RSA Encryption

Kurt Bryan

1 Introduction

Our starting point is the integers

$$\ldots, -2, -1, 0, 1, 2, \ldots$$

and the basic operations on integers, $+, -, \times, \div$. The basic material could hardly be more familiar. We denote the set integers with the letter $\mathbf{Z}$ and the set of positive integers by $\mathbf{Z}^+$.

1.1 Divisibility

Given two integers a and b we write that $a|b$ (and read "a divides b") if $b = ka$ for some integer k. For example, $7|21$ because $21 = (3)(7)$. Some simple facts about divisibility:

1. If $a|b$ then $a|(cb)$ for any integer c.
2. If $a|b$ and $a|c$ then $a|(xb + yc)$ for any integers x and y.
3. If $a|b$ and $b|c$ then $a|c$.

These are all very easy to prove—just appeal to the definition of $a|b$.

Of course, for arbitrary integers a and b we can't expect $a|b$, i.e., $b = qa$ for some integer q. But we do always have

The Division Property: Given any two integers a and b we can always write

$$b = qa + r$$

where q and r are unique integers and $0 \leq r < |a|$. The number q is called the *quotient* and r is called the *remainder*.

This is also called the *Division Algorithm*. It's a fact you've been familiar with since the 3rd grade. I won't bother to give a proof yet.

1.2 Prime Numbers

A number p whose only positive divisors are 1 (which divides anything) and p (itself) is called a *prime*. Numbers which are not prime are *composite*. The first 10 primes are $2, 3, 5, 7, 11, 13, 17, 19, 23, 29$.

The first, most important fact about primes numbers and the integers is

The Fundamental Theorem of Arithmetic: Every integer $n > 1$ can be factored into a product of prime numbers. The factorization is unique, aside from the order of the factors.

The Fundamental Theorem of Arithmetic may seem obvious—how could it be any other way? But in fact there are other "number systems" which look like the integers (with primes, factorization, etc.) but for which unique factorization FAILS. Here's a simple example: Let $2\mathbf{Z}$ denote the set of even integers. You can easily verify that $2\mathbf{Z}$ is closed under addition and multiplication. Call a number $n \in 2\mathbf{Z}$ *composite* if it's a product of two other numbers in $2\mathbf{Z}$, *prime* otherwise. It's easy to think of some primes in $2\mathbf{Z}$: $2, 6, 10, \ldots$. In fact, you can check that anything of the form $4k + 2$ is prime. Similarly, anything of the form $4k$ is composite. Now notice that 60 factors into primes in two ways, as

$$60 = 6 \cdot 10 = 2 \cdot 30.$$

Another such "number system" is $H = \{4k + 1; k \in \mathbf{Z}\}$, sometimes called the Hilbert numbers. You can check that H is closed under multiplication and that every element of H is either composite (factors into other elements of H) or prime (not composite). But elements of H need not factor uniquely into primes.

Fun Problem

- Find the smallest element of H which factors into primes in two different ways.

Proof of the Fundamental Theorem of Arithmetic: (Based on Giblin, *Primes and Programming*, Cambridge University Press) First we'll show that any integer can be factored. Start with some integer $n > 1$. If n is prime, we're done. Otherwise $n = ab$ where $a, b > 1$. If a and b are prime, we're done. If one or both are not prime apply the same argument to each piece.

For example, if a is prime but b is not then we can write $b = b_1 b_2$, and so $n = ab_1 b_2$. If all pieces are prime we're done, otherwise apply the argument again to any composite factor. At each stage the composite factors decrease by at least a factor of 2. It's clear this can continue for only a finite number of steps before all of the factors are prime.

