ID: 03
Name: Al-Naji, Nader

Collaborator #1: Last Name, First Name
Collaborator #2: Last Name, First Name

Solution to HW 10, Problem 1

COS 340 - Spring 2012

Consider the following sequence: $a_1 = a_2 = 1$ and successive terms of the sequence are defined by the identity $a_{n+1} = 2a_n + a_{n-1}$. Prove that $gcd(a_i, a_{i+1}) = 1$ for all $i \geq 1$.

Lemma 1: $a_n \; rem \; a_{n-1} = a_{n-2}$ for all $n \geq 3$.

To see this, note that $a_n = 2a_{n-1} + a_{n-2} = d \cdot q + r$ with $q = a_{n-1}$ and $0 \leq r < q = a_{n-1}$. Because we know that $0 \leq a_{n-2} < a_{n-1}$ by construction, we conclude that $r = a_{n-2}$ (and consequently $d = 2$) and, therefore, by the definition of the rem function, we have $a_n \; rem \; a_{n-1} = r = a_{n-2}$.

The rest of the proof follows by induction on n using Lemma 1.

Let $Q(n) =$ "$gcd(a_n, a_{n+1}) = 1$".

Base: $n = 1, n = 2$

$gcd(a_1, a_2) = gcd(1, 1) = 1$. $gcd(a_2, a_3) = gcd(1, 3) = 1$.

Step: Assume $Q(n)$ holds; prove $Q(n+1)$. That is, assume $gcd(a_n, a_{n+1}) = 1$; prove $gcd(a_{n+1}, a_{n+2}) = 1$.

Note that $gcd(a, b) = gcd(b, a)$ in general and consider $gcd(a_{n+1}, a_{n+2})$. We now use the fact shown in class and on page 66 of LL that $gcd(a, b) = gcd(a \; rem \; b, b)$. Applying this to $gcd(a_{n+1}, a_{n+2})$ and using Lemma 1 (which works since $n \geq 3$) along with the inductive hypothesis, we have: $gcd(a_{n+1}, a_{n+2}) = gcd(a_{n+2}, a_{n+1}) = gcd(a_{n+2} \; rem \; a_{n+1}, a_{n+1}) = gcd(a_n, a_{n+1}) = 1$.

Thus having shown that this holds for the base case where $n = 1$ and $n = 2$, and the step, we conclude that $gcd(a_n, a_{n+1}) = 1$ for all $n \geq 1$.

ID: 03
Name: Al-Naji, Nader

Collaborator #1: Last Name, First Name
Collaborator #2: Last Name, First Name

Solution to HW 10, Problem 2

COS 340 - Spring 2012

Consider $N = p \cdot q$ where p and q are primes. Prove that if $|p - q| < B$ then we can factor N in time $O(B \cdot ploylog(N))$.

Lemma 1: Division and checking divisibility are $O(n^2) = O(\log^2(N))$ operations where n is the maximum number of bits in the numbers to be divided.

Consider schoolbook long division of two binary numbers a and b (take b/a). For simplicity, let a and b both be n bits where n is the maximum number of bits in either a or b. We can do this without loss of generality by simply padding the more significant bits of the smaller number with zeros. Schoolbook long division in binary proceeds as follows where R is the remainder we keep track of and Q is the quotient (adapted from an instructor link provided on Piazza):

```
Q = 0      initialize quotient and remainder to zero
R = 0
for i = (n-1):0 do
R = R « 1
    R[0] = b[i]      set the least-significant bit of R equal to bit i of the numerator b
    if R ≥ a then
        R = R - a
        Q[i] := 1
    end
end
```

It should be clear that this is nothing more than schoolbook long division of two binary numbers a and b with b being the numerator and a being the denominator. It should also be clear that since left shifting requires moving n bits, subtraction requires n bit operations since we have to add bit by bit n times, and the $\geq$ test requires n bit checks in the worst case to compare the most significant bits (the rest of the operations in the inner loop are clearly constant time operations), and since we do each of these operations n times since they are all in the inner loop, the overall running time for division using this approach is $O(n^2)$. Further, it should be clear that to check if b is evenly divisible by a, we simply run this algorithm and check the value of R at the end to see if it is equal to zero, which makes divisibility checking an $O(n^2)$ operation as well. Since $n = \log(N)$, these running times can be written as $O(n^2) = O(\log^2(N))$.

