$. Alice chooses her secret $N_A = 12$ and Bob chooses his secret $N_B = 23$. Alice calculates

```
>> multell([3,5],12,1,7206,7211)

ans =
    1794    6375
```

She sends (1794,6375) to Bob. Meanwhile, Bob calculates

```
>> multell([3,5],23,1,7206,7211)

ans =
    3861    1242
```

and sends (3861,1242) to Alice. Alice multiplies what she receives by N_A and Bob multiplies what he receives by N_B:

```
>> multell([3861,1242],12,1,7206,7211)

ans =
    1472    2098

>> multell([1794,6375],23,1,7206,7211)

ans =
    1472    2098
```

Therefore, Alice and Bob have produced the same key.

Appendix D Sage Examples

Sage is an open-source computer algebra package. It can be downloaded for free from www.sagemath.org/ or it can be accessed directly online at the website https://sagecell.sagemath.org/. The computer computations in this book can be done in Sage, especially by those comfortable with programming in Python. In the following, we give examples of how to do some of the basic computations. Much more is possible. See www.sagemath.org/ or search the Web for other examples. Another good place to start learning Sage in general is [Bard] (there is a free online version).

D.1 Computations for Chapter 2

Shift ciphers

Suppose you want to encrypt the plaintext `This is the plaintext` with a shift of 3. We first encode it as an alphabetic string of capital letters with the spaces removed. Then we shift each letter by three positions:

```
S=ShiftCryptosystem(AlphabeticStrings())
P=S.encoding("This is the plaintext")
C=S.enciphering(3,P);C
```

When this is evaluated, we obtain the ciphertext

```
WKLVLVWKHSODLQWHAW
```

To decrypt, we can shift by 23 or do the following:

```
S.deciphering(3,C)
```

When this is evaluated, we obtain

```
THISISTHEPLAINTEXT
```

Suppose we don't know the key and we want to decrypt by trying all possible shifts:

```
S.brute_force(C)
```

Evaluation yields

```
0: WKLVLVWKHSODLQWHAW,
1: VJKUKUVJGRNCKPVGZV,
2: UIJTJTUIFQMBJOUFYU,
3: THISISTHEPLAINTEXT,
4: SGHRHRSGDOKZHMSDWS,
5: RFGQGQRFCNJYGLRCVR,
6: etc.

24: YMNXNXYMJUQFNSYJCY,
25: XLMWMWXLITPEMRXIBX
```

Affine ciphers

Let's encrypt the plaintext `This is the plaintext` using the affine function $3x + 1 \bmod 26$:

```
A=AffineCryptosystem(AlphabeticStrings())
P=A.encoding("This is the plaintext")
C=A.enciphering(3,1,P);C
```

When this is evaluated, we obtain the ciphertext

```
GWZDZDGWNUIBZOGNSG
```

To decrypt, we can do the following:

```
A.deciphering(3,1,C)
```

When this is evaluated, we obtain

```
THISISTHEPLAINTEXT
```

We can also find the decryption key:

```
A.inverse_key(3,1)
```

This yields

```
(9, 17)
```

Of course, if we "encrypt" the ciphertext using $9x + 17$, we obtain the plaintext:

```
A.enciphering(9,17,C)
```

Evaluate to obtain

```
THISISTHEPLAINTEXT
```

Vigenère ciphers

Let's encrypt the plaintext `This is the plaintext` using the keyword `ace` (that is, shifts of 0, 2, 4). Since we need to express the keyword as an alphabetic string, it is efficient to add a symbol for these strings:

```
AS=AlphabeticStrings()
V=VigenereCryptosystem(AS,3)
K=AS.encoding("ace")
P=V.encoding("This is the plaintext")
C=V.enciphering(K,P);C
```

The "3" in the expression for `V` is the length of the key. When the above is evaluated, we obtain the ciphertext

```
TJMSKWTJIPNEIPXEZX
```

To decrypt, we can shift by $0,\ 24,\ 22\ (=\ \text{ayw})$ or do the following:

```
V.deciphering(K,C)
```

When this is evaluated, we obtain

```
THISISTHEPLAINTEXT
```

Now let's try the example from Section 2.3. The ciphertext can be cut and pasted from ciphertexts.m in the MATLAB files (or, with a little more difficulty, from the Mathematica or Maple files). A few control symbols need to be removed in order to make the ciphertext a single string.

```
vvhq="vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtf
s . . . czvile"
```

(We omitted part of the ciphertext in the above in order to save space.) Now let's compute the matches for various displacements. This is done by forming a string that displaces the ciphertext by i positions by adding i blank spaces at the beginning and then counting matches.

```
for i in range(0,7):
C2 = [" "]*i + list(C)
count = 0
```

```
for j in range(len(C)):
if C2[j] == C[j]:
count += 1
print i, count
```

The result is

```
0 331
1 14
2 14
3 16
4 14
5 24
6 12
```

The `331` is for a displacement of 0, so all 331 characters match. The high number of matches for a displacement of 5 suggests that the key length is 5. We now want to determine the key.

First, let's choose every fifth letter, starting with the first (counted as 0 for Sage). We extract these letters, put them in a list, then count the frequencies.

```
V1=list(C[0::5])
dict((x, V1.count(x)) for x in V1)
```

The result is

```
C: 7,
D: 1,
E: 1,
F: 2,
G: 9,
I: 1,
J: 8,
K: 8,
N: 3,
P: 4,
Q: 5,
R: 2,
T: 3,
```

```
U: 6,
V: 5,
W: 1,
Y: 1
```

Note that `A, B, H, L, M, O. S, X, Z` do not occur among the letters, hence are not listed. As discussed in Subsection 2.3.2 , the shift for these letters is probably 2. Now, let's choose every fifth letter, starting with the second (counted as 1 for Sage). We compute the frequencies:

```
V2=list(C[1::5])
dict((x, V2.count(x)) for x in V2)

A: 3,
B: 3,
C: 4,
F: 3,
G: 10,
H: 6,
M: 2,
O: 3,
P: 1,
Q: 2,
R: 3,
S: 12,
T: 3,
U: 2,
V: 3,
W: 3,
Y: 1,
Z: 2
```

As in Subsection 2.3.2 , the shift is probably 14. Continuing in this way, we find that the most likely key is $\{2, 14, 3, 4, 18\}$, which is `codes`. let's decrypt:

```
V=VigenereCryptosystem(AS,5)

K=AS.encoding("codes")
P=V.deciphering(K,C);P

THEMETHODUSEDFORTHEPREPARATIONANDREADINGOFCODEMES
. . . ALSETC
```

D.2 Computations for Chapter 3

To find the greatest common divisor, type the following first line and then evaluate:

```
gcd(119, 259)
7
```

To find the next prime greater than or equal to a number:

```
next_prime(1000)
1009
```

To factor an integer:

```
factor(2468)
2^2 * 617
```

Let's solve the simultaneous congruences :

```
crt(1,3,5,7)
31
```

To solve the three simultaneous congruences :

```
a= crt(1,3,5,7)
```

```
crt(a,0,35,11)
66
```

Compute :

```
mod(123,789)456
699
```

Compute so that :

```
mod(65,987)(-1)
410
```

Let’s check the answer:

```
mod(65*410, 987)
1
```

D.3 Computations for Chapter 5

LFSR

Consider the recurrence relation $x_{n+4} \equiv x_n + x_{n+1} + x_{n+3} \bmod 2$, with initial values $1, 0, 0, 0$. We need to use 0s and 1s, but we need to tell Sage that they are numbers mod 2. One way is to define "**o**" (that's a lower-case "oh") and "**l**" (that's an "ell") to be 0 and 1 mod 2:

```
F=GF(2)
o=F(0); l=F(1)
```

We also could use `F(0)` every time we want to enter a 0, but the present method saves some typing. Now we specify the coefficients and initial values of the recurrence relation, along with how many terms we want. In the following, we ask for 20 terms:

```
s=lfsr_sequence([l,l,o,l],[l,o,o,o],20);s
```

This evaluates to

```
[1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
1, 1, 1, 0]
```

Suppose we are given these terms of a sequence and we want to find what recurrence relation generates it:

```
berlekamp_massey(s)
```

This evaluates to

```
x^4 + x^3 + x + 1
```

When this is interpreted as $x^4 \equiv 1 + 1x + 0x^2 + 1x^3$ mod 2, we see that the coefficients 1, 1, 0, 1 of the polynomial give the coefficients of recurrence relation. In fact, it gives the smallest relation that generates the sequence.

Note: Even though the output for `s` has 0s and 1s, if we try entering the command `berlekamp_massey([1,1,0,1,1,0])` we get `x^3 - 1`. If we instead enter `berlekamp_massey([l,l,o,l,l,o])`, we get `x^2 + x + 1`. Why? The first is looking at a sequence of integers generated by the relation $x_{n+3} = x_n$ while the second is looking at the sequence of integers mod 2 generated by $x_{n+2} \equiv x_n + x_{n+1}$ mod 2. Sage defaults to integers if nothing is specified. But it remembers that the 0s and 1s that it wrote in **s** are still integers mod 2.

D.4 Computations for Chapter 6

Hill ciphers

Let's encrypt the plaintext `This is the plaintext` using a matrix. First, we need to specify that we are working with such a matrix with entries in the integers mod 26:

```
R=IntegerModRing(26)
M=MatrixSpace(R,3,3)
```

Now we can specify the matrix that is the encryption key:

```
K=M([[1,2,3],[4,5,6],[7,8,10]]);K
```

Evaluate to obtain

```
[ 1 2  3]
[ 4 5  6]
[ 7 8 10]
```

This is the encryption matrix. We can now encrypt:

```
H=HillCryptosystem(AlphabeticStrings(),3)
P=H.encoding("This is the plaintext")
C=H.enciphering(K,P);C
```

If the length of the plaintext is not a multiple of 3 (= the size of the matrix), then extra characters need to be appended to achieve this. When the above is evaluated, we obtain the ciphertext

```
ZHXUMWXBJHHHLZGVPC
```

Decrypt:

```
H.deciphering(K,C)
```

When this is evaluated, we obtain

```
THISISTHEPLAINTEXT
```

We could also find the inverse of the encryption matrix mod 26:

```
K1=K.inverse();K1
```

This evaluates to

```
[ 8 16  1]
[ 8 21 24]
[ 1 24  1]
```

When we evaluate

```
H.enciphering(K,C):C
```

we obtain

```
THISISTHEPLAINTEXT
```

D.5 Computations for Chapter 9

Suppose someone unwisely chooses RSA primes and to be consecutive primes:

```
p=nextprime(987654321*10^50+12345);
q=nextprime(p+1)
n=p*q
```

Let's factor the modulus

without using the `factor` command:

```
s=N(sqrt(n), digits=70)
p1=next_prime(s)
p1, q1
(9876543210000000000000000000000000000000000000000
00000012773,
9876543210000000000000000000000000000000000000000
0000012617)
```

Of course, the fact that and are consecutive primes is important for this calculation to work. Note that we needed to specify 70-digit accuracy so that round-off error would not give us the wrong starting point for looking for the next prime. These factors we obtained match the original and , up to order:

```
p, q
(9876543210000000000000000000000000000000000000000
00000012617,
```

```
9876543210000000000000000000000000000000000000000
0000012773)
```

D.6 Computations for Chapter 10

Let's solve the discrete log problem $2^x \equiv 71 \pmod{131}$ by the Baby Step-Giant Step method of Subsection 10.2.2. We take $N = 12$ since $N^2 > p - 1 = 130$ and we form two lists. The first is $2^j \pmod{1}31$ for $0 \leq j \leq 11$:

```
for i in range(0,12): print i, mod(2,131)i

0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 125
9 119
10 107
11 83
```

The second is $71 \cdot 2^{-j} \pmod{1}31$ for $0 \leq j \leq 11$:

```
for i in range(0,12): print i, mod(71*mod(2,131)
(-12*i),131)

0 71
1 17
2 124
3 26
4 128
5 86
6 111
7 93
8 85
9 96
```

```
10 130
11 116
```

The number 128 is on both lists, so we see that $2^7 \equiv 71 \cdot 2^{-12*4} \pmod{131}$. Therefore,

$$71 \equiv 2^{7+4*12} \equiv 2^{55} \pmod{131}.$$

D.7 Computations for Chapter 12

Suppose a lazy phone company employee assigns telephone numbers by choosing random seven-digit numbers. In a town with 10,000 phones, what is the probability that two people receive the same number?

```
i = var('i') 1-product(1.-i/10^7,i,1,9999)
0.9932699132835016
```

D.8 Computations for Chapter 17

Lagrange interpolation

Suppose we want to find the polynomial of degree at most 3 that passes through the points $(1, 1), (2, 2), (3, 21), (5, 12)$ mod the prime $p = 37$. We first need to specify that we are working with polynomials in x mod 37. Then we compute the polynomial:

```
R=PolynomialRing(GF(37),"x")
f=R.lagrange_polynomial([(1,1),(2,2),(3,21),
(5,12)]);f
```

This evaluates to

```
22*x^3 + 25*x^2 + 31*x + 34
```

If we want the constant term:

```
f(0)
43
```

D.9 Computations for Chapter 18

Here is a game you can play. It is essentially the simplified version of poker over the telephone from Section 18.2.

