ACCESS 2001 - RSA
Thursday June14

Our Plan Today:

While the RSA public-key encryption method is still clear in your mind, (NOT?), we will do the example worked out in the Tom Davis "Cryptography" notes, page 13-14. This should help make things more concrete. This example uses small numbers and a one-letter message, and Maple will do all of our computations. I believe this will make the algorithm more clear to you. The explanation of the Algorithm on page 6-7 of the Rivest-Shamir-Adleman is also a very concise outline which will make more and more sense as you play with examples.

After we digest Davis' example we will try somewhat larger prime numbers, to help prepare you for the part of your group project in which you send yourselves (and me) encoded messages .

We will absolutely stop by 10:00 today so that we have plenty of time to make our way over to JTB 120 for the lecture by Professor Seger at 10:30. If I get carried away start to look like I'm going past 10:00 , please stop me. We have all morning tomorrow in the lab so we need not finish this handout today. If the last couple of days have made you dizzy, I want to remind you that the number theory Jim was talking about, and the relation to RSA cryptography is usually taught in our 4000-level (typically junior/senior) number theory course! ..... I am not expecting you to understand every detail, but your experience should convince you that even something as boring as the integers has magic buried in it.

I will make you do a lot of your own typing in Part I, so that you can have a fair experience fixing errors. Therefore, in the file you download, the Maple commands which you see in the hardcopy might be gone.

In this example **Bob is going to send a message to Alice,** so their roles from yesterday's pictures are reversed.

Part I
The Davis Example:

1) Remember that Bob must encipher his message to Alice using **Alice's** public key, since only Alice has the tools to decipher such a message. So at some earlier time Alice created her tools as follows: Alice picked two prime numbers, from page 13: (p=23 and q=41). Define p and q using Maple.

```
> p:=23;
  q:=41;
     #shift-return for multi-line commands.
     #definitions with :=, NOT =
```

$$p := 23$$

$$q := 41$$

2) Define their product to be the modulus "N" for Alice's circle algebra, (mod N arithmetic). (N should turn out to equal 943.)

```
> N:=p*q;
```

$$N := 943$$

3a) Compute the number N2:=(p-1)*(q-1), which turns out to be 880. Then pick a number e which is relatively prime to N2. Check that Davis' and therefore our choice e:=7 works

```
> N2:=(p-1)*(q-1);
  e:=7;
```

$$N2 := 880$$

$$e := 7$$

```
> gcd(e,880);   #greatest common denominator
  ifactor(N2);  #integer factor (into primes)
    #how would you find out about these commands if you
    #weren't sure how to use them, or even if they existed?
```

$$1$$

$$(2)^4\ (5)\ (11)$$

3b) Why do we care about N2 and e? The RSA encryption scheme takes a message (number) "x" and raise it to the power "e" (mod N). In Mapleese,

```
> encrypt:=x->x^e mod N;
    #if you defined e and N above,
    #then their values will be used for the encrypt function
```

$$encrypt := x \to x^e \bmod N$$

Note: In the picture from yesterday's notes the letters "e" and "d" were standing for the encryption **functions**. Today and in Jim's lecture yesterday, they denote **powers** which appear as part of the encryption and dcryption functions. And the requirement that e and N2 be relatively prime guarantees that we there is a decrypting power "d" for the decryption function. We return to this point later in the example.

4) The message which Bob wishes to send is the number M=35. (Davis uses the letter M to stand for message. We used "x" yesterday in our picture and in our discussion).

According to the table on page 9 this message is the letter "Y". Anyway, define M to be 35.

```
> M:=35;
```

$$M := 35$$

5) Encrypt the message, and call the value "C" for ciphertext. Yesterday we called the encrypted message "y".

```
> C:=encrypt(M);  #either of these should work
  C:=M^e mod N;
```

$$C := 545$$

$$C := 545$$

6) Great, Maple just did this step! Remember from our tables yesterday that multiplying and computing powers in modular arithmetic is easy because you can always reduce your intermediate terms to be less than the modulus. That's why Maple would have no problem, even for huge numbers. Anyway, Alice gets the message "C" which Bob has encrypted using Alice's public encryption information.

7) Now Alice will decrypt. So she needs to use the decryption power "d". How did she find "d" when she was setting up her system? With number theory of course, in particular with Euler's Theorem. She

wants the decryption and encryption functions to be inverse functions. Since each is going to be a power function, she wants the formulas

$$(M^e)^d = M$$

i.e.

$$M^{(e\,d)} = M$$

to hold in our mod N modular arithmetic. **What does d have to satisfy to make this last identity true?** Because M is relatively prime to N=pq we know that M^(N2)=1 mod N, by Euler's Theorem. Therefore **if e*d = 1 plus a multiple of N2**, then M raised to the multiple of N2 power will be 1, and M^(e*d) will indeed equal M, provided we take both values in the natural set {1,2, ...M-1}. **(The last sentence may need some contemplation in a quiet place.)** (And, actually we didn't need to require M relatively prime to N, see the explanation on page 7 of the Rivest-Shamir-Adleman paper.) Also, Jim showed us how to find d using the generalized Euclidean algorithm, and the condition that we can do so is that e and N2 be relatively prime, exactly the condition we imposed back in step 3. Luckily for us, we don't have to program the little loop which does this computation, Maple already has done so.

```
> isolve(e*d+y*N2=1);  #stands for "integer solve", recall
      #we don't really care what value y is, we just want d
```

$$\{y = 3 + 7\ \_N1,\ d = -377 - 880\ \_N1\}$$

N1 is a free parameter in the formual for d. If we take N1=-1, we obtain Davis' value for d

```
> d:=-377-880*(-1);
```

$$d := 503$$

8) The decoding function is given by raising C to the power d (mod N). Davis shows how quickly this can be done using modular arithmetic.

```
> decrypt:=y->y^d mod N;
```

$$decrypt := y \to y^d \bmod N$$

```
> decrypt(C);  #both of these should work
  C^d mod N;
```

$$35$$

$$35$$

WE DID IT!

