

19. Coding for Secrecy

19.1 Introduction

Protecting sensitive information from the prying eyes and ears of others is an important issue today as much as it has been for thousands of years. Government secrets, military strategies, business transactions, and communications of all kinds can be vulnerable to interception, thus creating the need for some sort of protective system.

Two such systems that have been employed before include **steganography** and **cryptography**. Steganography is concerned with physically concealing the information from others or hiding your communications in some clever fashion. Cryptography is concerned with mutating data, making it seemingly unreadable to those who are not intended to read it. Out of the two procedures, cryptography is the more mathematical, and as such we will discuss it at length in the next few sections of notes.

Additionally, as information transfers are becoming extremely easy for motivated parties to intercept with the increase in internet communications, solid cryptography is essential for security. Indeed, data on the internet is passed from router to router before reaching its final destination, often without the sender or receiver knowing exactly where it has been.

One sophisticated method of cryptography is called a **public key cryptosystem**. It hinges on the fact that there are a considerable number of functions that are easy to compute, but are extremely difficult to invert without some secret information. This scheme was developed by Whitfield Diffie and Martin Hellman.

To use a public key cryptosystem method, each party involved published a function somewhere that anyone can access. There is also an inverse to this function that the party keeps secret. We will discuss later how the person figures out his function's inverse in the first place.

For person B to send a message m to person A, person B computes $f_A(f_B^{-1}(m))$, where f_n is person n 's public function. B then sends this converted message, call it m' to person A, who can decipher it by computing $f_B(f_A^{-1}(m'))$. Notice that in both the encryption and decryption phases, only the intended parties possess all the necessary tools. The receiver is sure the sender is who he thinks it is and vice versa. Thus this method seems fairly ironclad. Outside parties would have to somehow figure out the f_A and/or f_B inverses to tamper with this communication process.

However, there are vulnerabilities with this system. One simple weakness is that a rival could learn your secret by espionage. This would enable decryption of messages meant only for you, and the sending of messages pretending to be you.

Also, someone could convince you that he is the intended recipient, and he could convince the recipient that he is you. In our letter scheme, that means that A thinks C is

B and B thinks C is A, where C is some intruding party. Thus, when B sends a message to A, he will encrypt it as $f_C(f_B^{-1}(m))$. C will intercept it, decrypt it and send $f_A(f_C^{-1}(m_2))$ to A, where m_2 may or may not be the same as the original m . Thus C will intercept all communications, and have the ability to eavesdrop or tamper with the message.

It's amusing that people have actually pulled off the same sort of scheme with direct interactions, specifically in the case of ATM usage. Two individuals were caught putting physical devices that looked like ATMs outside of them, completely concealing the actual machine. Then, ATM users would give their cards and passwords to the fakers, thinking they were the bank. With the user's cards and passwords, the crooks could then convince the real ATM that they were the customers so they could steal from them. This is basically the same situation that we discussed above. This sort of attack is avoided on the internet by use of something called a certificate. However, we will not discuss that here.

We will discuss, however, one method for public key cryptography that was developed here at MIT. It is called RSA encryption, after its developers Rivest, Shamir, and Adelman. As we mentioned, such systems require a function that is hard to invert. For RSA, the function is difficult to invert because it involves factoring a very large number. The first question that probably comes to your mind now is, what exactly is the encryption scheme and how is it performed?

In short, we take the message m , represented as a number, and raise it to the power $z \bmod N$, where N is the product of two large primes p and q . That is the original message is m , the encrypted is $m^z \bmod N$.

This of course raises some more questions. How do we find very large primes? How do we compute exponents mod N ? How can we invert this to recover the original message? And how could this be "broken," or decrypted by a third party?

18.2 RSA Encryption

Our first task is to find two large prime numbers. To do this, we first select a range of where we would like our prime number to be. For example, we may select 25 digits as the relative size of the prime we want. We will then incorporate some random elements as well as some brute force testing to find candidates for primality before we do a conclusive test. We do this because a conclusive test is costly in terms of resource usage, and thus we want to reduce the number of candidates we test.