There are five cards: ten, jack, queen, king, ace. They are shuffled and disguised by raising their numbers to a random exponent mod the prime 24691313099. You are supposed to guess which one is the ace.

The cards are represented by `ten = 200514`, etc., because `t` is the 20th letter, `e` is the 5th letter, and `n` is the 14th letter.

Type the following into Sage. The value of k forces the randomization to have 248 as its starting point, so the random choices of e and `shuffle` do not change when we repeat the process with this value of k. Varying k gives different results.

```
cards=[200514,10010311,1721050514,11091407,10305]
p=24691313099
k=248 set_random_seed(k)
e=randint(10,10^7)
def pow(list,ex):
   ret = []
   for i in list:
      ret.append(mod(i,p)^ex)
   return ret
s=pow(cards,2*e+1)
shuffle(s)
print(s)
```

Evaluation yields

```
[10426004161, 16230228497, 12470430058,
3576502017, 2676896936]
```

These are the five cards. None looks like the ace; that's because their numbers have been raised to powers mod the prime. Make a guess anyway. Let's see if you're correct.

Add the following line to the end of the program:

```
print(pow(s,mod(2*e+1,p-1)^(-1)))
```

and evaluate again:

```
[10426004161, 16230228497, 12470430058,
3576502017, 2676896936]
[10010311, 200514, 11091407, 10305, 1721050514]
```

The first line is the shuffled and hidden cards. The second line removed the kth power and reveals the cards. The fourth card is the ace. Were you lucky?

If you change the value of k, you can play again, but let's figure out how to cheat. Remove the last line of the program that you just added and replace it with

```
pow(s,(p-1)/2)
```

When the program is evaluated, we obtain

```
[10426004161, 16230228497, 12470430058,
3576502017, 2676896936]
[1, 1, 1, 24691313098, 1]
```

Why does this tell us the ace is in the fourth position? Read the part on "How to Cheat" in Section 18.2. Raise the numbers for the cards to the $(p-1)/2$ power mod p (you can put this extra line at the end of the program and ignore the previous output):

```
print(pow(cards,(p-1)/2))
[1, 1, 1, 1, 24691313098]
```

We see that the prime p was chosen so that only the ace is a quadratic nonresidue mod p.

If you input another value of k and play the game again, you'll have a similar situation, but with the cards in a different order.

D.10 Computations for Chapter 21

Elliptic curves

Let's set up the elliptic curve $y^2 \equiv x^3 + 2x + 3 \bmod 7$:

```
E=EllipticCurve(IntegerModRing(7),[2,3])
```

The entry `[2,3]` gives the coefficients a, b of the polynomial $x^3 + ax + b$. More generally, we could use the vector `[a,b,c,d,e]` to specify the coefficients of the general form $y^2 + axy + cy \equiv x^3 + bx^2 + dx + e$. We could also use `GF(7)` instead of `IntegerModRing(7)` to specify that we are working mod 7. We could replace the `7` in `IntegerModRing(7)` with a composite modulus, but this will sometimes result in error messages when adding points (this is the basis of the elliptic curve factorization method).

We can list the points on E:

```
E.points()
[(0:1:0), (2:1:1), (2:6:1), (3:1:1), (3:6:1),
(6:0:1)]
```

These are given in projective form. The point `(0:1:0)` is the point at infinity. The point `(2:6:1)` can also be written as `[2, 6]`. We can add points:

```
E([2,1])+E([3,6])
```

```
(6 :  0 :  1)

E([0,1,0])+E([2,6])
(2 :  6 :  1)
```

In the second addition, we are adding the point at infinity to the point $(2, 6)$ and obtaining $(2, 6)$. This is an example of $\infty + P = P$. We can multiply a point by an integer:

```
5*E([2,1])
(2 :  6 : 1)
```

We can list the multiples of a point in a range:

```
for i in range(10):
   print(i,i*E([2,6]))

(0, (0 : 1 : 0))
(1, (2 : 6 : 1))
(2, (3 : 1 : 1))
(3, (6 : 0 : 1))
(4, (3 : 6 : 1))
(5, (2 : 1 : 1))
(6, (0 : 1 : 0))
(7, (2 : 6 : 1))
(8, (3 : 1 : 1))
(9, (6 : 0 : 1))
```

The indentation of the `print` line is necessary since it indicates that this is iterated by the `for` command. To count the number of points on E:

```
E.cardinality()
6
```

Sage has a very fast point counting algorithm (due to Atkins, Elkies, and Schoof; it is much more sophisticated than listing the points, which would be infeasible). For example,

```
p=next_prime(10^50)
E1=EllipticCurve(IntegerModRing(p),[2,3])
n=E1.cardinality()
p, n, n-(p+1)
(100000000000000000000000000000000000000000000000151,
99999999999999999999999999112314733133761086232032,
-887685266866238913768120)
```

As you can see, the number of points on this curve (the second output line) is close to $p+1$. In fact, as predicted by Hasse's theorem, the difference $n-(p+1)$ (on the last output line) is less in absolute value than $2\sqrt{p} \approx 2 \times 10^{25}$.

Appendix E Answers and Hints for Selected Odd-Numbered Exercises

Chapter 2

1. 1. Among the shifts of *EVIRE*, there are two words: *arena* and *river*. Therefore, Anthony cannot determine where to meet Caesar.

2. 3. The decrypted message is 'cat'.

3. 5. The ciphertext is *QZNHOBXZD*. The decryption function is $21x + 9$.

4. 7. The encryption function is therefore $11x + 14$.

5. 9. The plaintext is *happy*.

6. 11. Successively encrypting with two affine functions is the same as encrypting with a single affine function. There is therefore no advantage of doing double encryption in this case.

7. 13. Mod 27, there are 486 keys. Mod 29, there are 812 keys.

8. 15.

    1. The possible values for α are 1,7,11,13,17,19,23,29.

    2. There are many such possible answers, for example $x = 1$ and $x = 4$ will work. These correspond to the letters 'b' and 'e'.

9. 19.

    2. The key is *AB*. The original plaintext is *BBBBBBABBB*.

10. 21. The key is probably *AB*.

11. 27. *EK IO IR NO AN HF YG BZ YB LF GM ZN AG ND OD VC MK*

12. 29. AAXFFGDGAFAX

Chapter 3

1. 1.

    1. One possibility: $1 = (-1) \cdot 101 + 6 \cdot 17$.

    2. 6

2. 3.

    1. $d = 13$

    2. Decryption is performed by raising the ciphertext to the 13th power mod 31.

3. 5.

    1. $x \equiv 22 \pmod{59}$, or
    $x \equiv 22, 81, 140, 199 \pmod{236}$.

    2. No solutions.

4. 7.

    1. If $n = ab$ with $1 < a, b < n$, then either $a \leq \sqrt{n}$ of $b \leq \sqrt{n}$. Otherwise, $ab > (\sqrt{n})(\sqrt{n}) = n$. If $1 < a <\leq \sqrt{n}$, let p be a prime factor of a.

    2. $\gcd\,(30030, 257) = 1$.

    3. No prime less than or equal to $\sqrt{257} \approx 16.03$ divides 257 because of the gcd calculation.

5. 9.

    1. The gcd is 257.

    2. $4883 = 257 \cdot 19$ and $4369 = 257 \cdot 17$.

6. 11.

    1. The gcd is 1.

    2. The gcd is 1.

    3. The gcd is 1.

7. 13.

    1. Use the Corollary in Section 3.2.

2. Imitate the proof of the Corollary in Section 3.2.

8. 17. $x \equiv 23 \pmod{70}$.

9. 19. The smallest number is 58 and the next smallest number is 118.

10. 21.

    1. $x \equiv 43, 56, 87, 100 \pmod{143}$.

    2. $x \equiv 44, 99 \pmod{143}$.

11. 23. The remainder is 8.

12. 25. The last two digits are 29.

13. 27. Use Fermat's theorem when $a \not\equiv 0 \pmod{p}$.

14. 29.

    1. $7^7 \equiv 3 \pmod{4}$.

    2. The last digit is 3.

15. 39.

    1. $\begin{pmatrix} 5 & 21 \\ 22 & 5 \end{pmatrix}$.

    2. $b \equiv 0, 2, 4, 6, 8, 10, 12, 16, 18, 20, 22, 24 \pmod{26}$
       .

16. 41. 2 and 13

17. 43.

    1. No solutions.

    2. There are solutions.

    3. No solutions.

Chapter 4

1. 1.
    1. 0.
    2. $P(M = cat \,|\, C = mxp) \neq P(M = cat)$.
2. 3. The conditional probability is 0. Affine ciphers do not have perfect secrecy.
3. 5.
    1. 1/2.
    2. $m_0 = HI$, $m_1 = BE$ is a possibility.
4. 11.
    1. Possible.
    2. Possible.
    3. Impossible.
5. 13. X

Chapter 5

1. 1.] The next four terms of the sequence are 1, 0, 0, 1.

2. 3. $c_0 = 1$, $c_1 = 0$, $c_2 = 0$, $c_3 = 1$.

3. 5. $c_0 = 2$ and $c_1 = 1$.

4. 7. $k_{n+2} \equiv k_n$.

5. 9. $c_0 = 4$ and $c_1 = 4$.

Chapter 6

1. 1. *eureka*.
2. 3. $\begin{pmatrix} 12 & 3 \\ 11 & 2 \end{pmatrix}$.
3. 5. $\begin{pmatrix} 7 & 2 \\ 13 & 5 \end{pmatrix}$.
4. 7.
    1. $\begin{pmatrix} 10 & 9 \\ 13 & 23 \end{pmatrix}$.
    2. $\begin{pmatrix} 10 & 19 \\ 13 & 19 \end{pmatrix}$.
5. 9. Use *aabaab*.
6. 13.
    1. Alice's method is more secure.
    2. Compatibility with single encryption.
7. 19. The jth and the $(j+1)$st blocks.

Chapter 7

1. 1.

    1. Switch left and right halves and use the same procedure as encryption. Then switch the left and right of the final output.

    2. After two rounds, the ciphertext alone lets you determine M_0 and therefore $M_1 \oplus K$, but not M_1 or K individually. If you also know the plaintext, you know M_1 are therefore can deduce K.

    3. Three rounds is very insecure.

2. 3. The ciphertext from the second message can be decrypted to yield the password.

3. 5.

    1. The keys for each round are all 1s, so using them in reverse order doesn't change anything.

    2. All 0s.

4. 7. Show that when P and K are used, the input to the S-boxes is the same as when P and K are used.

Chapter 8

1. 1.

