Number Theory Background

Prime Numbers

A prime number is an integer 2 or greater that is divisible by only 1 and itself, and no other positive integers. Prime numbers are very important to public key cryptography.

Fermat's Theorem

One really neat property of prime numbers is as follows:

For all prime numbers p and positive integers a such that gcd(a, p) = 1,

$a^{p-1} \equiv 1 \pmod{p}$.

The proof is as follows:

Let S = {1, 2, 3, …, p-1}

Now, let's create a set S' where each value is a value multiplied in S times an integer a, such that gcd(a, p) = 1. So S' looks like this:

S' = {a, 2a, 3a, …, (p-1)a}

It turns out that the remainders, when each value in S' is divided by p form the set S.

To prove this, we must show the two following things about the set S':

(a) No value in S' is divisible by p.
(b) For all distinct items x and y in S', x and y leave different remainders when divided by p.

The first is fairly easy to see. There's a theorem that if a prime number p divides into a product of integers ab, then either p divides evenly into a or p divides evenly into b. But if we take a look at the set of values in S', each is the product of a and an integer i, where i is in between 1 and p-1. It's clear that p does not divide any of these components. Thus, it follows that p can not divide any of the separate terms. These means that every item in S', when divided by p, leaves a remainder that is not 0, so the possible remainders are {1, 2, 3, …, p-1}.

To see (b), let's do a proof by contradiction. Assume the opposite, that two distinct items in S' are equivalent mod p. It follows that there are integers i and j ($1 \leq i < j \leq p-1$) such that

$aj \equiv ai \pmod{p}$

Now, let's do some algebra:

$aj - ai \equiv 0 \pmod{p}$

$a(j - i) \equiv 0 \pmod{p}$

By definition of divisibility, we have that p | (a(j-i)). Since p is prime, it follows that either p | a or p | (j - i). The first is not possible because we are given that gcd(a, p) = 1. Thus, it follows that p | (j - i), but this contradicts the fact that j - i > 0 and j - i < p-2, since p does not divide any of these integers. This is a contradiction. It follows that ai and aj can not be equivalent mod p and that no pair of values in the set S' are equivalent mod p.

Since none of the values of S' are equivalent to 0 mod p and there are p-1 values in S', it follows that the values of S' are equivalent to 1, 2, 3, …, p-1 mod p. Thus, the sets S and S' are equivalent mod p.

Since the sets are equivalent mod p, their products are equivalent mod p. This gives us:

$$\prod_{i=1}^{p-1} ai \equiv \prod_{i=1}^{p-1} i \ (mod\ p)$$

Subtract the term on the right over to the left:

$$\prod_{i=1}^{p-1} ai - \prod_{i=1}^{p-1} i \equiv 0\ (mod\ p)$$

Factor out (n-1)! from both of the products:

$$\prod_{i=1}^{p-1} i\ (\prod_{i=1}^{p-1} a - 1) \equiv 0\ (mod\ p)$$

Applying the definition of product, we get:

$$(p-1)!\ (a^{p-1} - 1) \equiv 0\ (mod\ p)$$

By definition of divisibility, we have $p|\ [(p-1)!\ (a^{p-1} - 1)]$. Since p is prime, it follows that either $p|(p-1)!$ or $p|(a^{p-1} - 1)$. The former isn't true since (p-1)! Only has divisors in between 1 and p-1, inclusive. It follows that the latter must be true. Writing this in its equivalent mod form we get:

$$(a^{p-1} - 1) \equiv 0\ (mod\ p)$$

Adding 1 to both sides we get:

$$a^{p-1} \equiv 1\ (mod\ p)$$

Euler Phi Function

First, let's define the Euler ϕ(phi) function:

$\phi(n)$ = the number of integers in the set {1, 2, ..., n-1} that are relatively prime to n.
$\phi(p) = p - 1$, for all prime numbers
$\phi(pq) = (p-1)(q-1)$, where p and q are distinct primes. Here is a derivation of that result:

We want to count all values in the set {1, 2, 3, ..., pq –1} that are relatively prime to pq. Instead, we could count all value in the set NOT relatively prime to pq. We can list these values:

p, 2p, 3p, ... , (q-1)p

q, 2q, 3q, ... (p-1)q

Note that each of these values are distinct. To notice this, see that no number of the first row is divisible by q and no number on the second row is divisible by p. This ensures that there are no repeats on both rows. since p and q are relatively prime, in order for q to be a factor of a number on the first row, it would have to divide evenly into either 1, 2, 3, ... q-1. But clearly, it does not. The same argument will show that none of the values on the second row are divisible by p.

Finally, we can count the number of values on this list. It's $(q-1) + (p-1) = p + q - 2$.
Now, in order to find $\phi(pq)$, we must subtract this value from pq –1 . So, we find:

$\phi(pq) = (pq - 1) - (p + q - 2) = pq - p - q + 1 = (p - 1)(q - 1)$.

Now, let's try to derive a more general result to calculate the ϕ for all positive integers.

First, we will extend our formula $\phi(p) = p - 1$, for all prime numbers, to numbers of the form $\phi(p^n)$. This extension is rather simple because for a number to NOT be relatively prime to p^n, it must be divisible by p. Looking at the list: 1, 2, 3, …, p, …, p^n-1, there are exactly $p^{n-1} - 1$ values on the list divisible by p. (These values are p, 2p, 3p, …, $(p^{n-2} - 1)p$.) Thus, we find that $\phi(p^n) = p^n - 1 - (p^{n-1} - 1) = p^n - p^{n-1}$.

Next, we generalize the result $\phi(pq) = (p - 1)(q - 1) = \phi(p)\phi(q)$ for two primes p and q to any number that is the product of relative prime values, m and n. This extension will take a bit more work. We must count the number of values in the set {1, 2, 3, …, mn – 1} that are relatively prime to mn. Let us write them out in a chart as follows:

1	2	3	4	…	m
m+1	m+2	m+3	m+4	…	2m
…					
(n-1)m+1	(n-1)m+2	(n-1)m+3	(n-1)m+4		nm

We must "cancel out" any term in this grid that is NOT relatively prime to either m or n.
First, let's cancel out the terms NOT relatively prime to m. Quickly note that if some value r is NOT relatively prime to m, then km+r is not either. Thus, if there is some value r in between 1 and

m inclusive that shares a common factor with m, then EVERY value in its column shares a common factor with m. Thus, there will be $\phi(m)$ columns that not canceled out. The other columns are completely canceled out.

Now, consider the remaining columns. We need only to look for values that share a common factor with n in these columns. Each column takes the following form:

r, m+r, 2m+r, 3m+r, …, (n-1)m+r.

Now, we will prove that each of these numbers is distinct mod n.

Assume to the contrary, that two values on the list are equivalent mod n. Let these two values be

im+r and jm+r, for $0 \leq i < j < n$. Thus, we have:

$im + r \equiv jm + r \pmod{n}$
$jm - im \equiv 0 \pmod{n}$
$m(j - i) \equiv 0 \pmod{n}$

It follows that n divides evenly into $m(i - j)$. But, we are given that $\gcd(m,n) = 1$. This implies that $n \mid (i - j)$. But, this is impossible because $0 < j - i < n$. This is our contradiction. Thus, it follows that each of the n numbers on that list is not equivalent mod n. Thus, there is exactly 1 number for each residue class mod n in the list. It follows that EXACTLY $\phi(n)$ of these are divisible by n. Finally, if we take a look at the numbers not crossed out, there are exactly $\phi(m)\phi(n)$ of them.

Here is a quick example with m = 8 and n = 15. All crossed out numbers are underlined. We have $\phi(8) = 4$ columns of numbers not crossed out.