Our message pieces can be no larger than the modulus N, so in the example above we can't transmit more than a single letter. (Or, more precisely, a single letter per packet.)

Part II:

A more practical size.

We need to use somewhat larger numbers if you guys are going to transmit actual messages. We won't make them anywhere near the actual sizes that are required for real security, however. Since each letter uses 2 spaces. If we break our messages into packets of 6 letters then that will 12 digits per packet. So we want N to have at least 13 digits, so that the packet is smaller than N. It seems hopeless doesn't it? But it turns out most of the commands already in Maple

```
> restart:   #this will clear all old definitions.
             #It's a good idea to restart when you begin
```

```
              #new work - of course you might need to
              #go back and re-enter some old commands that
              #you need again
> rand();     #random number generator,
              #default range is between 0 and 12 digits
```

427419669081

```
> bigger:=rand(1..10^30):  #bigger
  bigger();
```

693270343633073697474256143564

The default size will be bigger than we need. Now let's get some primes:

```
> good:=rand(1..10^7):
  good();
```

2291351

```
> for i from 1 to 100 do
    x:=good():
     if isprime(x)=true  #check if number is prime
      then print(x);     #if it is, let's see it
    fi;
  od:
```

7288397

7311323

3411311

520837

6633343

7109903

2505301

It is unlikely your numbers agree with mine. (Well, in truth everytime I restart I get the same so-called random numbers.) But let's all use two of mine. We do the RSA procedure in a more compressed form: Steps 1,2)

```
> p:=7288397;  #I got these with my mouse, by cut and paste
  q:=3411311;
  N:=p*q;           #our modulus
```

$p := 7288397$

$q := 3411311$

$N := 24862988858467$

To see that a system of this size is not secure, try the following command. This is the command that would fail if we had chosen primes of length 200 instead of 12, and that's the reason RSA is secure when you use huge primes

```
> ifactor(N);   #prime factorization
```

(3411311) (7288397)

3) Find an encoding power e

```
> N2:=(p-1)*(q-1);
```

$N2 := 24862978158760$

```
> for i from 1 to 10 do
     x:=good();
     if gcd(x,N2)=1
       then print(x);
     fi
  od:
```

2159161

8262489

2293629

```
> e:=2626811;
  gcd(e,N2);   #check relative prime
```

$$e := 2626811$$

$$1$$

7) Get decoding power (done out of order with notes).

```
> isolve(e*d + y*N2 =1);
     #find decryption power
```

$$\{d = -24141644356869 - 24862978158760\ \_N1, y = 2550601 + 2626811\ \_N1\}$$

```
> d:=-24141644356869 mod N2;
     #use the mouse to copy and paste big numbers
```

$$d := 721333801891$$

Technical Point: When we get to step 6, or certainly step 8 Maple will complain when we try to compute large powers of large numbers, so we have to lead it through this modular computation in smaller steps. The procedure below does an analgous computation to Davis', except using powers of 10 rather than powers of 2. It's fine with me if you just use this procedure. The encryption algorithm makes use of the digit procedure which picks of the coefficients of powers of 10 in the decimal expansion of a number. So make sure to load digit before you load encrypt.

```
> digit:=(x,n)->trunc(x/10^n)-10*trunc(x/10^(n+1));
```

$$digit := (x, n) \to \mathrm{trunc}\left(\frac{x}{10^{n}}\right) - 10\ \mathrm{trunc}\left(\frac{x}{10^{(n+1)}}\right)$$

```
> digit(123.56,-1);
  digit(123.56,2);
  digit(123.56,0);
     #check how digit picks off the digits corresponding
     #to powers of 10
```

5

1

3

```
>
> encrypt:= proc(M1,E,N3)  #message, encipher power,modulus
                  #we assume all M1's, E's have at most 14 digits
      local i,j,  #indices
            L1,   #list of succesive 10th powers of M1
            ans;  #answer
    L1[1]:=M1  mod N3;
    for i from 2 to 14 do
```

```
        L1[i]:=L1[i-1]^10 mod N3;
     od:
     ans:=1:    #initialize answer
     for j from 1 to 14 do
        ans:=ans*(L1[j]^digit(E,j-1)) mod N3;
      od:

   RETURN(ans);
   end:


 >
```

Let's check!!!

```
> M:=12345;
```

$$M := 12345$$

```
> encrypt(M,e,N);
```

$$24742637839651$$

```
> encrypt(24742637839651,d,N);
    #decryption is just encryption with a different power
```

$$12345$$

YES!!!

Part III
An Actual Message

We Use Davis' Table on page 9 to encrypt " **I'm Dizzy**". We will need two chunks

```
> M1:=19674900;
  M2:=1445626261;
```

$$M1 := 19674900$$

$$M2 := 1445626261$$

```
> C1:=encrypt(M1,e,N);
  C2:=encrypt(M2,e,N);
```

$$C1 := 21347848801116$$

$$C2 := 16436422768597$$

```
> encrypt(C1,d,N);
  encrypt(C2,d,N);
```

$$19674900$$

$$1445626261$$

```
>
```