    1. We have $W(4) = W(0) \oplus T(W(0)) = T(W(0))$. In the notation in Subsection 8.2.5, $a = b = c = d = 0$. The S-box yields $e = f = g = h = 99\,(\text{base } 10) = 01100011\,(\text{binary})$. The round constant is $r(4) = 00000010^0 = 00000001$. We have $e \oplus r(4) = 01100100$. Therefore,

$$W(4) = T(W(0)) = \begin{matrix} 01100100 \\ 01100011 \\ 01100011 \\ 01100011 \end{matrix}.$$

2. 3.

    1. Since addition in $GF(2^8)$ is the same as $\oplus$, we have $f(x_1) \oplus f(x_2) = \alpha(x_1 + x_2) = \alpha(x_3 + x_4) = f(x_3) \oplus f(x_4)$.

Chapter 9

1. 1. The plaintext is $1415 = no$.
2. 3.
    1. $d = 27$.
    2. Imitate the proof that RSA decryption works.
3. 5. The correct plaintext is 9.
4. 7. Use $d = 67$.
5. 9. Solve $de \equiv 1 \pmod{p-1}$.
6. 11. Bob computes b_1 with $bb_1 \equiv 1 \pmod{\phi(n)}$ and raises e to the power b_1.
7. 13. Divide the decryption by 2.
8. 15. It does not increase security.
9. 17. $e = 1$ sends plaintext, and $e = 2$ doesn't satisfy $\gcd(e, (p-1)(q-1)) = 1$.
10. 21. Compute a gcd.
11. 23. We have $(516107 \cdot 187722)^2 \equiv (2 \cdot 7)^2 \pmod{n}$.
12. 25. Combine the first three congruences. Ignore the fourth congruence.
13. 27. Use the Chinese Remainder Theorem to find x with $x \equiv 7 \pmod{p}$ and $x \equiv -7 \pmod{q}$.
14. 31. There are integers x and y such that $xe_A + ye_B = 1$.
15. 33. *HELLO*
16. 41. 12345.
17. 45. Find d' with $d'e \equiv 1 \pmod{12345}$.
18. 47.
    2. 1000000 messages.

Chapter 10

1. 1.

    1. $L_2(3) = 4$.
    2. $2^7 \equiv 11 \pmod{13}$.

2. 3.

    1. $6^5 \equiv 10$.
    2. x is odd.

3. 5. x is even.
4. 7. (a), (b) $L_2(24) = 72$.
5. 9. $x = 122$.
6. 13. Alice sends 10 to Bob and Bob sends 5 to Alice. Their shared secret is 6.
7. 15. Eve computes b_1 with $bb_1 \equiv 1 \pmod{p-1}$.

Chapter 11

1. 1. It is easy to construct collisions: $h(x) = h(x + p - 1)$, for example. (However, it is fairly quickly computed (though not fast enough for real use), and it is preimage resistant.)

2. 3.

    1. Finding square roots $\bmod pq$ is computationally equivalent to factoring.

    2. $h(x) \equiv h(n - x)$ for all x

3. 5. It satisfies (1), but not (2) and (3).

4. 9. (a) and (b) Let h be either of the hash functions. Given y of length n, we have $h(y \parallel 000 \cdots) = y$.

5. 11. Collision resistance.

Chapter 12

1. 1. $\frac{165}{288} \approx .573$
2. 5. $1 - e^{-7.3 \times 10^{-47}} \approx 0.$
3. 7. It is very likely that two choose the same prime.
4. 9. The probability is approximately $1 - e^{-500} \approx 0$ (or $1 - e^{-450} \approx 0$ if you take numbers of exactly 15 digits).

Chapter 13

1. 1. Use the congruence defining s to solve for a.
2. 3. See Section 13.4.
3. 7.
    1. Let $m_1 \equiv mr_1 r^{-1} \pmod{p-1}$.
4. 9. Imitate the proof for the usual ElGamal signatures.
5. 13. Eve notices that $\beta = r$.

Chapter 14

1. 1. Use the Birthday attack. Eve will probably factor some moduli.

2. 3.

    1. 0101 and 0110.

3. 5.

    1. Enigma does not encrypt a letter to itself, so DOG is impossible.

    2. If the first of two long plaintexts is encrypted with Enigma, it is very likely that at least one letter of the second plaintext will match a letter of the ciphertext. More precisely, each individual letter of the second plaintext that doesn't match the first plaintext has probability around 1/26 of matching, so the probability is $1 - (25/26)^{90} \approx 0.97$ that there is a match between the second plaintext and the ciphertext. Therefore, Enigma does not have ciphertext indistinguishability.

Chapter 15

1. 1. $K_{AB} = g_A(r_B) = 21$, $K_{AC} = g_A(r_C) = 7$ and $K_{BC} = g_B(r_C) = 29$.

2. 3. $a = 8, b = 8, c = 23$.

Chapter 16

1. 1.

    1. We have
    $r \equiv \alpha_1(cx + w) + \alpha_2 \equiv Hx + \alpha_1 w + \alpha_2 \pmod{q}$.
    Therefore

    $$g^r \equiv g^{w\alpha_1} g^{\alpha_2} g^{xH} \equiv g_w^{\alpha_1} g^{\alpha_2} g^{xH} \equiv a h^H \pmod{p}.$$

    2. Since $c_1 \equiv w + xc \pmod{q}$, we have
    $\alpha_1 c_1 \equiv w\alpha_1 + xH \pmod{q}$. Therefore,

    $$r \equiv \alpha_1 c_1 + \alpha_2 \equiv xH + w\alpha_1 + \alpha_2 \pmod{q}.$$

    Multiply by s and raise Ig_2 to these exponents to obtain

    $$(Ig_2)^r s \equiv (Ig_2)^{xsH} (Ig_2)^{ws\alpha_1} (Ig_2)^{s\alpha_2} \pmod{p}.$$

    This may be rewritten as

    $$A^r \equiv z^H b \pmod{p}.$$

    3. Since $r_1 \equiv usd + x_1$ and $r_2 \equiv sd + x_2 \pmod{q}$, we have

    $$g_1^{r_1} g_2^{r_2} \equiv (g_1^u g_2)^{sd} g_1^{x_1} g_2^{x_2} \equiv (I g_2^{sd} g_1^{x_1} g_2^{x_2} \equiv\equiv A^d B \pmod{p}.$$

2. 3.

    1. The only place r_1 and r_2 are used in the verification procedure is in checking that $g_1^{r_1} g_2^{r_2} \equiv A^d B$.

    2. The Spender spends the coin correctly once, using r_1, r_2 . The Spender then chooses any two random numbers r_1', r_2' with $r_1' + r_2' = r_1 + r_2$ and uses the coin with the Vendor, with r_1', r_2' in place of r_1, r_2. All the verification equations work.

3. 5. r_2 and r_2' are essentially random numbers (depending on hash values involving the clock), the probability is around $1/q$ that $r_2 \equiv r_2' \pmod{q}$. Since q is large, $1/q$ is small.

4. 7. Fred only needs to keep the hash of the file on his own computer.

Chapter 17

1. 1. One possibility is to take $p = 7$ and choose the polynomial $s(x) = 5 + x \pmod 7$ (where 5 is chosen randomly). Then the secret value is $s(0) = 5$, and we may choose the shares (1,6), (2,0), (3,1), and (4,2).

2. 3. *=63

3. 5. The polynomial is

   $$8\frac{(x-3)(x-5)}{(1-3)(1-5)} + 10\frac{(x-1)(x-5)}{(3-1)(3-5)} + 11\frac{(x-1)(x-3)}{(5-1)(5-3)}.$$

   The secret is $13 \pmod{17}$.

4. 7. $M = 77, 57, 37, 17.$

5. 9. Take a (10, 30) scheme and give the general 10 shares, the colonels five shares each, and the clerks two each.

6. 11. Start by splitting the launch code into three equal components using a three-party secret splitting scheme.

Chapter 19

1. 3.

    1. Nelson computes a square root of $y \bmod p$ and $\bmod q$, then combines them to obtain a square root of $y \bmod n$.

    2. Use the $x^2 \equiv y^2$ factorization method.

    3. No.

2. 5.

    Step 4: Victor randomly chooses $i = 1$ or 2 and asks Peggy for r_i.

    Step 5: Victor checks that $x_i \equiv r_i^e \pmod n$.

    They repeat steps 1 through 5 at least 7 times (since $(1/2)^7 < .01$).

3. 7.

    1. One way: Step 4: Victor chooses $i \neq j$ at random and asks for r_i and r_j. Then five repetitions are enough. Another way: Victor asks for only one of the r_k's. Then twelve repetitions suffice.

    2. Choose r_1, r_2. Then solve for r_3.

Chapter 20

1. 1. $H(X_1) = H(X_2) = 1$, and $H(X_1, X_2) = 2$.
2. 3. $H(X) = 2$.
3. 5. $H(Y) \leq H(X)$.
4. 7.
    1. 2.9710
    2. 2.9517
5. 9.
    1. $H(P) = -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right)$.
    2. This system matches up with the one-time pad, and hence $H(P|C) = H(P)$.
6. 13.
    1. $H(X) = \log_2 36$.
    2. Use Fermat's Theorem to obtain $H(Y) = 0$.

Chapter 21

1. 3.
    1. $(3, 2)$, $(3, 5)$, $(5, 2)$, $(5, 5)$, $(6, 2)$, $(6, 5)$, ∞.
    2. $(3, 5)$.
    3. $(5, 2)$.
2. 5.
    1. $(2, 2)$.
    2. She factors 35.
3. 7. One example: $y^2 \equiv x^3 + 17$.
4. 9.
    1. $3Q = (-1, 0)$.
5. 15. $y \equiv \pm 4, \pm 11 \pmod{35}$

Chapter 22

1. 3. Compute $\tilde{e}(aA, B)$ and $\tilde{e}(A, bB)$.

2. 7.

    1. Eve knows rP_0, P_1, k. She computes

    $$\tilde{e}(rP_0, P_1)^k = \tilde{(}kP_0, P_1)^r = \tilde{e}(H_1(\text{bob@computer.com}), P_1)^r = g^r.$$

    Eve now computes $H_2(g^r)$ and XORs it with t to get m.

3. 9. See Claim 2 in Section 9.1.

Chapter 23

1. 1. The basis $(0, 1)$, $(-2, 0)$ is reduced. The vector $(0, 1)$ is a shortest vector.

Chapter 24

1. 1.

    1. The original message is 0,1,0,0.

    2. The original message is 0,1,0,1.

2. 3.

    1. $n = 5$ and $k = 2$.

    2. $$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

    3. $(0, 0, 0, 0, 0), (1, 1, 0, 1, 0), (1, 0, 1, 0, 1), (0, 1, 1, 1, 1)$.

    4. $R = \frac{\log_2(4)}{5} = 0.4$.

3. 5.

    2. $d(C) = 2$.

4. 13. $1 + X + X^2 + X^3$ is in C and the other two polynomials are not in C.

5. 19. The error is in the 3rd position. The corrected vector is (1,0,0,1,0,1,1).

Chapter 25

1. 1.

    1. The period is 4.
    2. $m = 8$.
    3. $r = 4$.

Appendix F Suggestions for Further Reading

For the history of cryptography, see [Kahn] and [Bauer].

For additional treatment of topics in the present book, and many other topics, see [Stinson], [Stinson1], [Schneier], [Mao], and [Menezes et al.]. These books also have extensive bibliographies.

An approach emphasizing algebraic methods is given in [Koblitz].

For the theoretical foundations of cryptology, see [Goldreich1] and [Goldreich2]. See [Katz-Lindell] for an approach based on security proofs.

Books that are oriented toward protocols and practical network security include [Stallings], [Kaufman et al.], and [Aumasson].

For a guidelines on properly applying cryptographic algorithms, the reader is directed to [Ferguson-Schneier]. For a general discussion on securing computing platforms, see [Pfleeger-Pfleeger].