Proving that the factorization is unique is a little harder. The proof is by contradiction. Suppose that at least one integer greater than one can be factored in TWO ways. Then there is a smallest integer which factors in two ways—let n be that integer, and suppose

$$n = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$$

where all p's and q's represent primes. Of course all of the p's and q's must be distinct, for if, for example, $p_1 = q_k$ for some k then we could divide both sides by p_1 and find a smaller n with two distinct factorizations. We can suppose that $p_1 < q_1$. Start with $p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$ and subtract $p_1 q_2 q_3 \cdots q_s$ from both sides to obtain

$$p_1(p_2 p_3 \cdots p_r - q_2 q_3 \cdots q_s) = (q_1 - p_1) q_2 q_3 \cdots q_s \tag{1}$$

Define a new integer N by $N = (q_1 - p_1) q_2 q_3 \cdots q_s$ (N equals both sides of the above equation). Obviously $1 < N < n$, so N must factor uniquely into prime numbers. Now, we can find a prime factorization of N which contains the prime p_1—just finish factoring the remaining stuff in the parentheses on the left side of equation (1). But if we finish the factorization of N using the right side of (1) (by factoring $q_1 - p_1$), this leads to a factorization which cannot include p_1, since $q_1 - p_1$ is not divisible by p_1. This means that $N < n$ factorizes into primes in two different ways, contradicting our choice of n as the smallest such example. Thus the unique factorization must be true for all integers.

Fun Problems

1. Look at the proof of the Fundamental Theorem of Arithmetic—will the part which proves every number factors into primes work for $2\mathbf{Z}$? I'm not talking about the uniqueness part, just that there is a factorization.

2. Chase through the proof of the Fundamental Theorem for $2\mathbf{Z}$—where does it fail? It might be helpful to apply the method of proof to $n = 60$.

How many primes are there? It has been known for over 2000 years that

Theorem: (Euclid) The number of primes is infinite.

Proof: The proof is by contradiction. Obviously there is at least one prime, namely 2. Suppose that the largest prime is p. Form the number

$$N = 2 \cdot 3 \cdot 5 \cdots p + 1. \tag{2}$$

Obviously $N > p$, so by assumption N cannot be prime. But N has a prime factor, say q, and clearly q cannot be among $2, 3, \ldots, p$, for no such $q \leq p$ can divide the right side of equation (2). So $q > p$, a contradiction. We conclude that there is no largest prime number.

With the Fundamental Theorem at our disposal we can say a few more things about divisibility.

More Divisibility Facts

5. If p is a prime and $p|n$ then p is one of the primes in the prime power factorization of n.

6. Suppose that p is a prime and $p|(ab)$. Then $p|a$ or $p|b$ (or both).

7. Suppose p and q are distinct primes and $p|a$ and $q|a$. Then $(pq)|a$.

8. $a|b$ if and only if every prime p occurring in the prime factorization of a also occurs in the factorization of b, and the power of p in a is $\leq$ the power in b.

Problems:

- Prove the above assertions.

- Why can no integer of the form $n^2 + 4n + 3$ ever be prime for $n > 1$?

- One of the central questions in number theory is "how are the primes distributed among the integers?" Prove that if k is a positive integer then $k! + 2, k! + 3, k! + 4, \ldots k! + k$ are all composite. Hence there are arbitrarily large gaps in the integers in which no primes appear.

1.3 Greatest Common Divisors

If a and b are integers and d is another integer which divides both a and b, then d is called a *common divisor* of a and b. Among all the common divisors of a and b (of which there are finitely many) the largest is called the *greatest common divisor*, or "gcd" for short. The notation for the gcd of a and b is (a, b). It's easy to see that if the prime power factorizations of a and b are

$$\begin{aligned} a &= p_1^{k_1} p_2^{k_2} \cdots p_n^{k_n}, \\ b &= p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n}, \end{aligned}$$

where all the k's and m's are non-negative then

$$(a, b) = p_1^{\min(k_1, m_1)} p_2^{\min(k_2, m_2)} \cdots p_n^{\min(k_n, m_n)}. \tag{3}$$

For example, $60 = 2^2 \cdot 3^1 \cdot 5^1$ and $40 = 2^3 \cdot 3^0 \cdot 5^1$. The gcd of 40 and 60 must be $2^2 \cdot 3^0 \cdot 5^1 = 20$. For the record, the smallest number that is divisible by both a and b is called the *least common multiple*, or "lcm " for short. The notation for it is $[a, b]$. It can be found by changing the "mins" in equation (3) to "max". In the example with 40 and 60 you find that the $[40, 60] = 2^3 \cdot 3^1 \cdot 5^1 = 120$.

Here's a piece of terminology that we'll use A LOT: If $(a, b) = 1$ then a and b are said to be *relatively prime* to one another. They have no common divisors except 1.