Specifically, we randomly select the first 22 digits of the prime. We can be reasonably confident that we will find a prime with those first 22 digits, based on the frequency of prime numbers among numbers this size. Then, we take all the numbers with these 22 digits as their most significant digits and rule out the ones divisible by any number under k , for as large a k as we can with our resources. The remaining numbers become our candidates for primality and we will run a separate test on them. Note that we do this because we assume that for 25 digit numbers, our brute force testing will not

be able to test for divisibility by numbers up to the square root of the candidate (which would prove primality).

Before we outline the test for primality and before we analyze how efficient it is and how many numbers we must test, let's take a look at some useful tools.

18.3 Euclid's Algorithm

Euclid's Algorithm allows us to find the greatest common factor/divisor of two positive integers A and B.

We can do this by taking $B \bmod A$ to produce a third number C, provided $A > B$. We repeat this process over and over, finding $D = C \bmod B$, then $E = D \bmod C$, and so on. We can do this easily on a spreadsheet by putting A in cell A1, B in cell A2, and in cell A3 and on, we put:

$$= \text{mod}(A_{j-1}, A_j)$$

Eventually, the algorithm will terminate when A_j is a factor of A_{j-1} . The result of $A_{j-1} \bmod A_j$ will just be 0. At this point, we have our solution to finding the greatest common factor of the original A and B, which is the A_j that terminated the algorithm. Note that we can also work the algorithm backwards to find an expression for the factor as a linear combination of the two numbers A and B.

For example, this is how our spreadsheet would look with 237 and 177 as the original numbers:

237
177
60
57
3
0

By definition of the mod operation, each number n (starting with 60) is the remainder when the number two terms above n is divided by the number one term above n. Thus, we can express each term as a linear combination of the two terms below it. By linear combination we mean that if $A = cX + dY$, then A is a linear combination of X and Y. In this case, since one of the two terms is a remainder, its coefficient is one. That is $177/60 = 2$ remainder 57, thus $177 = 2*60 + 1*57$. As such, we can create the following table:

1. $237 = 1*177 + 60$
2. $177 = 2*60 + 57$
3. $60 = 57*1 + 3$
4. $57 = 3*19 + 0$

In this case, as in all cases, the final A_k (where there are k total numbers) is a factor of A_{k-1} , meaning that the factors that make up A_{k-2} in equation 3 must all be multiples of A_k . Since A_{k-2} and A_{k-1} are multiples of A_k , then equation 2 shows that A_{k-3}

must be a multiple of A_k as well. This logic continues up the list of equations, regardless of how large it is to show that the original two numbers must have A_k as a factor. This is the rationale of our first point (that A_k is the greatest common factor).

Reversing Euclid's algorithm can do something else important for us, which is determining the multiplicative inverse of a number mod another number. To clarify, the multiplicative inverse of a number n is the number m for which $mn = 1$. The same definition holds for modular arithmetic. Thus if $nm \bmod N = 1$, m and n are multiplicative inverses mod N .

To find multiplicative inverses, let's look at a manipulation of the first three equations above, which describes A_j in terms of A_{j-1} and A_{j-2} :

$$\begin{aligned} 60 - 57*1 &= 3 \\ 177 - 60*2 &= 57 \\ 237 - 177*1 &= 60 \end{aligned}$$

By plugging in the equation for 60 into the second equation, and then plugging in the new equation for 57 into the first equation we find that:

$$3 = 237*3 - 177*4$$

Thus, we have the gcd of two numbers in terms of the two numbers themselves. This may not seem like much, but consider the case that the gcd is 1. Then we would have $1 = nA - mB$. Rearranged this would be $1 + mB = nA$. Thus, n and A would be multiplicative inverses mod B . That is, $nA = mB + 1$; thus, $nA \bmod B = 1$. In this particular case, 3 is the gcd of 237 and 177, meaning that no multiplicative inverse exists for 177 in mod 237 terms.