The Internet, of course, contains a wealth of information about cryptographic issues. The Cryptology ePrint Archive server at `http://eprint.iacr.org/` contains very recent research. Also, the conference proceedings CRYPTO, EUROCRYPT, and ASIACRYPT (published in Springer-Verlag's Lecture Notes in Computer Science series) contain many interesting reports on recent developments.

Bibliography

- [Adrian et al.] Adrian et al., "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice," `https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf`.

- [Agrawal et al.] M. Agrawal, N. Kayal, and N. Saxena, “PRIMES is in P,” Annals of Math. 160 (2004), 781–793.

- [Alford et al.] W. R. Alford, A. Granville, and C. Pomerance, “On the difficulty of finding reliable witnesses,” Algorithmic Number Theory, Lecture Notes in Computer Science 877, Springer-Verlag, 1994, pp. 1–16.

- [Alford et al. 2] W. R. Alford, A. Granville, and C. Pomerance, “There are infinitely many Carmichael numbers,” Annals of Math. 139 (1994), 703–722.

- [Atkins et al.] D. Atkins, M. Graff, A. Lenstra, P. Leyland, “The magic words are squeamish ossifrage,” Advances in Cryptology – ASIACRYPT ’94, Lecture Notes in Computer Science 917, Springer-Verlag, 1995, pp. 263–277.

- [Aumasson] J-P. Aumasson, Serious Cryptography: A Practical Introduction to Modern Encryption, No Starch Press, 2017.

- [Bard] G. Bard, Sage for Undergraduates, Amer. Math. Soc., 2015.

- [Bauer] C.Bauer, Secret History: The Story of Cryptology, CRC Press, 2013.

- [Beker-Piper] H. Beker and F. Piper, Cipher Systems: The Protection of Communications, Wiley-Interscience, 1982.

- [Bellare et al.] M. Bellare, R. Canetti, and H. Krawczyk, “Keying Hash Functions for Message Authentication,” Advances in Cryptology (Crypto 96 Proceedings), Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

- [Bellare-Rogaway] M. Bellare and P. Rogaway, “Random oracles are practical: a paradigm for designing efficient protocols,” First ACM Conference on Computer and Communications Security, ACM Press, New York, 1993, pp. 62–73.

- [Bellare-Rogaway2] M. Bellare and P. Rogaway, “Optimal asymmetric encryption,” Advances in Cryptology – EUROCRYPT ’94, Lecture Notes in Computer Science 950, Springer-Verlag, 1995, pp. 92–111.

- [Berlekamp] E. Berlekamp, Algebraic Coding Theory, McGraw-Hill, 1968.
- [Bernstein et al.] Post-Quantum Cryptography, Bernstein, Daniel J., Buchmann, Johannes, Dahmen, Erik (Eds.), Springer-Verlag, 2009.
- [Bitcoin] bitcoin, https://bitcoin.org/en/
- [Blake et al.] I. Blake, G. Seroussi, N. Smart, Elliptic Curves in Cryptography, Cambridge University Press, 1999.
- [Blom] R. Blom, "An optimal class of symmetric key generation schemes," Advances in Cryptology – EUROCRYPT'84, Lecture Notes in Computer Science 209, Springer-Verlag, 1985, pp. 335–338.
- [Blum-Blum-Shub] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," SIAM Journal of Computing 15(2) (1986), 364–383.
- [Boneh] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," Amer. Math. Soc. Notices 46 (1999), 203–213.
- [Boneh et al.] D. Boneh, G. Durfee, and Y. Frankel, "An attack on RSA given a fraction of the private key bits," Advances in Cryptology – ASIACRYPT '98, Lecture Notes in Computer Science 1514, Springer-Verlag, 1998, pp. 25–34.
- [Boneh-Franklin] D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," Advances in Cryptology – CRYPTO '01, Lecture Notes in Computer Science 2139, Springer-Verlag, 2001, pp. 213–229.
- [Boneh-Joux-Nguyen] D. Boneh, A. Joux, P. Nguyen, "Why textbook ElGamal and RSA encryption are insecure," Advances in Cryptology – ASIACRYPT '00, Lecture Notes in Computer Science 1976, Springer-Verlag, 2000, pp. 30–43.
- [Brands] S. Brands, "Untraceable off-line cash in wallets with observers," Advances in Cryptology – CRYPTO'93, Lecture Notes in Computer Science 773, Springer-Verlag, 1994, pp. 302–318.
- [Campbell-Wiener] K. Campbell and M. Wiener, "DES is not a group," Advances in Cryptology – CRYPTO '92, Lecture Notes in Computer Science 740, Springer-Verlag, 1993, pp. 512–520.
- [Canetti et al.] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," Proceedings of the thirtieth annual ACM symposium on theory of computing, ACM Press, 1998, pp. 209–218.
- [Chabaud] F. Chabaud, "On the security of some cryptosystems based on error-correcting codes," Advances in Cryptology –

EUROCRYPT'94, Lecture Notes in Computer Science 950, Springer-Verlag, 1995, pp. 131–139.

- [Chaum et al.] D. Chaum, E. van Heijst, and B. Pfitzmann, "Cryptographically strong undeniable signatures, unconditionally secure for the signer," Advances in Cryptology – CRYPTO '91, Lecture Notes in Computer Science 576, Springer-Verlag, 1992, pp. 470–484.

- [Cohen] H. Cohen, A Course in Computational Number Theory, Springer-Verlag, 1993.

- [Coppersmith1] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," IBM Journal of Research and Development, vol. 38, no. 3, May 1994, pp. 243–250.

- [Coppersmith2] D. Coppersmith, "Small solutions to polynomial equations, and low exponent RSA vulnerabilities," J. Cryptology 10 (1997), 233–260.

- [Cover-Thomas] T. Cover and J. Thomas, Elements of Information Theory, Wiley Series in Telecommunications, 1991.

- [Crandall-Pomerance] R. Crandall and C. Pomerance, Prime Numbers, a Computational Perspective, Springer-Telos, 2000.

- [Crosby et al.] Crosby, S. A., Wallach, D. S., and Riedi, R. H. "Opportunities and limits of remote timing attacks," ACM Trans. Inf. Syst. Secur. 12, 3, Article 17 (January 2009), 29 pages.

- [Damgård et al.] I. Damgård, P. Landrock, and C. Pomerance, "Average case error estimates for the strong probable prime test," Mathematics of Computation 61 (1993), 177–194.

- [Dawson-Nielsen] E. Dawson and L. Nielsen, "Automated Cryptanalysis of XOR Plaintext Strings," Cryptologia 20 (1996), 165–181.

- [Diffie-Hellman] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Trans. in Information Theory, 22 (1976), 644–654.

- [Diffie-Hellman2] W. Diffie and M. Hellman, "Exhaustive cryptanalysis of the NBS data encryption standard," Computer 10(6) (June 1977), 74–84

- [Ekert-Josza] A. Ekert and R. Jozsa, "Quantum computation and Shor's factoring algorithm," Reviews of Modern Physics, 68 (1996), 733–753.

- [FIPS 186-2] FIPS 186-2, Digital signature standard (DSS), Federal Information Processing Standards Publication 186, U.S. Dept. of Commerce/National Institute of Standards and Technology, 2000.

- [FIPS 202] FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Information Processing Standards Publication 202, U.S. Dept. of Commerce/National Institute of Standards and Technology, 2015, available at http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

- [Ferguson-Schneier] N. Ferguson and B. Schneier, Practical Cryptography, Wiley, 2003.

- [Fortune-Merritt] S. Fortune and M. Merritt, "Poker Protocols," Advances in Cryptology – CRYPTO'84, Lecture Notes in Computer Science 196, Springer-Verlag, 1985, pp. 454–464.

- [Gaines] H. Gaines, Cryptanalysis, Dover Publications, 1956.

- [Gallager] R. G. Gallager, Information Theory and Reliable Communication, Wiley, 1969.

- [Genkin et al.] D. Genkin, A. Shamir, and E. Tromer, "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis," December 18, 2013, available at www.cs.tau.ac.il/~tromer/papers/acoustic-20131218.pdf

- [Gilmore] Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design, Electronic Frontier Foundation, J. Gilmore (editor), O'Reilly and Associates, 1998.

- [Girault et al.] M. Girault, R. Cohen, and M. Campana, "A generalized birthday attack," Advances in Cryptology – EUROCRYPT'88, Lecture Notes in Computer Science 330, Springer-Verlag, 1988, pp. 129–156.

- [Goldreich1] O. Goldreich, Foundations of Cryptography: Volume 1, Basic Tools, Cambridge University Press, 2001.

- [Goldreich2] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press, 2004.

- [Golomb] S. Golomb, Shift Register Sequences, 2nd ed., Aegean Park Press, 1982.

- [Hankerson et al.] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer-Verlag, 2004.

- [Hardy-Wright] G. Hardy and E. Wright, An Introduction to the Theory of Numbers. Fifth edition, Oxford University Press, 1979.

- [Heninger et al.] N. Heninger, Z. Durumeric, E. Wustrow, J. A. Halderman, "Mining your and : Detection of widespread weak key in network devices," Proc. 21st USENIX Security Symposium, Aug. 2012; available at https://factorable.net.

- [HIP] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," May 2006; available at https://

tools.ietf.org/html/rfc4423

- [Joux] A. Joux, “Multicollisions in iterated hash functions. Application to cascaded constructions,” Advances in Cryptology – CRYPTO 2004, Lecture Notes in Computer Science 3152, Springer, 2004, pp. 306–316.
- [Kahn] D. Kahn, The Codebreakers, 2nd ed., Scribner, 1996.
- [Kaufman et al.] C. Kaufman, R. Perlman, M. Speciner, Private Communication in a Public World. Second edition, Prentice Hall PTR, 2002.
- [Kilian-Rogaway] J. Kilian and P. Rogaway, “How to protect DES against exhaustive key search (an analysis of DESX),” J. Cryptology 14 (2001), 17–35.
- [Koblitz] N. Koblitz, Algebraic Aspects of Cryptography, Springer-Verlag, 1998.
- [Kocher] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” Advances in Cryptology – CRYPTO ’96, Lecture Notes in Computer Science 1109, Springer, 1996, pp. 104–113.
- [Kocher et al.] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” Advances in Cryptology – CRYPTO ’99, Lecture Notes in Computer Science 1666, Springer, 1999, pp. 388–397.
- [Konikoff-Toplosky] J. Konikoff and S. Toplosky, “Analysis of Simplified DES Algorithms,” Cryptologia 34 (2010), 211–224.
- [Kozaczuk] W. Kozaczuk, Enigma: How the German Machine Cipher Was Broken, and How It Was Read by the Allies in World War Two; edited and translated by Christopher Kasparek, Arms and Armour Press, London, 1984.
- [KraftW] J. Kraft and L. Washington, An Introduction to Number Theory with Cryptography, CRC Press, 2018.
- [Lenstra et al.] A. Lenstra, X. Wang, B. de Weger, “Colliding X.509 certificates,” preprint, 2005.
- [Lenstra2012 et al.] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, “Ron was wrong, Whit is right,” https://eprint.iacr.org/2012/064.pdf.
- [Lin-Costello] S. Lin and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice Hall, 1983.
- [MacWilliams-Sloane] F. J. MacWilliams and N. J. A. Sloane, The Theory of Error-Correcting Codes, North-Holland, 1977.
- [Mantin-Shamir] I. Mantin and A. Shamir, “A practical attack on broadcast RC4,” In: FSE 2001, 2001.