Problems:

- Find $(264, 270)$ by factoring each piece and using equation (3).
- True or False: The gcd of a and b is divisible by all of the other common divisors of a and b.
- Suppose $a|b$. What is (a, b)?
- If p is prime and n is an integer, what is (p, n)? What is $[p, n]$?
- Suppose that $a|(bc)$ and that $(a, b) = 1$. What conclusion can you draw? Is the conclusion true if we drop the condition $(a, b) = 1$?
- Suppose $(a, b) = d$. What is $(a/d, b/d)$?
- Simplify $(a, b)[a, b]$.

1.4 Euclid's Algorithm for the GCD

Contemplate for a moment how you would compute the gcd of the numbers

$$\begin{aligned} a &= 3879982794798732472474773847888374209001112283331183701987340987320498666098723 1 \\ b &= 1978129749719374901878810389566630923870198720980019309846563100987109763010197 3 \end{aligned}$$

using equation (3). You have to start by factoring each number. Since both have about 70 digits, it's going to take awhile. A long while. Larger numbers have been factored by big computers, but your laptop probably isn't going to make a dent in these. What if I told you that there is a way to find the gcd of a and b that's much faster—so fast that you could reasonably do it BY HAND in an afternoon, if you were inclined to spend a couple hours doing arithmetic with 70 digit numbers? Of course, your laptop will do it instantly if you use this fast algorithm.

The algorithm is called *Euclid's Algorithm*. It's easy to understand, especially if you work through it a few times. Here's Euclid's algorithm, via an example. Let's find $(565, 165)$. By the division property of the integers (or simple arithmetic!), we can write

$$565 = 3 \cdot 165 + 70.$$

Now it's easy to see that if any number d divides both 565 and 165, then $d|70$. Similarly, anything that divides 70 and 165 automatically divides 565. In other words, 70 and 165 have exactly the same common divisors as 565 and 165, and in particular it must be the case that $(565, 165) = (165, 70)$. This observation reduces the problem of finding $(565, 165)$ to the slightly easier problem of finding $(70, 165)$. Repeat the process: write

$$165 = 2 \cdot 70 + 25.$$

From the reasoning above we know that $(165, 70) = (165 - 2 \cdot 70, 70) = (25, 70)$. Easier still. Continue:

$$70 = 2 \cdot 25 + 20$$

and so we know $(70, 25) = (70 - 2 \cdot 25, 25) = (20, 25)$. Repeat again:

$$25 = 1 \cdot 20 + 5$$

so that $(25, 20) = (25 - 1 \cdot 20, 20) = (5, 20)$. If you do this one more time you find that

$$20 = 4 \cdot 5 + 0$$

i.e., $5|20$, so $(5, 20) = 5$. In the end we've determined that

$$(565, 165) = (165, 70) = (70, 25) = (25, 20) = (20, 5) = 5.$$

Here are the computation arranged in a very systematic way:

$$\begin{aligned} 565 &= 3 \cdot 165 + 70, \\ 165 &= 2 \cdot 70 + 25, \\ 70 &= 2 \cdot 25 + 20, \\ 25 &= 1 \cdot 20 + 5, \\ 20 &= 4 \cdot 5 + 0 \end{aligned}$$

If you want to be algorithmic about it you could say that to find (a, b) (with a and b positive)

1. Set n equal to the larger of a or b, m equal to the smaller.
2. Write $n = qm + r$ with $0 \leq r < m$.
3. If $r = 0$ terminate. Then $(a, b) = m$, so return m.
4. Set $n = m$, $m = r$ and return to step 2.

Fun Problem

- Use Euclid's Algorithm to find the gcd of 300 and 125.