1	2	3	4	5	6	7	8	In each column there are
9	10	11	12	13	14	15	16	$\phi(15) = 8$ numbers not
17	18	19	20	21	22	23	24	crossed out.
25	26	27	28	29	30	31	32	
33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	
49	50	51	52	53	54	55	56	
57	58	59	60	61	62	63	64	
65	66	67	68	69	70	71	72	
73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	
89	90	91	92	93	94	95	96	
97	98	99	100	101	102	103	104	
105	106	107	108	109	110	111	112	
113	114	115	116	117	118	119	120	

Now, given these two results, we can derive a formula for $\phi(n)$ for any positive integer n. Given n's prime factorization, one can simply calculate the phi function of each prime factor separately and multiply these all together.

For example, $\phi(2^5 \times 3 \times 7^2) = \phi(2^5)\phi(3)\phi(7^2) = (2^5 - 2^4)(3 - 1)(7^2 - 7) = 16(2)(42) = 1344$.

Euler's Theorem

Euler's Theorem: If gcd(a,n) = 1, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Definition of a reduced residue system modulo n: A set of $\phi(n)$ numbers $r_1, r_2, r_3, \ldots r_{\phi(n)}$ such that $r_i \neq r_j$, for all $1 \leq i < j \leq \phi(n)$ with $\gcd(r_i, n) = 1$ for all $1 \leq i \leq \phi(n)$.

Theorem about reduced residue systems: If $r_1, r_2, r_3, \ldots r_{\phi(n)}$ is a reduced residue system modulo n, and gcd(a,n) = 1, then $ar_1, ar_2, ar_3, \ldots ar_{\phi(n)}$ is ALSO a reduced residue system modulo n.

Proof: We need to prove two things in order to verify the theorem above:

1) $ar_i \neq ar_j$, for all $1 \leq i < j \leq \phi(n)$
2) $\gcd(ar_i, n) = 1$ for all $1 \leq i \leq \phi(n)$

Proof of 1:

Assume to the contrary that there exist distinct integers i and j such that $ar_i \equiv ar_j \pmod{n}$. We can deduce the following:

$ar_i \equiv ar_j \pmod{n}$
$(ar_i - ar_j) \equiv 0 \pmod{n}$.
$n \mid (a(r_i - r_j))$

We know that gcd(a,n) = 1. Thus, based on a theorem proved earlier, it follows that $n \mid (r_i - r_j)$. But, this infers that $r_i \equiv r_j \pmod{n}$. This contradicts our premise that $r_1, r_2, r_3, \ldots r_{\phi(n)}$ is a reduced residue system modulo n. Thus, we can conclude that $ar_i \neq ar_j$, for all $1 \leq i < j \leq \phi(n)$.

Proof of 2:

Since gcd(a,n)=1 and $\gcd(r_i,n)=1$, it follows that n shares no common factors with a or r_i. Thus, it shares no common factors with their product and we can conclude that $\gcd(ar_i, n) = 1$ for all $1 \leq i \leq \phi(n)$.

Now, we will use this theorem to prove Euler's theorem:

Let $r_1, r_2, r_3, \ldots r_{\phi(n)}$ be a reduced residue system modulo n, and gcd(a,n)=1. Then we have that $ar_1, ar_2, ar_3, \ldots ar_{\phi(n)}$ is a reduced residue system modulo n. Since both are reduced residue systems modulo n, we know that the their products are equivalent mod n:

$$\prod_{i=1}^{\phi(n)} ar_i \equiv \prod_{i=1}^{\phi(n)} r_i \pmod{n}$$

$$\prod_{i=1}^{\phi(n)} ar_i - \prod_{i=1}^{\phi(n)} r_i \equiv 0 \pmod{n}$$

$$a^{\phi(n)} \prod_{i=1}^{\phi(n)} r_i - \prod_{i=1}^{\phi(n)} r_i \equiv 0 \pmod{n}$$

$$\left(\prod_{i=1}^{\phi(n)} r_i\right)(a^{\phi(n)} - 1) \equiv 0 \pmod{n}$$

Thus, we have that n divides this product. But, we know that gcd(r_i, n) = 1 for each value of i. Thus the first large product of $\phi(n)$ terms is relatively prime to n. It follows that n divides the last factor:

$n \mid (a^{\phi(n)} - 1)$
$a^{\phi(n)} \equiv 1 \pmod{n}$, proving Euler's Theorem.