- [Mao] W. Mao, Modern Cryptography: Theory and Practice, Prentice Hall PTR, 2004.
- [Matsui] M. Matsui,“Linear cryptanalysis method for DES cipher,” Advances in Cryptology – EUROCRYPT’93, Lecture Notes in Computer Science 765, Springer-Verlag, 1994, pp. 386–397.
- [Menezes et al.] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997.
- [Merkle-Hellman] R. Merkle and M. Hellman, “On the security of multiple encryption,” Comm. of the ACM 24 (1981), 465–467.
- [Mikle] O. Mikle, “Practical Attacks on Digital Signatures Using MD5 Message Digest,” Cryptology ePrint Archive, Report 2004/356, http://eprint.iacr.org/2004/356, 2nd December 2004.
- [Nakamoto] S. Nakamoto, ”Bitcoin: A Peer-to-peer Electronic Cash System,” available at https://bitcoin.org/bitcoin.pdf
- [Narayanan et al.] A. Narayanan, J. Bonneau, E. Felten, A. Miller, S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction (with a preface by Jeremy Clark), Princeton University Press 2016.
- [Nelson-Gailly] M. Nelson and J.-L. Gailly, The Data Compression Book, M&T Books, 1996.
- [Nguyen-Stern] P. Nguyen and J. Stern, “The two faces of lattices in cryptology,” Cryptography and Lattices, International Conference, CaLC 2001, Lecture Notes in Computer Science 2146, Springer-Verlag, 2001, pp. 146–180.
- [Niven et al.] I. Niven, H. Zuckerman, and H. Montgomery, An Introduction to the Theory of Numbers, Fifth ed., John Wiley & Sons, Inc., New York, 1991.
- [Okamoto-Ohta] T. Okamoto and K. Ohta, “Universal electronic cash,” Advances in Cryptology – CRYPTO’91, Lecture Notes in Computer Science 576, Springer-Verlag, 1992, pp. 324–337.
- [Pfleeger-Pfleeger] C. Pfleeger, S. Pfleeger, Security in Computing. Third edition, Prentice Hall PTR, 2002.
- [Pomerance] C. Pomerance, “A tale of two sieves,” Notices Amer. Math. Soc. 43 (1996), no. 12, 1473–1485.
- [Quisquater et al.] J.-J. Quisquater and L. Guillou, “How to explain zero-knowledge protocols to your children,” Advances in Cryptology – CRYPTO ’89, Lecture Notes in Computer Science 435, Springer-Verlag, 1990, pp. 628–631.
- [Rieffel-Polak] E. Rieffel and W. Polak, “An Introduction to Quantum Computing for Non-Physicists,” available at

xxx.lanl.gov/abs/quant-ph/9809016.

- [Rosen] K. Rosen, Elementary Number Theory and its Applications. Fourth edition, Addison-Wesley, Reading, MA, 2000.
- [Schneier] B. Schneier, Applied Cryptography, 2nd ed., John Wiley, 1996.
- [Shannon1] C. Shannon, “Communication theory of secrecy systems,” Bell Systems Technical Journal 28 (1949), 656–715.
- [Shannon2] C. Shannon, “A mathematical theory of communication,” Bell Systems Technical Journal, 27 (1948), 379–423, 623–656.
- [Shoup] V. Shoup, “OAEP Reconsidered,” CRYPTO 2001 (J. Killian (ed.)), Springer LNCS 2139, Springer-Verlag Berlin Heidelberg, 2001, pp. 239–259.
- [Stallings] W. Stallings, Cryptography and Network Security: Principles and Practice, 3rd ed., Prentice Hall, 2002.
- [Stevens et al.] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov, “The first collision for full SHA-1,” `https://shattered.io/static/shattered.pdf`.
- [Stinson] D. Stinson, Cryptography: Theory and Practice. Second edition, Chapman & Hall/CRC Press, 2002.
- [Stinson1] D. Stinson, Cryptography: Theory and Practice, CRC Press, 1995.
- [Thompson] T. Thompson, From Error-Correcting Codes through Sphere Packings to Simple Groups, Carus Mathematical Monographs, number 21, Mathematical Assoc. of America, 1983.
- [van der Lubbe] J. van der Lubbe, Basic Methods of Cryptography, Cambridge University Press, 1998.
- [van Oorschot-Wiener] P. van Oorschot and M. Wiener, “A known-plaintext attack on two-key triple encryption,” Advances in Cryptology – EUROCRYPT ’90, Lecture Notes in Computer Science 473, Springer-Verlag, 1991, pp. 318–325.
- [Wang et al.] X. Wang, D. Feng, X. Lai, H. Yu, “Collisions for hash functions MD-4, MD-5, HAVAL-128, RIPEMD,” preprint, 2004.
- [Wang et al. 2] X. Wang, Y. Yin, H. Yu, “Finding collisions in the full SHA1,” to appear in CRYPTO 2005.
- [Washington] L. Washington, Elliptic Curves: Number Theory and Cryptography, Chapman & Hall/CRC Press, 2003.
- [Welsh] D. Welsh, Codes and Cryptography, Oxford, 1988.

- [Wicker] S. Wicker, Error Control Systems for Digital Communication and Storage, Prentice Hall, 1995.
- [Wiener] M. Wiener, “Cryptanalysis of short RSA secret exponents,” IEEE Trans. Inform. Theory, 36 (1990), 553–558.
- [Williams] H. Williams, Edouard Lucas and Primality Testing, Wiley-Interscience, 1998.
- [Wu1] T. Wu, “The secure remote password protocol,” In: Proc. of the Internet Society Network and Distributed Security Symposium, 97–111, March 1998.
- [Wu2] T. Wu, “SRP-6: Improvements and refinements to the Secure Remote Password protocol,” 2002; available through `http://srp.stanford.edu/design.html`

Index

Contents

Landmarks

26. 12
27. 13
28. 14
29. 15
30. 16
31. 17
32. 18
33. 19
34. 20
35. 21
36. 22
37. 23
38. 24
39. 25
40. 26
41. 27
42. 28
43. 29
44. 30
45. 31
46. 32
47. 33
48. 34
49. 35
50. 36
51. 37
52. 38
53. 39
54. 40
55. 41
56. 42
57. 43
58. 44
59. 45
60. 46
61. 47
62. 48
63. 49
64. 50
65. 51
66. 52
67. 53
68. 54
69. 55
70. 56
71. 57
72. 58
73. 59
74. 60
75. 61
76. 62
77. 63
78. 64
79. 65

80. 66
81. 67
82. 68
83. 69
84. 70
85. 71
86. 72
87. 73
88. 74
89. 75
90. 76
91. 77
92. 78
93. 79
94. 80
95. 81
96. 82
97. 83
98. 84
99. 85
100. 86
101. 87
102. 88
103. 89
104. 90
105. 91
106. 92
107. 93
108. 94
109. 95
110. 96
111. 97
112. 98
113. 99
114. 100
115. 101
116. 102
117. 103
118. 104
119. 105
120. 106
121. 107
122. 108
123. 109
124. 110
125. 111
126. 112
127. 113
128. 114
129. 115
130. 116
131. 117
132. 118
133. 119

134. 120
135. 121
136. 122
137. 123
138. 124
139. 125
140. 126
141. 127
142. 128
143. 129
144. 130
145. 131
146. 132
147. 133
148. 134
149. 135
150. 136
151. 137
152. 138
153. 139
154. 140
155. 141
156. 142
157. 143
158. 144
159. 145
160. 146
161. 147
162. 148
163. 149
164. 150
165. 151
166. 152
167. 153
168. 154
169. 155
170. 156
171. 157
172. 158
173. 159
174. 160
175. 161
176. 162
177. 163
178. 164
179. 165
180. 166
181. 167
182. 168
183. 169
184. 170
185. 171
186. 172
187. 173

188. 174
189. 175
190. 176
191. 177
192. 178
193. 179
194. 180
195. 181
196. 182
197. 183
198. 184
199. 185
200. 186
201. 187
202. 188
203. 189
204. 190
205. 191
206. 192
207. 193
208. 194
209. 195
210. 196
211. 197
212. 198
213. 199
214. 200
215. 201
216. 202
217. 203
218. 204
219. 205
220. 206
221. 207
222. 208
223. 209
224. 210
225. 211
226. 212
227. 213
228. 214
229. 215
230. 216
231. 217
232. 218
233. 219
234. 220
235. 221
236. 222
237. 223
238. 224
239. 225
240. 226
241. 227

242. 228
243. 229
244. 230
245. 231
246. 232
247. 233
248. 234
249. 235
250. 236
251. 237
252. 238
253. 239
254. 240
255. 241
256. 242
257. 243
258. 244
259. 245
260. 246
261. 247
262. 248
263. 249
264. 250
265. 251
266. 252
267. 253
268. 254
269. 255
270. 256
271. 257
272. 258
273. 259
274. 260
275. 261
276. 262
277. 263
278. 264
279. 265
280. 266
281. 267
282. 268
283. 269
284. 270
285. 271
286. 272
287. 273
288. 274
289. 275
290. 276
291. 277
292. 278
293. 279
294. 280
295. 281

296. 282
297. 283
298. 284
299. 285
300. 286
301. 287
302. 288
303. 289
304. 290
305. 291
306. 292
307. 293
308. 294
309. 295
310. 296
311. 297
312. 298
313. 299
314. 300
315. 301
316. 302
317. 303
318. 304
319. 305
320. 306
321. 307
322. 308
323. 309
324. 310
325. 311
326. 312
327. 313
328. 314
329. 315
330. 316
331. 317
332. 318
333. 319
334. 320
335. 321
336. 322
337. 323
338. 324
339. 325
340. 326
341. 327
342. 328
343. 329
344. 330
345. 331
346. 332
347. 333
348. 334
349. 335

350. 336
351. 337
352. 338
353. 339
354. 340
355. 341
356. 342
357. 343
358. 344
359. 345
360. 346
361. 347
362. 348
363. 349
364. 350
365. 351
366. 352
367. 353
368. 354
369. 355
370. 356
371. 357
372. 358
373. 359
374. 360
375. 361
376. 362
377. 363
378. 364
379. 365
380. 366
381. 367
382. 368
383. 369
384. 370
385. 371
386. 372
387. 373
388. 374
389. 375
390. 376
391. 377
392. 378
393. 379
394. 380
395. 381
396. 382
397. 383
398. 384
399. 385
400. 386
401. 387
402. 388
403. 389

404. 390
405. 391
406. 392
407. 393
408. 394
409. 395
410. 396
411. 397
412. 398
413. 399
414. 400
415. 401
416. 402
417. 403
418. 404
419. 405
420. 406
421. 407
422. 408
423. 409
424. 410
425. 411
426. 412
427. 413
428. 414
429. 415
430. 416
431. 417
432. 418
433. 419
434. 420
435. 421
436. 422
437. 423
438. 424
439. 425
440. 426
441. 427
442. 428
443. 429
444. 430
445. 431
446. 432
447. 433
448. 434
449. 435
450. 436
451. 437
452. 438
453. 439
454. 440
455. 441
456. 442
457. 443

458. 444
459. 445
460. 446
461. 447
462. 448
463. 449
464. 450
465. 451
466. 452
467. 453
468. 454
469. 455
470. 456
471. 457
472. 458
473. 459
474. 460
475. 461
476. 462
477. 463
478. 464
479. 465
480. 466
481. 467
482. 468
483. 469
484. 470
485. 471
486. 472
487. 473
488. 474
489. 475
490. 476
491. 477
492. 478
493. 479
494. 480
495. 481
496. 482
497. 483
498. 484
499. 485
500. 486
501. 487
502. 488
503. 489
504. 490
505. 491
506. 492
507. 493
508. 494
509. 495
510. 496
511. 497

512. 498
513. 499
514. 500
515. 501
516. 502
517. 503
518. 504
519. 505
520. 506
521. 507
522. 508
523. 509
524. 510
525. 511
526. 512
527. 513
528. 514
529. 515
530. 516
531. 517
532. 518
533. 519
534. 520
535. 521
536. 522
537. 523
538. 524
539. 525
540. 526
541. 527
542. 528
543. 529
544. 530
545. 531
546. 532
547. 533
548. 534
549. 535
550. 536
551. 537
552. 538
553. 539
554. 540
555. 541
556. 542
557. 543
558. 544
559. 545
560. 546
561. 547
562. 548
563. 549
564. 550
565. 551

566. 552
567. 553
568. 554
569. 555
570. 556
571. 557
572. 558
573. 559
574. 560
575. 561
576. 562
577. 563
578. 564
579. 565
580. 566
581. 567
582. 568
583. 569
584. 570
585. 571
586. 572
587. 573
588. 574
589. 575
590. 576
591. 577
592. 578
593. 579
594. 580
595. 581
596. 582
597. 583
598. 584
599. 585
600. 586
601. 587
602. 588
603. 589
604. 590
605. 591
606. 592
607. 593
608. 594
609. 595
610. 596
611. 597
612. 598
613. 599
614. 600
615. 601
616. 602
617. 603
618. 604
619. 605

620. 606
621. 607
622. 608
623. 609
624. 610
625. 611
626. 612
627. 613
628. 614
629. 615
630. 616
631. 617
632. 618
633. 619
634. 620
635. 621

Long description

The illustration shows Alice is connected to encrypt with an arrow labeled as plaintext pointed towards encrypt. Encrypt is connected to decrypt with an arrow labeled as ciphertext. Decrypt is connected to Bob. Encrypt is connected with Encryption Key with an arrow pointed downward Encrypt. Decrypt is connected with Decryption Key with an arrow pointed downward Decrypt. The arrow between Encrypt and Decrypt is connected with Eve with an arrow pointed downward.