Worst Case Performance of Euclid's Algorithm

What we want here is some kind of bound on how many steps the algorithm will take in terms of the input a and b. This is surprisingly easy to determine. Suppose that the algorithm takes n steps for some input a and

b, with $a > b$. Start by defining $a_{n+1} = a$ and $a_n = b$, and then write the algorithm in the form

$$\begin{aligned} a_{n+1} &= q_n a_n + a_{n-1}, \\ a_n &= q_{n-1} a_{n-1} + a_{n-2}, \\ a_{n-1} &= q_{n-2} a_{n-2} + a_{n-3}, \\ &\vdots \\ a_4 &= q_3 a_3 + a_2, \\ a_3 &= q_2 a_2 + a_1, \\ a_2 &= q_1 a_1 + 0. \end{aligned}$$

At each stage of course a_k gets smaller, so that $0 \leq a_k < a_{k+1}$. The most important point is that everything in sight above is a positive integer. In particular, $a_1 \geq 1$. Also, since $a_1 < a_2$ we know that $a_2 \geq 2$. Now look at $a_3 = q_2 a_2 + a_1$. Since q_2 is positive, $q_2 \geq 1$, so $a_3 \geq a_2 + a_1$, i.e., $a_3 \geq 3$. Apply the same reasoning to $a_4 = q_3 a_3 + a_2$. We get $a_4 \geq a_3 + a_2$ or $a_4 \geq 5$. Similarly $a_5 \geq a_4 + a_3 = 8$, and $a_6 \geq a_5 + a_4 = 13$, and so on.

You may recognize this pattern: It is apparent that $a_k \geq f_{k+1}$, where f_k denotes the kth Fibonacci number. In particular, we find that

If Euclid's algorithm takes n steps to compute (a, b) with $a > b$ then $a \geq f_{n+2}$.

The Fibonacci numbers are the "worst case" for Euclid's algorithm. We can be assured that if $a < f_{n+2}$ then Euclid's algorithm will take AT MOST n steps to compute the gcd (a, b). In fact, since

$$f_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

we can write $a < f_{n+2}$ very explicitly as

$$a < \frac{1}{\sqrt{5}} (\phi_1^{n+2} - \phi_2^{n+2})$$

where $\phi_1 = \frac{1+\sqrt{5}}{2}$ and $\phi_2 = \frac{1-\sqrt{5}}{2}$. But since $|\phi_2| < 1$, ϕ_2^n gets small very rapidly as n gets large, and so we can really write (at least for sufficiently large n)

$$a < \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{n+2}$$

or equivalently, for any given a Euclid's Algorithm will need AT MOST

$$n = \frac{\ln(a) + \ln(\sqrt{5})}{\ln(\frac{1+\sqrt{5}}{2})} - 2 \approx 2.08 \ln(a) - 0.33 \approx 4.8 \log_{10}(a) - 0.33$$

steps to compute the GCD of a and any smaller number.

Problem: It's easy to see that the number of steps in Euclid's Algorithm is really determined by the *smaller* of the two input numbers, not the larger, for after the first iteration the numbers a and b will both be equal to or less than the smaller of the two original inputs. Use this observation to show that the number of steps needed by Euclid is less than 5 times the number of base ten digits in the smaller of the two inputs.

Here's a fact that will be quite useful in the future:

Theorem: If $(a, b) = h$ then there are integers s and t such that

$$h = sa + tb. \tag{4}$$

For example, $(132, 216) = 12$, and $12 = 5 \cdot 132 - 3 \cdot 216$. An easy way to see that the Theorem always works is to prove it via Euclid's algorithm. Here's how Euclid's algorithm works on $(132, 216)$:

$$\begin{aligned} 216 &= 1 \cdot 132 + 84, \\ 132 &= 1 \cdot 84 + 48, \\ 84 &= 1 \cdot 48 + 36, \\ 48 &= 1 \cdot 36 + 12, \\ 36 &= 3 \cdot 12 + 0 \end{aligned}$$

From the above computations (starting with the second to last line) we get

$$\begin{aligned} 12 &= 48 - 36 \\ &= 48 - (84 - 48) = 2 \cdot 48 - 84 \\ &= 2 \cdot (132 - 84) - 84 = 2 \cdot 132 - 3 \cdot 84 \\ &= 2 \cdot 132 - 3 \cdot (216 - 132) = 5 \cdot 132 - 3 \cdot 216. \end{aligned}$$

At each stage we replace the smallest number on the right with an integer combination of the next two largest numbers from the previous step of Euclid's Algorithm. So $(132, 216) = 5 \cdot 132 - 3 \cdot 216$.

It's not hard to check that the above example isn't unique—it works for the gcd of any two numbers. Try another example to convince yourself. Some other useful facts that follow from equation (4) are:

1. For integers a and b let M be the set $M = \{as + bt;\ s, t \text{ integers}\}$. Then $h = (a, b)$ is the smallest positive number in M.