We can outline a general algorithm for rearranging the equations as we have just done in this specific case. In general, we can write each number in the form: $A_{j-2} = c_{j-1} * A_{j-1} + A_j$ where c_j is the integer obtained when dividing A_{j-2} by A_{j-1} . The equations on the previous page are written in this form. We can rearrange them into $A_j = A_{j-2} - c_{j-1} * A_{j-1}$. The goal is to write the gcd as a linear combination of our two original numbers. Thus, we start writing the gcd in the form: $\text{gcd} = L_{j+1} * A_j + S_{j+1} * A_{j+1}$ by using the first form given above. We can then make substitutions using the second equation form given above. That is, we can substitute for A_{j+1} using the equation: $A_{j+1} = A_{j-1} - c_j A_j$. After the substitution, we have: $\text{gcd} = (L_{(j+1)} - c_{(j)} * S_{(j+1)}) * A_{(j)} + S_{(j+1)} * A_{(j-1)}$. In other words, we can keep the original form of $\text{gcd} = L_{n+1} * A_n + S_{n+1} * A_{n+1}$ if we assign $L_j = S_{(j+1)}$ and $S_j = L_{(j+1)} - c_{(j)} * S_{(j+1)}$. We can continue with these substitutions until the gcd is in terms of the original two numbers. That way we can find the multiplicative inverse of one mod the other. This is easily implemented on a spreadsheet. We start with $S = 1$ and $L = 0$, since, the final equation we get when performing Euclid's Algorithm is that gcd (1 in the case of finding multiplicative inverses) multiplied by some constant is equal to $1 * \text{the factor}$

above it, plus zero. This is because the algorithm terminates when a number is a factor of the one before it. As such, Euclid's Algorithm can be reversed.

18.4 Chinese Remainder Theorem

Consider two **relatively prime numbers** A and B . Recall that relatively prime means that A and B contain no common factors besides 1. We will be looking at the remainder of an integer x when it is divided by the product AB .

We can characterize the number x in two ways. We can give its remainder upon dividing by AB and we can give its pair of remainders on dividing by A and dividing by B separately. The **Chinese remainder theorem** states that these two results are equivalent, and that this equivalence is preserved under arithmetic operations.

For example, let $A = 7$, $B = 13$, and $AB = 91$. Any number, say 164, can be described by its remainder when divided by 91, which is 73, or its pair of remainders when divided by 7 and 13, which is (3, 8). The theorem states that these are equivalent characterizations. We can multiply the remainder 73 by 11 and get 803, which is $75 \pmod{91}$ or $(5, 10) \pmod{7 \text{ and } 13}$. Or, we can multiply each number in the ordered pair (3, 8) by 11 and see what happens. We get (33, 88), which $\pmod{7 \text{ and } 13}$ is (5, 10). Thus, we see what we mean by arithmetic operations preserving the remainder relations.

To prove this theorem, note that:

- 1 – Every possible remainder on dividing by the product has a pair of remainders upon dividing by the factors.
- 2 – The number of possible remainders upon dividing by the product is the same as the number of pairs on dividing by the factors. This is true because the number of remainders when dividing by some number N is equal N . Thus the number of remainders when dividing by AB is AB and the number of remainders when dividing by A and B is A for the first, B for the second, for a total of AB possible pairs.
- 3- No two distinct remainders upon dividing by the product can have the same pair of remainders when dividing by the factors. If this occurred, then the difference between the two numbers would have to have a 0 remainder on dividing by each of the factors. Thus, the difference would have a remainder 0 when dividing by the product. Thus, the two seemingly "distinct" remainders are actually the same when taken $\pmod{\text{the product}}$. For example 25 and 49 are two numbers that have the same pair of remainders when dividing by 4 and 6 (1, 1). Of course, the two numbers differ by 24, which is the product of 4 and 6, showing that $25 \pmod{24} = 29 \pmod{24} = 1$.