Now, with Lemma 1 in hand, we can make progress on our problem. Recall that $N = p \cdot q$ is the product of two primes and consider $\sqrt{N}$. Since $N = \sqrt{N} \cdot \sqrt{N}$, if we have $0 < l < \sqrt{N}$, and $N = l \cdot m$ for some choice of l and m, then it must be the case that $m = N/l = \sqrt{N} \cdot \sqrt{N}/l > \sqrt{N}$, otherwise the product $l \cdot m$ would be strictly less than N. Further, if we have $l = \sqrt{N}$ and $N = l \cdot m$, then it must be the case that $m = \sqrt{N}$ as well, otherwise the original equality would not hold. Thus, if N is the product of two prime numbers p and q, using the above observations, it must be the case the $\sqrt{N} \in [min(p, q), max(p, q)]$. It should be clear that the maximum size of this range is less than or equal to $|p - q|$. So, if we simply start with $a_1 = \lfloor \sqrt{N} \rfloor$ and begin checking divisibility with $b = N$ iterating upward, that is, check if b/a_i is an even division for $i = 1, 2, ...$, we will eventually (that is, after no more than $|p - q|$ iterations) attempt to divide $b = N$ by $a_k = max(p, q)$ (and succeed in finding an even divisibility) after at most $k = |p - q|$ steps. At this point, having found that a_k divides N, and since N is the product of two prime factors p and q, we can conclude that $p = a_k$ and $q = N/a_k = N/p$. Of course, we do no more than $|p - q|$ divisions to reach this conclusion as stated above, and since each division takes $O(\log^2(N))$ time by Lemma 1, the total running time of this process is therefore $O(B \cdot \log^2(N))$.

ID: 03
Name: Al-Naji, Nader

Collaborator #1: Last Name, First Name
Collaborator #2: Last Name, First Name

Solution to HW 10, Problem 3

COS 340 - Spring 2012

1. Let's enhance the security of RSA by "doubling" the encryption. Given primes p and q, instead of choosing e and d such that $d \cdot e = 1(mod(p-1)(q-1)$, we pick (e_i, d_i) with $d_i \cdot e_i = 1(mod(p-1)(q-1))$ for $i = 1, 2$. To encrypt a plaintext, we first encrypt it using e_1, then encrypt the ciphertext again using e_2. For decryption, we use d_2 first and then d_1. What are the advantages or disadvantages of doing this over standard RSA?

This encryption scheme doubles the difficulty of encryption and doesn't make the messages sent and received any less vulnerable to attack. To see this, we consider two possible strategies of attack:

1) Attack by trying to compute p and q and, from this, compute d_1 and d_2.

Note that e_1 and e_2 are public. By this strategy, we attempt to factor n and, once we have p and q (if we can somehow manage to get them), we can compute d_1 and d_2 in $log^2(n)$ time by simply running $ExtendeEuclid(e_1, (p-1)(q-1))$ and $ExtendedEuclid(e_1, (p-1)(q-1))$ respectively. Double encryption doesn't make finding p and q any harder and only increases the amount of work we have to do in computing the decryption key by a constant factor, since we now have to run *ExtendedEuclid* twice instead of once. This difference is extremely marginal and, therefore, doesn't appear to provide much extra security. Perhaps if it made finding p and q twice as difficult this would be a reasonable thing to do, but running *ExtendedEuclid* is cheap compared to finding p and q and so the amount of work we have to do to with this strategy is essentially unchanged.

2) Attack by trying to compute a decryption key by brute force of the space d lives in.

The argument for why this isn't any more difficult is subtle. Since it appears we would have to do two brute force searches of the space of possible values of d_1 and d_2, operations which each could take a very long time to do depending on the size of n, it would seem as though doubling this amount of time would make doing this worthwhile for the party being attacked. However, encrypting twice doesn't make it any more difficult for the attacker to get the decryption keys by running a brute force search.

