April 23, 2024

Since some people need more time on the project, due date is now (and won't change) friday at midnight. Homework due friday is now due sunday midnight.

From Crypto.mw, as usual

```
> with(StringTools):
```

Alphabet Setup

StringToList, ListToString

StringToList converts a string into a list of numbers representing the position of each character in the Alphabet.
ListToString converts such a list back into a text string.

Note that this differ slightly from what we did in class, in that Alphabet[n] is represented by n-1. This will be more convenient when doing arithmetic.
April 20, 2024: adjusted StringToList to remove unrecognized characters and issue a warning (as in homework problem 30).

```
> StringToList:=proc(str::string)
    global Alphabet;
    local numlist:=map( s->SearchText(s,Alphabet)-1, Explode
  (str));
    numlist := remove(x->x=-1, numlist);  # just kill any -1
  entries.
    if (length(str) > numelems(numlist)) then
      WARNING("%1 unrecognized characters omitted.",length
  (str)-numelems(numlist));
    fi;
    return(numlist);
  end:
```

```
> ListToString:=proc(numlist::list(nonnegint))
    global Alphabet;
    return(Implode(map(k->Alphabet[k+1],numlist)) );
  end:
```

Affine cipher

The affine cipher is like Caesar, has both a shift and a mulitplication, $x \to m \cdot x + b \bmod p$
The encryption is not invertible if the muliplicand m and the length of the Alphabet and not relatively prime.
To encrypt $x \to m \cdot x + b$, use Affine(plaintext, m, b);
Decrypt with Affine(crypttext, m, b, decrypt)

$$x \mapsto m \cdot x + b$$

$$m$$

$$x \mapsto m \cdot x + b \qquad \textbf{(1.3.1)}$$

```
> Affine:= proc(msg::string, m::integer, b::integer:=0,
  {decrypt::truefalse :=false})
```

```
    global Alphabet;
    local Alen:=length(Alphabet);
    if (gcd(m,Alen) <> 1) then
      error(sprintf("m=%d is not relatively prime to the length
  of the Alphabet=%d", m, Alen));
     fi;
    if (decrypt) then
      return(Affine(msg,modp(1/m,Alen),modp(-b/m,Alen)));
     fi:
    return(ListToString(map(x->modp(m*x+b,Alen), StringToList
  (msg))));
  end:
```

```
> msg:="abcABC";
  StringToList(msg);
```

$$msg := \text{"abcABC"}$$

$$[26, 27, 28, 0, 1, 2] \quad (1)$$

```
> Affine(msg,5);
```

$$\text{"QVaAFK"} \quad (2)$$

```
> StringToList(%);
```

$$[16, 21, 26, 0, 5, 10] \quad (3)$$

To operate on blocks of letters, we change base.

```
> base57:=StringToList(msg)
```

$$base57 := [26, 27, 28, 0, 1, 2] \quad (4)$$

change to base 57^2

```
> convert(base57,base,57,57^2);
```

$$[1565, 28, 115] \quad (5)$$

115=1+2*57

```
> convert([1,2,3,4],base,10,100);
```

$$[21, 43] \quad (6)$$

```
> 34 = 3*10+4
```

$$34 = 34 \quad (7)$$

```
> convert([3,4,0,0], base, 10, 100000);
```

$$[43] \quad (8)$$

Maple does least significant digit first.

Plan: modify StringToList, ListToString to convert in larger blocks, then do arithmetic on those larger blocks.

```
> StringToList:=proc(str::string, {blocksize::posint:=1})
    global Alphabet;
     local Alen:=length(Alphabet);
    local numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    numlist := remove(x->x=-1, numlist);  # just kill any -1
  entries.
```

```
    if (length(str) > numelems(numlist)) then
      WARNING("%1 unrecognized characters omitted.",length(str)-
   numelems(numlist));
    fi;
    if (blocksize>1) then
      numlist:=convert(numlist,base,Alen,Alen^blocksize);
    fi;
   return(numlist);
  end:
```

```
> ABC2:=StringToList("abcABC",blocksize=2);
```

$$ABC2 := [1565, 28, 115] \quad (9)$$

```
> ABC3:=StringToList("abcABC",blocksize=6);
```

$$ABC3 := [1214032652] \quad (10)$$

```
> ABC4:=StringToList("abcABC",blocksize=4);  # let's worry about
  this later.
```

$$ABC4 := [92537, 115] \quad (11)$$

How do we undo?

```
> ListToString:=proc(nums::list(nonnegint),{blocksize::posint:=1})
    global Alphabet;
    local Alen:=length(Alphabet);
   local numlist:=nums;
   if (blocksize>1) then
     numlist:=convert(numlist,base,Alen^blocksize,Alen);
    fi;
   return(Implode(map(k->Alphabet[k+1],numlist)) );
  end:
> ListToString(ABC2,blocksize=2);
```

"abcABC" (12)

```
> ListToString(ABC2);  # this is wrongo
```

"c" (13)

```
> ListToString(ABC4,blocksize=4);  #hey, this worked!  Why?
```

"abcABC" (14)

```
> ListToString(ABC3,blocksize=3);
```

"abcABC" (15)

```
> StringToList("AAAA",blocksize=2);
```

[0] (16)

```
> ListToString(%);
```

"A" (17)

Now let's make it useful in Affine:

```
> Affine:= proc(msg::string, m::integer, b::integer:=0,
  {decrypt::truefalse :=false}, {blocksize::posint:=1})
    global Alphabet;
    local Alen:=length(Alphabet);
    if (gcd(m,Alen) <> 1) then
      error(sprintf("m=%d is not relatively prime to the length of
  the Alphabet=%d", m, Alen));
    fi;
    if (decrypt) then
      return(Affine(msg,modp(1/m,Alen^blocksize),modp(-b/m,
```

```
   Alen^blocksize) ,
            ':-blocksize'=blocksize) ) ;
     f i :
    local numlist:=StringToList(msg,':-blocksize'=blocksize);
    return(ListToString(map(x->modp(m*x+b,Alen^blocksize) ,
  numlist), ':-blocksize'=blocksize) ) ;
  end:
> abc5:=Affine("abcd",5,blocksize=2); StringToList(%);
                                  abc5 := "QXah"

                                  [16, 23, 26, 33]                    (18)
> Affine(abc5,5,decrypt,blocksize=2);
                                      "abcd"                          (19)
```

Next time, we add salt.