Thus, there is a one-to-one correspondence between factor remainder pairs and product remainders. Also note that taking remainders on dividing by products has no effect on taking remainders when dividing by factors. That is, the remainder when x is divided by a factor is equal to the remainder when dividing by the factor of the remainder when dividing by the product. For example, the remainder of 57 when dividing by 4 (1), is the same as the remainder when $57 \bmod 24 = 9$ is divided by 4 (1). Thus the preservation of the correspondence of arithmetic operations here is a consequence of the consistency of arithmetic on taking remainders with arithmetic before taking remainders. In other words the correspondence is preserved because both representations have to agree with ordinary arithmetic before taking remainders.

It is generally not easy to find one representation of x given the other, but the fact that they both exist will be very useful to us.

18.5 Taking products mod N

Recall that our methods for encryption and decryption require raising very large numbers to very large powers. Under normal circumstances, this would be preposterous, since the size of the number would get out of hand quickly. However, in this case, we are limiting ourselves to computations using modular arithmetic, or remainders on dividing by some number N . As a result, we can design a very manageable procedure for multiplication.

We first write the factors to be multiplied as binary sequences and construct the product sequence much like we did in calculating the product of polynomials. However, we are no longer doing calculations mod 2. We then take the remainder of the product much like we took the remainder in the polynomial section.

Specifically, we create a remainder table for the number N that we wish to divide our numbers by and take the remainder. That is, if we are using mod N multiplication, we make a remainder table for N . We then take the dot product of the product sequence with the remainder table. Again, we are not using mod 2 ($2=0$) simplifications in this case, any coefficients greater than 1 must be reduced through carrying.

If any of these concepts are unfamiliar to you, such as remainder tables or polynomial multiplication, you should consult the polynomial encoding sections of these notes (section 10).

Suppose for example, that N is a k digit number to some base.

Then the product of two remainders mod N will in general be a number requiring at most $2k-1$ coefficients of powers of the base. These coefficients will be convolutions of the coefficients of the two factors as they are in all multiplications.

The remainders of all powers that are smaller than N will be themselves. The remainders of higher powers can be computed by the same general approach that we used

to compute the remainder table, except that we cannot use $2=0$ and should carry coefficients that are larger than the base into the next power, as we normally do in multiplication. However, we will not go into details here.

18.6 Raising numbers to a power

Being able to multiply two numbers mod another number enables us to do what we originally set out to do. We can now raise numbers to exponents. Raising a number x to power M mod N will require between $\lg M$ and $2\lg M$ multiplications.

We start with a binary representation of M . For example, if M is 26, we write it as 11010. Then, we successively raise x to powers given by the first k bits of binary sequence, with k starting at 1 and going to the total number of bits. That is, we raise x to the power 1, then to the 11, then to 110, then to 1101, and finally to 11010. We do this because it is easy to form each successive power based on the previous one.

To convert x^A to x^B when B is the same binary number as A except it has an extra 0 appended to it, we simply square x^A . This is because the difference between A and B is that all the ones in B represent a power that is twice as much as in A . For example, if $A = 11$, $B = 110$. A is 3, B is 6. If B is the same as A except B has an extra 1 appended to it, then we square x^A and multiply by x . This is because $B = 2A + 1$. All these multiplications are done, of course, mod N .

To finish our example, when $M = 26 = 11010$, we start with the number 1 and perform our algorithm. Square 1 and multiply by x , then square and multiply by x , then square, then square and multiply by x , then square and we're done. Spreadsheet implementation is trivial and we will not discuss this here.

At a minimum, the number of multiplications is one per bit of M (when the bits are 0). At maximum, the number of multiplications is two per bit of M (when the bits are 1). Since there should be $\lg M$ bits, the number of multiplications is between $\lg M$ and $2\lg M$.

18.7 Testing for primality

With these tools under our belt, we can now return to the original problem we were working on: testing candidates for primality. First let's examine how primes and non-primes differ.