To see this, consider the following encryption and simplification of a message x using e_1 and e_2:

$E(x) = (x^{e_1})^{e_2} = x^{e_1} \cdot ... \cdot x^{e_1} (e_2 \text{ times}) = x^{e_1 \cdot e_2} = x^{e_3} (modn)$.

Note what happened in the last step. Because double encryption amounts to raising to a power, we end up with $e_1 \cdot e_2$ in the exponent, which simply becomes another constant e_3. Further, since e_1 and e_2 are relatively prime to $(p-1)(q-1)$, it follows that e_3 will be as well. Now if $0 < e_3 < (p-1)(q-1)$, it is clear that finding $d_3 = ExtendedEuclid(e_3, (p-1)(q-1))$ by brute force will allow us to decrypt all messages encrypted using e_1 and e_2 and the scheme is no more secure than with single encryption. If $e_3 > (p-1)(q-1)$, it takes a little more effort to see that we can still follow the same strategy. Consider the following simplification when $e_3 > (p-1)(q-1)$ taking into account that for an equality to hold $(modn)$ is equivalent to it holding $modp$ *and* $modq$:

$x^{e_3} = x^{(p-1)(q-1)k + e_3'} = x^{(p-1)(q-1)k} \cdot x^{e_3'} = x^{e_3'} (modn)$ by Fermat's Little Theorem using $x^{(p-1)k} = 1(modp)$ and likewise for q, with $e_3' < (p-1)(q-1)$.

What this equation is essentially saying is that even if $e_1 \cdot e_2$ is larger than $(p-1)(q-1)$, we can reduce this product down to the point where x is being exponentiated by a single number strictly less than $(p-1)(q-1)$ and that, since e_3 was relatively prime to $(p-1)(q-1)$, this new number will be as well, making it a completely valid *single* encryption key. Thus, this shows that encrypting a message twice with two different keys has the effect of encrypting once with a valid key that is equivalent to their product mod n. Thus, a brute force search

for d_3 is possible and requires no more work than a brute force search under single encryption. Thus, double encryption provides no extra protection under this method of attack either.

Having considered these two very common methods of attack using RSA and having shown that they are essentially unaffected by double encryption, we conclude that the extra effort double encryption adds for the encryptor is not offset by the increase in difficulty for the attacker.

Consider an RSA system with the parameters: $p = 101$, $q = 107$, $e = 4497$. Suppose that the ciphertext has the numerical value $C = 7411$. What is M?

We can use the *ExtendedEuclid* algorithm to compute d. More specifically, we run $ExtendedEuclid(e, (p-1)(q-1)) = ExtendedEuclid(4497, 10600)$, which returns coefficients x, y such that $4497x + 10600y = 1$, the coefficient x being the inverse of e that we require. Running the algorithm in detail we have:

$10600 = 2 \cdot 4497 + 1606$
$4497 = 2 \cdot 1606 + 1285$
$1606 = 1 \cdot 1285 + 321$
$1285 = 4 \cdot 321 + 1$

And manipulating these equations:

$1606 = 10600 - 2 \cdot 4497$
$1285 = 4497 - 2 \cdot 1606$
$321 = 1606 - 1 \cdot 1285$
$1 = 1285 - 4 \cdot 321$

And substituting to get the equation of the form we desire starting from the last:

$1 \cdot 1285 - 4 \cdot 321 = 1$
$1 \cdot 1285 - 4 \cdot (1606 - 1 \cdot 1285) = 1$
$-4 \cdot 1606 + 5 \cdot 1285 = 1$
$-4 \cdot 1606 + 5 \cdot (4497 - 2 \cdot 1606) = 1$
$5 \cdot 4497 - 14 \cdot 1606 = 1$
$5 \cdot 4497 - 14 \cdot (10600 - 2 \cdot 4497)$
$33 \cdot 4497 - 14 \cdot 10600 = 1.$

Thus, we have as our final answer from the algorithm that $d = 33$. Checking this, we see that $e \cdot d = 4497 \cdot 33 = 148401 = 1(mod 10600)$. Finally, decoding C by raising it to d, we get: $7411^3 3 = 2(mod 10807) \rightarrow M = 2$.