Wilson's Theorem
The theorem follows rather simply from some of our following work:

$(p - 1)! \equiv -1 \pmod{p}$ for all primes p.

This result can be verified for p = 2. Now, let's consider all odd p. Since each value 1, 2, …, p – 1 is relatively prime to p, each has an inverse mod p. We know that the inverse of 1 is 1 and the inverse of p – 1 is p – 1. But, for each other value on the list, its inverse is different than itself.

To see this, let's directly set up an equation for a value k that is its own inverse mod p:

$k^2 \equiv 1 \pmod{p}$
$k^2 - 1 \equiv 0 \pmod{p}$
$(k - 1)(k + 1) \equiv 0 \pmod{p}$

This implies that $p \mid (k - 1)$ or $p \mid (k + 1)$. These are exactly the two values we have written above as having self inverses.

Now, consider the product

1 x 2 x 3 x 4 … x (p – 1)

1 x (p – 1) x (2 x 3 x 4 … x (p – 2))

Each of the terms in the second set of parentheses (there are an even number of them), have their inverses mod p in that set. We can pair up these values such that

$$\begin{aligned} 1 \times (p - 1) \times (2 \times 3 \times 4 \ldots \times (p - 2)) &\equiv 1 \times (p - 1) \times 1 \times 1 \ldots \times 1 \pmod{p} \\ &\equiv (p - 1) \bmod p \\ &\equiv -1 \pmod{p} \end{aligned}$$

Primality Testing

In order for a RSA system to be created, one needs a way of creating large prime numbers. The standard way to do this would be to generate a random odd integer and then test to see if it's prime or not.

The difficulty with this method is that testing for primality is a time-consuming task. The standard method of trial division would take thousands of years with some of the numbers we want to deal with. Recently, there has been the discovery of a polynomial-time algorithm to test for primality, but for common everyday procedures, it is also too time-consuming.

It turns out that the most practical solution to this problem is utilizing a probabilistic algorithm. A probablistic algorithm is one that doesn't ALWAYS return the correct answer, but does so with some probability. This may sound unacceptable, but amazingly enough, we can utilize a test that is 75% accurate to create an overall algorithm that is accurate nearly every time.

Our key goal is to differentiate/categorize integers into one of two categories: prime or composite. If we can find a property that one group has that the other group doesn't (most of the time), then that can be our litmus test for categorizing integers. This isn't the most intuitive litmus test, but it turns to work out quite well. Earlier in these notes we found out that for all $1 < a < p$, where p is prime,

$$a^{p-1} \equiv 1 \bmod p$$

This is a statement true for all primes, but as it turns out, is false for most values of a, for most composites. (Remember for composite numbers, n, the correct exponent is phi(n), which is strictly less than n-1.)

Thus, the intuitive idea for our algorithm for testing if n is prime or not is as follows:

1) Pick a random value a, $1 < a < n$.
2) Calculate $a^{n-1} \bmod n$.
3) If the answer is not 1, answer composite.
4) If the answer is 1, answer "probably prime."

We know that if this calculation does not yield one, then the number we are testing is DEFINITIVELY composite, since ALL primes would yield one.

But, if we get one, we can't be positive that the number is prime, since there are some composites paired with some values of a that result in an answer of one as well.

It turns out that the probability this algorithm is incorrect when it answers "probably prime" is no more than 1/2 (unless we are dealing with Carmichael numbers).