Long description

The illustration shows a box with three rows. In the rows, the letters are shown with their frequencies.

In the first row, a: .082, b: .015, c: .028, d: .043, e: .127, f: .022, g: .020, h: .061, i: .070, j:.002. In the second row, k: .008, l: .040, m: .024, n:.067, o: .075, p: .019, q: .001, r: .060, s: .063, t: .091.

In the third row, u: .028, v: .010, w: .023, x: .001, y: .020, z: .001.

Long description

The illustration shows a box. In the box, the letters are shown with their frequencies, e: .127, t: .091, a: .082, o: .075, i: .070, n: .067, s: .063, h: .061, r: .060.

Long description

The table shows counting diagrams with 9 rows labeled as W, B, R, S, I, V, A, P, N and 9 columns labeled as W, B, R, S, I, V, A, P, N. The table shows various corresponding values for each row and each column.

Long description

The table shows 5 versus 5 matrix table with row labeled as A, D, F, G, X and columns labeled as A, D, F, G, X. The table shows various corresponding values for each column and each row.

Long description

The table shows a matrix table with 5 columns labeled as R, H, E, I, N. The table shows various corresponding various values for each column.

Long description

The table shows a matrix table with 5 columns labeled as E, H, I, N, R. The table shows various corresponding values for each column.

Long description

The illustration shows four rectangular boxes labeled as R, L, M, N respectively placed vertically parallel to each other. The illustration shows a square box labeled as S beyond the four boxes. S is connected to N, N is connected to M, M is connected to L, L is connected to R with an arrow and vice versa. Beyond S, a rectangular box labeled as K, keyboard is connected to S with an arrow at the very lower position and a bar with four bulbs labeled as glow lamps is connected to S with an arrow at the very upper position.

Long description

A table shows 6 rows and 6 columns for addition mod 6. The table contains various values for each rows and columns.

Long description

A table shows 6 rows and 6 columns for multiplication mod 6. The table contains various values for each rows and columns.

Long description

The table shows various symbols and their decimal and binary number in five numbers of rows. The first row contains the symbol exclamation : 33 : 0100001, double inverted : 34 : 0100010, hash : 35 : 0100011, Dollar : 36 : 0100100, percentage : 37 : 0100101, and : 38 : 0100110, single inverted : 39 : 0100111. The second row contains the symbols open bracket : 40 : 0101000, closed bracket : 41 : 0101001, asterisk : 42 : 0101010, plus : 43 : 0101011, coma : 44 : 0101100, minus : 45 : 0101101, dot : 46 : 0101110, slash: 47 : 0101111. The third row contains 0 : 48 : 0110000, 1 : 49 : 011000, 2 : 50 : 0110010, 3 : 51 : 0110011, 4 : 52 : 0110100, 5 : 53 : 0110101, 6 : 54 : 011010, 7 : 55 : 0110111. The fourth row contains 8 : 56 : 0111000, 9 : 57 : 0111001, colon : 58 : 0111010, semi colon : 59 : 0111011, less than : 60 : 0111100, equal : 61 : 0111101, greater than : 62 : 0111110, question mark : 63 : 0111111. The last row contains the symbols at the rate : 64 : 1000000, A : 65 : 1000001, B : 66 : 1000010, C : 67 : 1000011, D : 68 : 1000100, E : 69 : 1000100, F : 70 : 1000110, G : 71 : 1000111.

Long description

The block diagram of a stream cipher encryption shows an arrow pointed to x subscript n plus 2, x subscript n plus 2 is connected to x subscript n plus 1 with an arrow, and then x subscript n plus 1 is connected to x subscript n with an arrow which further connects to an adder. Then, an arrow points towards p subscript n plus 2 which is connected to p subscript n plus 1 with an arrow, and then p subscript n plus 1 is points to p subscript n with an arrow which further connects to an adder. The output of the adder is fed to c subscript n, from which an output is obtained.

Long description

The image of a linear feedback shift register satisfying x subscript n plus 3 equals x subscript n plus 1 plus x subscript n shows that x subscript n plus 2 is connected to x subscript n plus 1 with an arrow which is further connected to x subscript n with an arrow. x subscript n plus 2 and x subscript n is further fed to an adder and the output of the adder is again fed to x subscript n plus 2. x subscript n and plaintext are fed to another adder from which the output ciphertext is obtained.

Long description

The first block contains P subscript 1 which represents the first plain text which is then connected to block E subscript K which represents encryption function. Block C subscript 0 which represents initialization vector is connected to block P subscript 1 and E subscript K through a connector. Block E subscript K is then connected to block C subscript 1 which represents the first ciphertext. Block that contains P subscript 2 which represents second plain text is connected to E subscript K through a connector. Block C subscript 1 is connected to the connector that connects block P subscript 2 and block E subscript K. Again, the block E Subscript K connects the block C subscript 2 which represents the second ciphertext from where the arrow connects to dot-dot-dot.

Long description

The illustration shows eleven blocks connected to each other. The first block labeled as X subscript 1 connects to the second block E subscript which again, connects to the block that contains O subscript 1 in a register that denotes 8 leftmost bits that connect with a connector P subscript 1 that shows 8 bits and C subscript 1 is connected to the connector. Again, C subscript 1 is connected to register that contains C subscript 1 present in rightmost side of the block X subscript 2 that connects the block X subscript 2 and shifts the rightmost register labeled as drop to the leftmost side that is represented by downward arrows which connects to the block E subscript K which again connects to the block that contains O subscript 2 in a register that denotes 8 leftmost bits that connects with a connector P subscript 2 that shows 8 bits and C subscript 2 is connected to the connector. Again C subscript 2 is connected to register that contains C subscript 2 present in rightmost side of the block X subscript 2 that connects the block X subscript 3 and shifts the rightmost register labeled as drop to the leftmost side that is represented by downward arrows which connects to the block E subscript K which again connects to the block that contains O subscript 3 in a register that denotes 8 leftmost bits that connects with a connector P subscript 2 that shows 8 bits and C subscript 3 is connected to the connector which is further connected to dot-dot-dot. The first block X subscript 1 is connected to the next X subscript 1 block that contains O subscript 1 and X subscript 2 block is connected to the next X subscript 2 block that contains O subscript 2.

Long description

The illustration shows eleven blocks connected to each other. The first block labeled as X subscript 1 connects to the second block E subscript which again connects to the block that contains O subscript 1 in a register that denotes 8 leftmost bits that connect with a connector P subscript 1 that shows 8 bits and C subscript 1 is connected to the connector. Again, O subscript 1 is connected to register that contains O subscript 1 present in rightmost side of the block X subscript 1 that connects the block X subscript 2 and shifts the rightmost register labeled as drop to the leftmost side that is represented by downward arrows which connects to the block E subscript K which again connects to the block that contains O subscript 2 in a register that denotes 8 leftmost bits that connects with a connector P subscript 2 that shows 8 bits and C subscript 2 is connected to the connector. Again O subscript 2 is connected to register that contains O subscript 2 present in rightmost side of the block X subscript 2 that connects the block X subscript 3 and shifts the rightmost register labeled as drop to the leftmost side that is represented by downward arrows which connects to the block E subscript K which again connects to the block that contains O subscript 3 in a register that denotes 8 leftmost bits that connects with a connector P subscript 2 that shows 8 bits and C subscript 3 is connected to the connector which is further connected to dot-dot-dot. The first block X subscript 1 is connected to the next X subscript 1 block that contains O subscript 1 and X subscript 2 block is connected to the next X subscript 2 block that contains O subscript 2.

Long description

The illustration shows nine blocks that are separated. The first block labeled as X subscript 1 connects to the second block E subscript which again connects to the block that contains O subscript 1 in a register that denotes 8 leftmost bits that connect with a connector P subscript 1 that shows 8 bits and C subscript 1 is connected to the connector. Block X subscript 2 equals X subscript 1 plus 1 connects to the block E subscript K which again connects to the block that contains O subscript 2 in a register that denotes 8 leftmost bits that connect with a connector P subscript 2 that shows 8 bits and C subscript 2 is connected to the connector. Block X subscript 3 equals X subscript 2 plus 1 connects to the block E subscript K which again connects to the block that contains O subscript 3 in a register that denotes 8 leftmost bits that connect with a connector P subscript 2 that shows 8 bits and C subscript 3 is connected to the connector which is further connected to dot-dot-dot.

Long description

The illustration shows two inputs K subscript i and R subscript i minus l fed to a feedback which is labeled as f. Output of feedback is fed to an adder which is denoted as XOR. A output R subscript i is obtain from the adder. Another input L subscript I minus l is fed to the adder. An arrow is shown from R subscript i minus l which is labeled as L subscript i.

Long description

The illustration shows six numbers, 1, 2, 3, 4, 5, 6 on the upper row and eight numbers 1, 2, 4, 3, 4, 3, 5, 6 on lower row respectively. Several arrows are shown between 1 to 1, 2 to 2, 4 to 4, 3 to 3, 4 to 4, 3 to 3, 5 to 5, 6 to 6 from upper row to bottom row.

Long description

Flow chart shows an input R subscript i minus l which is fed to E left parenthesis R subscript i minus l right parenthesis. Output of E left parenthesis R subscript i minus l right parenthesis is fed to an adder. Adder output is subdivided into two parts 4 bits and 4 bits. Output of 4bits and 4 bits are connected to S subscript 1 and S subscript 2 respectively. Output of the S subscript 1 and S subscript 2 is connected to f left parenthesis R subscript i minus l, K subscript i right parenthesis. On the right side, another input is fed to an adder which is labeled as K subscript i.

Long description

Table shows two columns, first four bits along with their frequencies, the values are labeled as 0000: 12, 1000: 33, 0001: 7, 1001: 40, 0010: 8, 1010: 35, 0011: 15, 1011: 35, 0100: 4, 1100: 59, 0101: 3, 1101: 32, 0110: 4, 1110: 28, 0111: 6 and 1111: 39.

Another table shows two columns, last four bits along with their frequencies, the values are labeled as 0000: 14, 1000: 8, 0001: 6, 1001: 16, 0010: 42, 1010: 8, 0011: 10, 1011: 18, 0100: 27, 1100: 8, 0101: 10, 1101: 23, 0110: 8, 1110: 6, 0111: 11, and 1111: 17.