ID: 03 **Collaborator #1:** Last Name, First Name
Name: Al-Naji, Nader **Collaborator #2:** Last Name, First Name

Solution to HW 10, Problem 4

COS 340 - Spring 2012

Alice and Bob are each to given n bit strings: Alice has $x_1x_2...x_n$ and Bob has $y_1y_2...y_n$. Alice wants to communicate a few bits to Bob to enable him to determine whether their n bit strings are identical or not. Obviously, Alice can do this by sending her entire n bit string to Bob. It turns out that there is a way to do it with many fewer bits if we allow Bob to make errors with small probability.

Let $r_1, r_2, ..., r_n$ be n mutually independent random bits. Suppose the two n bit strings $x_1x_2...x_n$ and $y_1y_2...y_n$ differ in k bits. Compute the probability that $\sum_{i=1}^{n} x_i r_i = \sum_{i=1}^{n} y_i r_i (mod 2)$.

The probability that these quantities are equal is 1 if $k = 0$ and $1/2$ if $k > 0$. The proof is by induction.

Let $Q(k)$ = "The probability that the quantities are equal is $1/2$ if $k > 0$ and 1 if $k = 0$"

Base: $k = 0$ and $k = 1$.

If there are no errors in the strings, then the sums will agree trivially always since they will be adding the same thing, so the probability is 1. If there is one difference between the strings, say on bit j, then the sums will be different only if r_j is one, which happens with probability $1/2$. Thus the statement holds.

Step: Assume true for $Q(k)$; prove for $Q(k+1)$.

If we consider only the sum over the first k different bits, then we see that this sum can either agree or disagree, each of which happens with probability $1/2$ by the inductive hypothesis. If they agree, then the corresponding $(k+1)_{th}$ bit in r needs to be turned off to prevent the sums from shifting back into disagreement. This happens with probability $1/2$. If the sums disagree, then the corresponding $(k+1)_{th}$ bit in r needs to be turned on to allow the parity of the sums to shift into agreement. Thus, we have for our probability, assuming the bit in r corresponding to the $(k+1)_{th}$ different bit is r_j:

P(agree when k+1 bits different) =
$P(r_j = 1)P$(sum over k incorrect bits disagree) $+ P(r_j = 0)P$(sum over k incorrect bits agree)
$= 1/2 \cdot 1/2 + 1/2 \cdot 1/2 = 1/2$.

Thus, having shown that the statement holds for the base and the step, we conclude that the probability is 1 when $k = 0$ and $1/2$ when $k > 0$.

2. Use your result from the previous part to devise a scheme for Alice to send $O(\log(1/\delta))$ bits to Bob so that Bob can determine the correct answer with probability at least $1 - \delta$. Alice and Bob are allowed to share some information in advance before they are each given their n bit strings.

Allow Alice and Bob to share m random strings of length n, generated just as the r random string in the first part, before receiving their x and y strings. After receiving their x and y strings, have Alice and Bob compute m sums over their strings, one for each r string previously shared, just as in the first part, and have Alice send her m resulting bits over to Bob for comparison. Have Bob compare each of Alice's m bits to those generated from his own string (and the m random strings that he shares with Alice) and declare that the x and y strings are the same if all of these bits match his own, and that they are different if any of these bits don't match his own. Because all of the r bits are mutually independent, recalling the results derived about independent random bits on homework 3 and using the result derived in the first part, the probability of the strings agreeing when there are errors (this is the probability that a mistake is made) is P(false positive) $= (1/2)^m$. Thus, the probability that Bob can determine the correct answer through this process is $P(correct) = 1 - (1/2)^m$ since when the strings are the same, he will always report as such, and when they are different his error probability is

$(1/2)^m$. Allowing $\delta = (1/2)^m$, it should be clear that for any given choice of m, we have $m = \lg(2^m) = \lg(1/\delta)$ bits transferred and an error probability of $1 - (1/2)^m = 1 - \delta$.

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.