Given a number N , the numbers less than N that are relatively prime to N form a group under the operation of multiplication mod N . The product of any two such numbers is still relatively prime to N and subtracting any multiple of N from the product will leave the resulting remainder relatively prime to N . Thus this set is closed under multiplication and forms a group.

If N is prime, every number less than N is relatively prime to it, making the order of the group $N-1$. If N has no square factors, the order of the groups is the product over the

prime factors p_j of N or $(p_j - 1)$. For example, if N is 15, the prime factors are 5 and 3, and the order of the group is $4 \cdot 2 = 8$.

Recall from the notes of group theory that the order of a specific element x of a group is the power that you must raise it to in order to arrive at the group's identity element. This is also the order of the subgroup that x generates. Thus, by Lagrange's Theorem, it must be a divisor of the order of the group.

Thus, if N is prime, the power that you must raise all group elements to in order to get the identity is a divisor of $(N-1)$. Let's call the divisor $(N-1)/m$. Thus, if you raise your element to the $(N-1)$ power, you should get the identity element (which is 1). This is because $x^{(N-1)} = 1^m = 1$. Therefore, if we take any non-zero $x \pmod N$ and raise it to the $(N-1)$ power mod N , the remainder will be 1 if N is prime.

For most non-prime N this statement will not be true. For example, if N is the product of two primes, p and q , then there will be x 's whose order mod N are $p-1$ and $q-1$, and the statement that $p-1$ is a divisor of $pq-1$ is the statement that $p-1$ divides $q-1$, and vice versa, which cannot both be true.

Unfortunately, there are non-prime numbers, called **Carmichael numbers** for which any x raised to the power $(N-1)$ is 1 mod N , so this test is not completely reliable for primality.

However, primes differ from non-primes in another respect: the remainders on dividing by a prime p form a **field**, and in it every quadratic equation, such as $x^2-1=0 \pmod N$ has at most two solutions, as was true when we considered remainders on dividing by polynomials.

However, if N has two or more distinct prime factors, then this particular equation will have at least 4 solutions.

To rationalize this let A and B be relatively prime factors whose product is N . Then the four numbers whose representations as pairs of remainders on dividing by A and by B are $(-1,-1), (1,1), (1,-1)$ and $(-1,1)$ will all obey the equation $x^2=1$. For example, if $N = 15$, $A = 3$, and $B = 5$, the four numbers are 1, 4, 11, and 14. You can verify these yourself. Since the pairs of remainders mod A and mod B obey the equation, the Chinese Remainder Theorem tells us that the numbers represented by those pairs must also obey the equation. There must be 8 such solutions if N has three distinct prime factors, and so on. $(-1,-1)$ is -1 , and $(1,1)$ is 1 here.

With this information we should be able to find a strong test for primality that is better than just raising a number x to the power $(N-1) \pmod N$.

We will however start by raising x to the power $(N-1)$. If the result is not 1, we immediately know that N is not a prime. If it is 1, we will examine the outcomes of intermediate results to our exponent method, specifically, the last result before $(N-1)$.

Remember, the last intermediate result before raising to the $(N-1)$ power is not the $(N-2)$ power when using our method of raising numbers to powers. Using the method we detailed in the previous section, since $(N-1)$ is an even number, the last intermediate result is $(N-1)/2$.

If the result of $x^{(N-1)} \bmod N$ is 1, we look at the last result before the 1. If N is prime, the last predecessor of the 1 must be 1 or -1 . If the second to last result y is not 1 or -1 , this means that $y^2 \bmod N = 1$. This is impossible if N is prime. This is because, as we stated before $x^2 = 1 \bmod N$ has only two solutions if N is prime, if it is a Carmichael number, it may have more.

This provides the basis for our primality test. We take some value for x , compute $x^{(N-1)} \bmod N$ using our method from section 18.6, and examine the results of the computation. If the result is not 1, N is not prime. If the result is 1, but the intermediate results show that the square of some number other than -1 or 1 equals 1, then the number is not prime. If the number passes both of these tests, we test another value of x .