Long description

An input Plaintext is fed to IP. The output of IP is subdivided in two parts L subscript 0 and R subscript 0 respectively. The output of L subscript 0 is connected to an adder whereas the output of R subscript 0 is connected to the adder through a feedback f, where an input K subscript 1 is fed. Output of the adder is connected to R subscript 1 and same of R subscript 0 is connected to L subscript 1. The output of L subscript 1 is connected to an adder whereas the output of R subscript 1 is connected to the adder through a feedback f, where an input K subscript 2 is fed. Output of the adder is connected to R subscript 2 and same of R subscript 1 is connected to L subscript 2. L subscript 2 and R subscript 2 are further connected to L subscript 15 and R subscript 15 respectively. The output of L subscript 15 is connected to an adder whereas the output of R subscript 15 is connected to the adder through a feedback f, where an input K subscript 16 is fed. Output of the adder is connected to R subscript 16 and same of R subscript 15 is connected to L subscript 16. The outputs obtained from R subscript 16 and L subscript 16 are fed to IP superscript negative 1 which is further fed to ciphertext.

Long description

The values are labeled as follows:

Row 1: 58 50 42 34 26 18 10 2 60 52 44 36 28 20 12 4

Row 2: 62 54 46 38 30 22 14 6 64 56 48 40 32 24 16 8

Row 3: 57 49 41 33 25 17 9 1 59 51 43 35 27 19 11 3

Row 4: 61 53 45 37 29 21 13 5 63 55 47 39 31 23 15 7

Long description

The values are labeled as follows:

Row 1: 32 1 2 3 4 5 4 5 6 7 8 9

Row 2: 8 9 10 11 12 13 12 13 14 15 16 17

Row 3: 16 17 18 19 20 21 20 21 22 23 24 25

Row 4: 24 25 26 27 28 29 28 29 30 31 32 1

Long description

The values are labeled as follows:

Row 1: 16 7 20 21 29 12 28 17 1 15 23 26 5 18 31 10

Row 2: 2 8 24 14 32 27 3 9 19 13 30 6 22 11 4 25

Long description

A flow chart starts with an input R subscript i minus 1 which is fed to Expander. The output of expander is fed to E left parenthesis R i minus 1 right parenthesis. The output of E left parenthesis R i minus 1 right parenthesis and another input K subscript i are fed to an adder. The output of the adder is subdivided into eight 6 bits data which are labeled as B subscript 1, B subscript 2, B subscript 3, B subscript 4, B subscript 5, B subscript 6, B subscript 7 and B subscript 8 respectively. The outputs of B subscript 1, B subscript 2, B subscript 3, B subscript 4, B subscript 5, B subscript 6, B subscript 7 and B subscript 8 respectively are fed to S subscript 1, S subscript 2, S subscript 3, S subscript 4, S subscript 5, S subscript 6, S subscript 7 and S subscript 8 respectively. Again the outputs of S subscript 1, S subscript 2, S subscript 3, S subscript 4, S subscript 5, S subscript 6, S subscript 7 and S subscript 8 respectively are fed to eight 4 bits data which are labeled as C subscript 1, C subscript 2, C subscript 3, C subscript 4, C subscript 5, C subscript 6, C subscript 7 and C subscript 8 respectively. All the outputs are fed to permutation. The output of permutation is fed to f left parenthesis R subscript i minus 1 comma K i right parenthesis.

Long description

The values are labeled as follows:

Row 1: 57 49 41 33 25 17 9 1 58 50 42 34 26 18

Row 2: 10 2 59 51 43 35 27 19 11 3 60 52 44 36

Row 3: 63 55 47 39 31 23 15 7 62 54 46 38 30 22

Row 4: 14 6 61 53 45 37 29 21 13 5 28 20 12 4

Long description

The values are labeled as follows:

Round: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Shift: 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

Long description

A table shows information of 48 bits chosen from the 56-bit string C subscript i D subscript i and the output are K subscript i. The table shows four rows and twelve columns.

Row 1: 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21 and 10.

Row 2: 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13 and 2.

Row 3: 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33 and 48.

Row 4: 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29 and 32.

Long description

A table shows information of S-Boxes. The table divided into eight parts depicted as S-box 1, S-box 2, S-box 3, S-box 4, S-box 5, S-box 6, S-box 7 and S-box 8 where each table shows four rows and 16 columns with several numbers.

Long description

A box labeled as plaintext is linked with the box AddRoundKey, The box AddRoundKey is labeled as W left parenthesis 0 right parenthesis, W left parenthesis 1 right parenthesis, W left parenthesis 2 right parenthesis, W left parenthesis 3 right parenthesis with a left inward arrow. Below the box AddRoundKey, a block with four boxes labeled as Round 1. In the block the first box is SubBytes which is linked with the below box labeled as ShiftRows. Again the ShiftRows is linked with MixColumns and lastly the MixColumns box is linked with AddRoundKey which is labeled as W left parenthesis 4 right parenthesis, W left parenthesis 5 right parenthesis, W left parenthesis 6 right parenthesis, W left parenthesis 7 right parenthesis with a left inward arrow. Again a block with four boxes labeled as Round 9, starting with the first box labeled as SubBytes which is linked with ShiftRows. Again ShiftRows is linked with MixColumns and lastly the MixColumns box is linked with AddRound Key which is labeled as W left parenthesis 36 right parenthesis, W left parenthesis 37 right parenthesis, W left parenthesis 38 right parenthesis, W left parenthesis 39 right parenthesis with a left inward arrow. Again a block with three boxes labeled as Round 10, starting with the first box labeled as SubBytes which is linked with ShiftRows. Again ShiftRows is linked with AddRound Key which is labeled as W left parenthesis 40 right parenthesis, W left parenthesis 41 right parenthesis, W left parenthesis 42 right parenthesis, W left parenthesis 43 right parenthesis with a left inward arrow. Below the block, a box is linked labeled as Ciphertext.

Long description

A table shows the information of Rijndael Encryption. The data listed for:

Line 1: ARK, using the 0^{th} round key.

Line 2: Nine rounds of SB, SR, MC, ARK, using round keys 1 to 9.

Line 3: A final round: SB, SR, ARK, using the 10^{th} round key.

Long description

A table shows S-Box for Rijndael that shows various numerical values. The values are listed as

99 124 119 123 242 107 111 197 48 1 103 43 254 215 171 118

202 130 201 125 250 89 71 240 173 212 162 175 156 164 114 192

183 253 147 38 54 63 247 204 52 165 229 241 113 216 49 21

4 199 35 195 24 150 5 154 7 18 128 226 235 39 178 117

9 131 44 26 27 110 90 160 82 59 214 179 41 227 47 132

83 209 0 237 32 252 177 91 106 203 190 57 74 76 88 207

208 239 170 251 67 77 51 133 69 249 2 127 80 60 159 168

81 163 64 143 146 157 56 245 188 182 218 33 16 255 243 210

205 12 19 236 95 151 68 23 196 167 126 61 100 93 25 115

96 129 79 220 34 42 144 136 70 238 184 20 222 94 11 219

224 50 58 10 73 6 36 92 194 211 172 98 145 149 228 121

231 200 55 109 141 213 78 169 108 86 244 234 101 122 174 8

186 120 37 46 28 166 180 198 232 221 116 31 75 189 139 138

112 62 181 102 72 3 246 14 97 53 87 185 134 193 29 158

225 248 152 17 105 217 142 148 155 30 135 233 206 85 40
223

140 161 137 13 191 230 66 104 65 153 45 15 176 84 187 22

Long description

A table shows the information of Rijndael Decryption.
The data listed for

line 1: ARK, using the 10^{th} round key.

Line 2: Nine rounds of ISB, ISR, IMC, IARK, using round keys 9 to 1.

Line 3: A final round: ISB, ISR, ARK, using the 0^{th} round key.

Long description

A table shows information of The RSA Algorithm. The data listed for

Line 1: Bob chooses secret primes p and q and computes n equal pq.

Line 2: Bob chooses e with gcd left parenthesis e comma left parenthesis p minus 1 right parenthesis left parenthesis q minus 1 double right parenthesis equals 1.

Line 3: Bob computers d with de-triple equal 1 left parenthesis mod left parenthesis p minus 1 right parenthesis left parenthesis q minus 1 double right parenthesis.

Line 4: Bob makes n and e public comma and keeps p comma q comma d secret.

Line 5: Alice encrypts m as c triple equal m superscript e left parenthesis mod n right parenthesis and sends c to Bob.

Line 6: Bob decrypts by computing m triple equal c superscript d left parenthesis mod n.

Long description

A table shows information of RSA Encryption Exponents. The data listed for e and percentage as follows:

Column 1: 65537, 17, 41, 3, 19, 25, 5, 7, 11, 257 and others.

Column 2: 95.4933 percent, 3.1035 percent, 0.4574 percent, 0.3578 percent, 0.1506 percent, 0.1339 percent, 0.1111 percent, 0.0596 percent, 0.0313 percent, 0.0241 percent and 0.0774 percent.

Long description

A table shows information of Factorization Records. The data listed for year and number of digits as follows:

Column 1: 1964, 1974, 1984, 1994, 1999, 2003, 2005 and 2009.

Column 2: 20, 45, 71, 129, 155, 174, 200 and 232.

Long description

The long message is an arbitrary message like ..... 01101001... which pass through the hash function and produces the output message digest of fixed length 256 bit- 11...10.

Long description

Initially, a value IV is fed as input in the first block f. A two-bit string M sub 0 is first fed into the block. The blocks are then fed one-by-one into f. The message M goes from M sub 0 to up to M sub left parenthesis n-1right parenthesis. The final output is the hash value H left parenthesis M right parenthesis.

Long description

r is taken as the rate and c is taken as the capacity. First a block 0 0 is introduced. First 0 is the rate r and second 0 is the capacity c. Rate r occupies maximum part of the block. Then the message signal M subscript 0 is transferred to block f from where the message is transferred back to rate and capacity block and so on till message signal M subscript n minus 1. This is the absorbing part. The diagram is then separated by a dotted line after which the squeezing part starts. Here the outputs Z subscript 0 and so on are squeezed out of the rate and capacity and f blocks.

Long description

An illustration shows three blocks labeled as Data Record k minus 1, Data Record k, and Data Record k + 1 and each block contains a data block, h left parenthesis right parenthesis, and a pointer. A vertical box with h left parenthesis right parenthesis and a pointer is shown at the upper top right. The vertical box is linked with the third block with the help of an arrow. Again the third block is linked with the second box and also the second box is linked with the first block with the help of an arrow.

Long description

An illustration shows a tree diagram. The first block of the tree diagram is h left parenthesis right parenthesis. The first block is linked with the second block with a downward arrow and the second block has h left parenthesis right parenthesis h left parenthesis right parenthesis. The second block is divided into two child blocks labeled as h left parenthesis right parenthesis h left parenthesis right parenthesis and h left parenthesis right parenthesis h left parenthesis right parenthesis. Again the block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as h left parenthesis right parenthesis h left parenthesis right parenthesis and h left parenthesis right parenthesis h left parenthesis right parenthesis. Also, the block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as h left parenthesis right parenthesis h left parenthesis right parenthesis and h left parenthesis right parenthesis h left parenthesis right parenthesis. The block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as R0 and R1. The block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as R2 and R3. The block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as R4 and R5. The block h left parenthesis right parenthesis h left parenthesis right parenthesis is subdivided into two child blocks labeled as R6 and R7.

Long description

The illustration shows a basic Kerberos model with the participants Cliff, Serge, Trent and Grant, where Cliff is at the center which is linked to Trent, Grant and Serge. Cliff sends a message to Trent indicated by an arrow numbered 1. Trent replies the message to Cliff again indicated by an arrow numbered 2. Cliff sends a message to Grant indicated by an arrow numbered 3. Grant replies the message to Cliff again indicated by an arrow numbered 4. Cliff sends a final message to Serge indicated by an arrow numbered 5.

Long description

The illustration shows a block diagram of a Certification Hierarchy. The block “CA” abbreviated for Certification Authority, is characterized as Client, Client and “RA” abbreviated for Registration Authorities. The block “RA” is again sub-divided into three Clients.