Thus, if we want to increase our confidence in the "probably prime" answer, we can simply repeat the test multiple times. Here is some pseudocode with the idea:

```
boolean isPrime(int n) {
  for (int i=0; i<50; i++) {
     int a = rand()%n;
     if (pow(a,n-1)%n != 1)
       return false;
  }
  return true;
}
```

Basically, all we do is test 50 random values of a. If any of them triggers a value other than one, we can be sure the number is composite. The probability a composite number gives the answer of one 50 straight times is less than $(.5)^{50}$.

The only exception to this is a special set of (infrequent) numbers known as Carmichael numbers. These are composite numbers for which each value of a that is relatively prime to n always yields 1 in this computation.

To thwart Carmichael numbers, the Miller-Rabin test utilizes a further property of Fermat's theorem. In particular, each possible value a has an "order" mod p. The order is simply the smallest exponent that a must be raised to, to obtain 1 mod p. From this point on, the modular exponentiation values cycle. Here are a couple examples:

$2^1 = 2 \bmod 7$	$3^1 = 3 \bmod 7$	
$2^2 = 4 \bmod 7$	$3^2 = 2 \bmod 7$	
$2^3 = 1 \bmod 7$	$3^3 = 6 \bmod 7$	
$2^4 = 2 \bmod 7$	$3^4 = 4 \bmod 7$	
$2^5 = 4 \bmod 7$	$3^5 = 5 \bmod 7$	(order of 2 mod 7 is 3,
$2^6 = 1 \bmod 7$	$3^6 = 1 \bmod 7$	order of 3 mod 7 is 6.)

Let k be the order of a mod p, for some prime p. What often happens is that $a^{k/2} = -1 \bmod p = (p-1) \bmod p$, if k is even. (This is true for the second example above.) But, for Carmichael numbers, this property doesn't typically hold.

The Miller-Rabin test utilizes this fact. Here is the algorithm for testing if n is prime:

1. write $n - 1 = 2^k m$, where m is odd.
2. choose a random a, $1 < a < n$.
3. Compute $b = a^m \bmod n$.
4. if b == 1, answer probably prime and quit.
5. for (i=0; i<k; i++)
6. if (b = -1 mod n)
   answer probably prime and quit.
   else
   $b = b^2 \bmod n$
7. if you get here, answer "composite"

(taken from Cryptography: Theory and Practice by Stinson)

The basic rationale here is that if we look at the following list of numbers mod n:

$a^m, a^{2m}, a^{4m}, \ldots, a^{(n-1)/2}$

for a prime number, either the first one will be 1, or one of the values on the list will be -1. If this isn't true, the number is definitively composite. Furthermore, these restrictions are more stringent than the original, so fewer composite numbers will be able to pass this test. In particular, this test thwarts Carmichael numbers.

The error of this algorithm is at most 25% (better than the previously stated 50%). Thus, if we run this algorithm 50 times and it reports "probably prime", we can be sure with probability $1 - (.25)^{50}$ that the number is indeed prime.

Fast Modular Exponentiation

In order to do all of the items mentioned in this lecture, it's necessary to calculate a modular exponentiation (with a rather large exponent) very quickly.

The typical iterative algorithm:

```
int modExp(int base, int exp, int n) {
  int ans = 1;
  for (int i=0; i<exp; i++)
    ans = (ans*base)%n;
  return ans;
}
```

Is too slow because the loop will run exp number of times, and that could be absolutely huge. (Think tens of thousands or years or more...)

