RSA Attacks Problem Set

*Instructions: Understanding the content on this sheet is required. You will need to be able to describe all three attacks on RSA given below, as well as the ways to prevent them. Actually coding the challenges is not required, however. If you have finished your RSA assignment, you must work on these challenges, but they can be done in any order. Doing at least three will give a bonus 20 point lab grade, which can help raise an average even though it isn’t extra credit.*

*To submit any challenge answers, make a Word document and paste your answers there. Print and submit when ready.*

RSA isn’t immune from attacks. For one thing, as we’ve seen, certain numbers turn out to be easier to factor than others.

*Challenge #1: Make a new version of your prime generating algorithm that allows you to specify a maximum distance between two primes. Continue to generate primes between 2^511 and 2^512. How far apart can two primes be for our difference-of-squares algorithm to factor their product in less than 1 minute? Less than 3 minutes? Either through your own mathematical intuition or Google searching, what do you think would be a reasonable gap between primes to enforce, to avoid this attack? After you write your answers in the Word doc, modify your actual prime generating code to enforce this gap. Comment the lines that add the modification, and paste in the Word doc as well.*

In addition, since the encryption key is available, one possible attack against RSA is to simply guess what the encoded message was and encode your guess with the available key, checking to see if you got it right.

*Challenge #2: Assume you know that a particular website gathers username and password data from a user and saves them in the following format:*

userid:exampleid, password:exampleid

*Then, it* ***pads the remaining characters with spaces*** *(not “Z”s) after the password, and encrypts to RSA using the following alphabet and key:*

alpha=AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz.;:-, !'0123456789

e=65537

m=90487058565911344860746920532714345107344888706603388689414973118770115868824428038927974148369533060365472446789821522968583861289788613738073758712025234963231865997137994675667715748727317358600691229070201161969867856640249424928418473804035245009947964183514868695116591835621822263433693274598400210107

*You intercept a message sent from my computer as I’m using this website. You assume that my username is either meckel, mceckel, or malcolm.eckel. The ‘m’, ‘c’, and ‘e’ characters that represent my initials might be capitalized also; you’re not sure. You intercept the following encoded number:*

89897807505048742568624266076183099622868191865791675665199726198073166863517809194896619121285164863721775826174711988683028633449762554684260071137667277939952878337117007006682648484027616481552917086710938427047904293454110730563208669424803871616352517233729085665806037146392445377160226330123954583864

*Is there any chance I was dumb enough to use one of the 1,000 most common passwords?*

*(Hint: yes.)*

*(Note: this challenge will not actually give you my real password.)*

Another attack on RSA encryption is called the *common modulus attack*. This arises when someone has encoded a message using the same *mod*, but two *different exponents*. This allows the message to be recovered easily without factoring the modulus at all! Say we call the two encrypted messages:

and

If we use Bezout’s Identity to solve for integers *x* and *y* such that then we simply compute:

(the original message)

Why? Well, take a look:

Oops!

One problem with coding an implementation of this attack is that one of *x* or *y* will be negative and negative exponents are problematic; as we’ve seen, they can’t simply be cycled up mod *n*. We can get around this, though, as long as *E* and *F* are coprime to *n* (which is extremely likely). For example, say *y* is negative. To compute , we will need to find and compute instead (which will now be computed using all positive numbers). Similarly for *E* and *x*, if *x* is negative.

*Challenge #3: Say you know that a certain organization sends encrypted information internally using a different e/d pair for each employee, but all with the following alphabet and mod:*

alpha= AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz.;:-, !'0123456789

m=106382053895550781631285989946671994901422402885384005734669390130404880949239142241842903713130115581490732292356870236631337904281854676807813263510001411983900084040159377605552511174634634036897745619667852912154300017597144357866633696471699967965565310764085206116047453528193353554423244052263729215021

*You intercept the following two encrypted messages. The first is sent with e = 65537, the second with e = 65539.*

9705214362176240488757154315269142226527342998912847822297075911563010684409685695827814130120258121752447148402533008661025750189640278508809845292177713038172472283895091451792871260549975042925801012917082250306617751605755352005930978382582506591529676350876122579466924493048347835402255719630632534163

88590435487871971803114351358643590003318398961051262151659593981700089564948599552525355620185034894658408726566489912554374308385411798543850162234517186786865315340506585183177906894450757137531814625246170907267488855285361923673004877088042108314244704349201379912731310282750824044803256776887507026390

*You think it is actually the same message sent to the two different employees. What does it say?*

One way of preventing both of the previous attacks on RSA is to insert junk character sequences into the message at random. When you do this, the encoded message is no longer predictable from the input. Obviously, any encoding method that does this would need to be matched with a decoding method that would be able to recognize and remove the junk sequences.

*Challenge #4: Modify your method that converts input strings into numbers to be encoded so that it randomly inserts junk sequences of random length, at least 3 in any given message. The junk sequences should be clearly marked (for instance, begin with < and end with >) and should contain random other characters from the encoding alphabet. Any padding at the end of the message should also be a similar junk sequence – no more padding with all “Z”s.*

*Also, modify your decoding method so it looks for any junk sequences and removes them before final output.*

*In the Word doc, paste two different sample runs of your new methods using the same e and m. Have them print out the modified strings with junk inserted and the encoded numbers, to show that when different sequences are inserted, the strings encode differently. In addition, paste your modified code for converting strings to numbers and back again into the Word doc as well. Comment the sections that add and remove junk characters.*