Long description

The screenshot shows a CA's Certificate; General.

Line 1: This certificate has been verified for the following uses, colon

Line 2: Email Signer Certificate

Line 3: Email Recipient Certificate

Line 4: Status Responder Certificate

Line 5: Issued to, colon

Line 6: Organization left parenthesis O right parenthesis, colon, VeriSign, Inc

Line 7: Organizational Unit left parenthesis OU right parenthesis, colon, Class 1 Public Primary Certification Authority dash G2

Line 8: Serial Number

Line 9: Issued by, colon

Line 10: Organization left parenthesis O right parenthesis, colon, VeriSign, Inc

Line 11: Organizational Unit left parenthesis OU right parenthesis, colon, Class 1 Public Primary Certification Authority dash G2

Line 12: Validity, colon

Line 13: Issues on, colon, 05 forward slash 17 forward slash 98

Line 14: Expires on, colon, 05 forward slash 18 forward slash 98

Line 15: Fingerprints, colon

Line 16: SHA1 Fingerprint, colon

Line 17: MD5 Fingerprint, colon

Long description

The screenshot shows a CA's Certificate; Details.

Line 1: Certificate Hierarchy

Line 2: Verisign Class 1 Public Primary Certification Authority dash G2

Line 3: Certificate Fields

Line 4: Verisign Class 1 Public Primary Certification Authority dash G2

Line 5: Certificate

Line 6: Version, colon, Version 1

Line 7: Serial Number, colon

Line 8: Certificate Signature Algorithms, colon, PKCS hashtag 1 SHA dash 1 With RSA Encryption

Line 9: Issuer, colon, OU equals VeriSign Trust Network

Line 14: Validity

Line 15: Not before, colon, 05 forward slash 17 forward slash 98

Line 16: Not after, colon, 05 forward slash 18 forward slash 98

Line 17: Subject, colon, OU equals VeriSign Trust Network

Line 22: Subject Public Key Info, colon, PKCS hashtag 1 RSA Encryption

Line 23: Subject’s Public Key, colon

Line 24: A table of 9 rows and 16 columns is shown

Line 25: Certificate Signature Algorithm, colon, PKCS hashtag 1 SHA dash 1 With RSA Encryption

Line 26: Certificate Signature Name, colon

Line 27: A table of 8 rows and 16 columns is shown.

Long description

The screenshot shows a Clients Certificate.

Line 1: Certificate Hierarchy

Line 2: Verisign Class 3 Public Primary CA

Line 3: Certificate Fields

Line 4: Verisign Class 3 Public Primary Certification Authority

Line 5: Certificate

Line 6: Version, colon, Version 3

Line 7: Serial Number, colon

Line 8: Certificate Signature Algorithms, colon, md5RSA

Line 9: Issuer, colon, OU equals www.verisign.com forward slash CPS Incorp.

Line 14: Validity

Line 15: Not before, colon, Sunday, September 21, 2003

Line 16: Not after, colon, Wednesday, September 21, 2005

Line 17: Subject, colon, CN equals online.wellsfargo.com

Line 22: Subject Public Key Info, colon, PKCS hashtag 1 RSA Encryption

Line 23: Subject's Public Key, colon, 30 81 89 02 81 81 00 a9

Line 24: Basic Constraints, colon, Subject Type equals
End Entity, Path Length Constraint equals None

Line 25: Subject's Key Usage, colon, Digital Signature,
Key Encipherment left parenthesis AO right parenthesis

Line 26: CRL Distribution Points, colon

Line 27: Certificate Signature Algorithm, colon, MD5
With RSA Encryption

Line 28: Certificate Signature Value, colon

Long description

A table represents Block ID colon 76 Create Coins. The table contains 4 rows and 3 columns. The Block ID colon 76 Create Coins is divided into three columns labeled as Trans ID with values 0, 1 and 2, Value with values 5, 2 and 10 and Recipient with values PK subscript BB, PK subscript Alice and PK subscript Bob.

Long description

A table represents Block ID colon 77 Consume Coins colon 41 left parentheses 2 right parentheses, 16 left parentheses 0 right parentheses, 31 left parentheses 1 right parentheses, Create Coins. The table contains 6 rows and 3 columns. The Block ID colon 77 Consume Coins colon 41 left parentheses 2 right parentheses, 16 left parentheses 0 right parentheses, 31 left parentheses 1 right parentheses, Create Coins is divided into three columns labeled as Trans ID with values 0, 1, 2 and 3, Value with values 15, 5, 4 and 11 and Recipient with values PK subscript Sarah Store, PK subscript Alice, PK subscript Bob and PK subscript Charles.

Long description

The block diagram of Bit coin’s block chain and a Merkle tree of transactions three blocks in adjacent position. Each block has four lines of codes. The third block connects to the second and the second block connects to the first.

Line 1: prev underscore hash tag, colon, h left parenthesis right parenthesis

Line 2: timestamp

Line 3: merkleroot, colon, h left parenthesis right parenthesis

Line 4: nonce

Another block is placed just below the second block, which is comprised of one sub block, with a line: h left parenthesis right parenthesis, indented, h left parenthesis right parenthesis. Below the first, there are two other sub-blocks with lines in each block as: h left parenthesis right parenthesis, indented, h left parenthesis right parenthesis, connected with two arrows with the first block. The above two sub-blocks are connected to four sub-blocks represented as TX.

Long description

The box is having a uniform thickness highlighted with black. Another solid rectangle is present inside the tunnel and below it a door is made.

Long description

Illustration shows the Tunnel Used in the Zero-Knowledge Protocol. The tunnel is represented by a rectangular box with an opening at the top. Inside it, another tunnel is made by rectangular box with an opening at the top. Both the boxes are having a uniform thickness highlighted with black. Another solid rectangle is present inside the tunnel and a Central Chamber is presented below it.

Long description

A schematic diagram of Huffman Encoding, showing outputs a, b, c and d, arranged vertically. Output c corresponding to 0.1 and d corresponding to 0.1 gives 0.2, with two choices 0 and 1. Again, 0.2 and b corresponding 0.3 gives 0.5, with two choices 0 and 1. Further, 0.5 and a corresponding 0.5 gives 1, with two choices 0 and 1.

Long description

The illustration shows Shannon's Experiment on the Entropy of English. There are six sentences “there is no reverse”, “on a motorcycle a”, “friend of mine found”, “this out rather”, “dramatically the” and “other day”. Below each letter, information obtained is displayed.

Long description

The x-axis ranges from negative 1 to 3, in increments of 1 and the y-axis ranges from negative 4 to 4, increments of 2. The first curve begins at the first quadrant, passes decreasing through point 1 of the x-axis, then ends decreasing at the fourth quadrant making a peak point at point 1 of the x-axis.

Long description

The x-axis ranges from negative 4 to 8, in increments of 2 and the y-axis ranges from negative 20 to 20, in increments of 10. The curve starts from the first quadrant passing through the point 10 of the y-axis, then passes through the second quadrant decreasing through the point negative 4 of the x-axis intersecting the point negative 10 and ends decreasing at the fourth quadrant.

Long description

The graph shows a curve, a straight line and another doted straight line that is parallel to the y-axis and it is plotted in the first and second quadrant. The curve begins at the first quadrant passes decreasing through the y-axis, enters the second quadrant passes through the x-axis, enters the third quadrant passes through the y-axis and ends decreasing at the fourth quadrant.

Long description

The illustration shows a straight line segment and two vectors. The first vector labeled as v subscript 1 lies along the line segment pointing toward the right direction. The second vector labeled as v subscript 2 starts from the beginning point of the first vector, pointing upwards and slightly deflexed towards right from the angle 90 degrees. The second line lies between two dotted lines perpendicular to the line segment.

Long description

The illustration shows a straight line segment and two vectors. The first vector labeled as v subscript 1 lies along the line segment pointing toward the right direction. The second vector labeled as v subscript 2 starts from the beginning point of the first vector, pointing upwards and deflexed at angle 60 degrees towards the right. The second vector lies between two dotted lines perpendicular to the line segment and the vector crosses the perpendicular line at the very right.

Long description

The illustration shows an input message which is fed to the encoder, encoder output is fed into the noisy channel through codewords. Noisy channel output is fed to decoder and decoder output is fed to message.

Long description

The x-axis ranges from 0.1 to 0.5, in increments of 0.1 and the y-axis labeled as code rate ranges from 0.2 to 1, in increments of 0.2. The graph shows a non-linear decreasing curve starting from point 1 of the y-axis and ends at point 0.5 on the x-axis.

Long description

The illustration shows a white rectangular bar labeled as light source with a vertical edge at the right end. A black rectangular bar with a vertical edge labeled as Polaroid A at the right end which is inclined to a grey rectangular bar with a vertical black edge is shown. An arrow labeled as light pointing towards right is shown below the illustration.

Long description

The illustration shows a white rectangular bar labeled as light source with a vertical edge at the right end. A black rectangular bar with a vertical edge labeled as Polaroid A at the right end which is inclined to a grey rectangular bar with a vertical edge labeled as Polaroid C and a black vertical edge is are shown. An arrow labeled as light pointing towards right is shown below the illustration.

Long description

The illustration shows a white rectangular bar labeled as light source with a vertical edge at the right end. A black rectangular bar with a vertical edge labeled as Polaroid A at the right end which is inclined to a grey rectangular bar with a vertical edge labeled as Polaroid B is shown. The Polaroid B is inclined with a gray rectangular bar with a vertical bar labeled as Polaroid C which is inclined with a grey rectangular bar with a vertical black edge is shown. An arrow labeled as light pointing towards right is shown below the illustration.

Long description

The image shows a graph of y versus x. The x axis ranges from 0 to 7, in increments of 1 and the y-axis ranges from 0 to 1, in increments of 0.2. Following coordinates (0, 0.15), (1, 0.15), (2, 0.35), (3, 0.85), (4, 0.35), (5, 0.85), (6, 0.35) and (7, 0.15) are marked.

Long description

The image shows a graph of y versus x. The x axis ranges from 0 to 14, in increments of 2 and the y-axis ranges from 0 to 1, in increments of 0.2. Following coordinates (0, 1), (1, 0.2), (2, 0.3), (3, 0.9), (4, 0), (5, 0.2), (6, 0.65), (7, 0.3), (8, 0), (9, 0.3), (10, 0.65), (11, 0.2), (12, 0), (13, 0.9), (14, 0.25) and (15, 0.2) are marked.

Long description

The image shows a graph of y versus x. The x axis ranges from 0 to 500, in increments of 100 and the y-axis ranges from 0 to 0.2, in increments of 0.05. The graph starts from y equals 0.01, is comprised of four peaks with an altitude of y equals 0.02. The graph is comprised of numerous dots, whose density decreases towards the peaks.

Long description

A graph ranges negative 4 to 4 in increment of 2 in vertical axis and negative 1 to 3 in increment of 1 in horizontal axis. The graph plots an oval shaped curve at point 0 on vertical axis. Another elliptic curve is plotted that starts from 3 on horizontal axis, vertex lies at (1, 0) and extends to (3, 5).

Long description

A y versus x graph ranges from negative 4 to 4 in increment of 2 in y axis and negative 1 to 3 in increment of 1 in x axis. The graph plots an oval shaped curve whose vertex lies at the origin and (negative 1, 0). Another elliptic curve is plotted which is symmetric about the x axis and vertex lies at (1, 0).

Long description

A graph ranges negative 5 to 5 in increment of 1 in vertical axis and negative 1 to 3 in increment of 0.5 in horizontal axis. The graph plots an oval shaped curve at point 0 on vertical axis. Another elliptic curve is plotted that starts from 3 on horizontal axis, vertex lies at (1, 0) and extends to (3, 5). An equation is shown above the graph depicting y subscript 2 minus x times left parenthesis x minus 1 right parenthesis times left parenthesis x plus 1 right parenthesis equals 0.