Instead, we must make use of "intermediate" computations. For example, if I know that $a^{5000} = 3 \bmod n$, I can immediately calculate that $a^{10000} = 9 \bmod n$, since $3^2 = 9 \bmod n$. (Basically, all I am doing is squaring both sides of the first equation to yield the second equation.) This one quick step saved 5000 multiplications. Here's a quick algorithm that encompasses this idea:

```
int fastModExp(int base, int exp, int n) {
  if (exp == 0) return 1;
  if (exp == 1) return base%n;
  if (exp%2 == 0) {
    int temp = fastModExp(base, exp/2, n);
    return (temp*temp)%n;
  }
  return (base*fastModExp(base, exp-1, n))%n;
}
```

The run-time of this algorithm is logarithmic in the value of the exponent, exp, instead of linear. This is a huge improvement. Consider exp = 10^{20}, $\log_2 10^{20} = 66.4$, which is much, much, much, much smaller than 10000000000000000000.

Factoring Algorithms

One of the ways that RSA could potentially be broken is through factoring the public key n. Once an opponent has this, they can go through the same steps the person creating the keys went through to obtain d. As of right now, there aren't any factoring algorithms that reliably factor a large composite number into its two prime components in a reasonable amount of time. Here we will look at two fairly elementary algorithms for factoring that are an improvement over trial division:

The Fermat Factoring Algorithm is hinged upon the following factoring fact:

$x^2 – y^2 = (x + y)(x – y)$.

Since we are assuming that our goal is to factor an integer that is the product of two large odd (prime) numbers, we know that the difference between the two factors will be even. Since this is the case, there MUST exist and x and y that satisfy the following equations:

$x + y = p$
$x – y = q$, $N = pq$, where N is the number of factor, and p and q are both large primes.

Namely, x is the "halfway" point in between p and q, and y is the distance from this halfway point to both p and q.

So the idea is as follows: Imagine you are trying to factor N = 26441.

The first thing we know is that x > sqrt(26441), since we must solve

$N = 26441 = x^2 – y^2 = (x + y)(x – y)$, and y^2 is positive.

sqrt(26441) is 162.6. Thus, our first possible value for x is 163. Try it out:

$26441 = 163^2 – y^2 \quad \rightarrow y^2 = 163^2 – 26441 = 128$ (Not a perfect square)

Now, keep on trying successive values of x:

$164^2 – 26441 = 455$ (Not a perfect square)
$165^2 – 26441 = 784$ (This is 28^2, so we are done. x = 165, y = 28, p = 193, q = 137.)

Thus, we find 26441 = 193 x 137.

This algorithm will ALWAYS succeed, but sometimes, its run time will be prohibitive. In particular, the algorithm does well when the two factors are very close to each other in magnitude. It does poorly if they are far apart. Keep in mind that two twenty digit numbers, such as 10^{19} and $9x10^{19}$ are very far apart. (The difference between the two is $8x10^{19}$.)

The Pollard-Rho Factoring Algorithm works based on the notion of order previously described. But, unlike the Fermat algorithm, it may not succeed in all instances. The idea is to create a sequence of numbers that will eventually cycle, when considered mod n. If it cycles mod n, it must also cycle mod p, (where p is one of n's two distinct prime divisors.) Also, it is likely that the cycle mod p has a shorter length than the cycle mod n. If we can detect the cycle mod p (even though we don't know p), perhaps we can utilize that information to get p. Here's the algorithm:

1. let a = 2, b = 2.
2. while (true) {
    a = a^2 + 1 mod n
    b = b^2 + 1 mod n
    b = b^2 + 1 mod n
    d = GCD(a – b, n)
    if (1 < d && d < n) return d;
    if (d == n) return failed;
  }

The basic idea is that both a and b are taken from the following sequence of numbers: 5, 26, 677, ... In each iteration, a is the first number, then the second number, then the third number, etc and b is the second number, then the fourth numberm then the sixth number, etc. When we calculate a – b, we are calculating the difference between successive terms, then the second and fourth term, then the third and sixth term, etc. Essentially we are trying different cycle lengths in the sequence, it the very worst case, the sequence will cycle every n-1 values (which is an exceedingly long time), but in most instances it should cycle much more quickly. If the cycle for p is shorter, then the algorithm will work.