We do this for the first 20 or so primes less than N , and if it passes all the time, we declare N a prime. The chance of N passing these tests is extremely small ($\sim 10^{-6}$) if N is not prime.

This probability is extremely small because there are only two ways N could pass the test if it is not prime. The first is that all the odd powers of x encountered in the computation of raising to the $(N-1)$ power are ± 1 . In other words, all the square roots encountered back to the last initial segment of the binary representation $N-1$ are ± 1 , provided that the binary segment ends in 1 (it is an odd power). The other way a non-prime could pass is that some squaring produces a -1 , whose square is 1 in the formation of $x^{N-1} \bmod N$.

To address these two cases, we again speak in terms of remainder pairs mod A and B for a non-prime AB . In order for N to pass the test in the first case, x must be $(1, 1)$ or $(-1, -1)$, so that each time it is raised to an odd power, the result is one of those two remainder pairs. That is, if we take 1 or -1 and square it and multiply by $(-1, -1)$, we get $(-1, -1)$. Equally as often, the x we choose can be $(-1, 1)$ or $(1, -1)$, which will not result in 1 or -1 when x is raised to an odd power. Thus, half the x 's of this kind will pass and half will fail.

In the second case, we know that x to some even power is -1 , or $(-1, -1)$ in remainder pair form. Thus, $y^Q = -1 \bmod N$ where Q is even. We can then find a matching z to this y , such that $z^Q = (1, -1)$ and pair every y to a z . Thus, x values that have y as an intermediate step can be paired to those that have z as an intermediate step. Therefore, there is an equal number of x 's that pass the primality test as there is that fail it.

These two ways that N could pass that test imply that if x is chosen at random and $x^{N-1} = 1 \bmod N$, at least half the time the test will fail, even for Carmichael numbers. If

the test is repeated independently 20 times, the chance of passage every time is less than 2^{-20} .

Recently, someone announced an algorithm for primality testing that provably always works without making random choices repeatedly as done here, but the method we just developed works fine in practice.

18.8 Implementing RSA

Using the tools we have developed, we can implement the RSA encryption algorithm.

First we find two large primes using the procedure we developed in sections 18.2 and 18.7. Call these p and q and their product is N . The product $(p-1)(q-1)$ is the order of the group of remainders that are relatively prime to N (prove this as an exercise). We can also find a number z that is relatively prime to $(p-1)(q-1)$. We find this by guessing and then checking using Euclid's algorithm.

Next, we use Euclid's algorithm in reverse (outlined in 18.3) to find the multiplicative inverse of $z \bmod (p-1)(q-1)$. Call this y , thus $yz = 1 \bmod (p-1)(q-1)$.

In our public key cryptosystem, the public key will be the numbers N and z ; p and q are kept private. Thus, someone can encode a message using $m^z \bmod N = c(m)$. Since p and q are not public, only he who knows them can calculate y so that they can decrypt the message using $c(m)^y \bmod N = m$.

Exercises

- Exercise 1* Verify that the order of the group of remainders relatively prime to $N = pq$ is $(p-1)(q-1)$, given that p and q are prime.
- Exercise 2* Set up a spreadsheet that finds the gcd of two inputted numbers, and expresses that gcd as a linear combination of the two numbers.
- Exercise 3* Set up a spreadsheet that raises two numbers to a power mod N for the highest value of N possible on your machine (but no more than 20 digits) using ordinary multiplication of numbers (You have to multiply numbers which requires twice the precision of the length of N).
- Exercise 4* Set up a spreadsheet that checks 200 numbers in a row to eliminate those with any small prime factors (where small means under 200).
- Exercise 5* Set up a spreadsheet that examines candidates for primality from your exercise-4 spreadsheet to test them for primality (expand $N-1$ in binary

and raise some x to the power $N-1 \pmod N$ by the procedure described in the notes).

-Steven Kannan