2. If a and b are relatively prime ($(a, b) = 1$) then there are integers s and t such that $as + bt = 1$.

3. For fixed integers a, b, and d, the equation $as + bt = d$ is solvable in s and t iff $(a, b)|d$.

To prove (1), notice that we already know that there exist integers s and t such that $as + bt = (a, b)$; the part we have to prove is that (a, b) is the smallest element in M. Suppose it wasn't, so that we can find s and t such that

$$h = as + bt$$

with $0 < h < (a, b)$. But (a, b) divides both a and b, and hence divides the right side of the above equation, so $(a, b)|h$. This is impossible if $h < (a, b)$ so our original assumption is false: (a, b) is the smallest positive number in M. Assertion (2) is just the special case of Theorem above when a and b are relatively prime. Part (3) is homework!

Fun Problem

- Suppose $a = 4044$ and $b = 792$. Find integers s and t so that $(a, b) = as + bt$.

2 Congruences and Fermat's Theorem

2.1 Congruences

You already know the basics of linear congruences. If it's two o'clock now, then in three hours it will be five o'clock. In 14 hours it will be four o'clock.

In 323 hours what time will it be? Well, $2 + 323 = 325$, but we should toss out all possible multiples of 12. In this case $325 = 12 \cdot 27 + 1$, so it will be one o'clock. If we were on a 24 hour clock we'd toss out all possible multiples of 24.

The idea behind congruences is exactly the same as the clock arithmetic above; we just work with a clock that has m hours instead of 12, where m can be any positive integer we want. Two integers a and b are said to be *congruent* modulo m, written

$$a \equiv b \pmod{m}$$

if $m|(a-b)$. Another way to look at it is that if we write $a = q_1 m + r_1$ and $b = q_2 m + r_2$ with $0 \leq r_1 < m$ and $0 \leq r_2 < m$ (this always be done) then $a \equiv b \pmod{m}$ means $r_1 = r_2$.

When someone asks the value of $a \pmod{m}$ it's convention to report a number r where $r \equiv a \pmod{m}$ and $0 \leq r < m$. For example, 93 (mod 15) is 3 (mod 15). One thing worth noting: $a \pmod{a}$ would be simplified to $0 \pmod{a}$. It's like reporting that it's zero o'clock, instead of twelve o'clock.

Properties of Congruences

1. If $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$ then $a \equiv c \pmod{m}$.
2. $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then $a + c \equiv b + d \pmod{m}$.
3. $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then $ac \equiv bd \pmod{m}$.
4. If $(c, m) = 1$ and $ac \equiv bc \pmod{m}$ then $a \equiv b \pmod{m}$.

To prove (1), note that $a \equiv b \pmod{m}$ means $m|(a-b)$, i.e., $(a-b)/m$ is an integer. By hypothesis so is $(b-c)/m$. But then $(a-b)/m + (b-c)/m = (a-c)/m$ is an integer, i.e., $m|(a-c)$ or $a \equiv c \pmod{m}$. The proofs of (2) and (3) are just as easy. Only the proof of (4) involves any subtlety at all. We are given that $ac \equiv bc \pmod{m}$, meaning that $ac - bc$ is divisible by m, so

$$(ac - bc) = km \tag{5}$$

for some integer k. We want to show $a - b$ is a multiple of m. Equation (5) makes it clear that $c|(km)$. But if $(c, m) = 1$ this means that $c|k$. Then Then

$$a - b = \frac{k}{c} m.$$

and k/c is an integer. This is the very definition of $a \equiv b \pmod{m}$.

The last property isn't true if $(c, m) \neq 1$. For example, $14 \equiv 20 \pmod{6}$, but dividing both sides by 2 produces $7 \equiv 10 \pmod{6}$, which is false.

Fun Problems

- Show that if n is any odd integer then $n^2 - 1$ is divisible by 8.
- Suppose that the decimal representation for a is $a_n a_{n-1} \cdots a_1 a_0$, i.e., $a = a_n 10^n + \cdots + a_1(10) + a_0$. Show that

  $$a \equiv a_0 + a_1 + \cdots + a_n \pmod{3}$$

  and so deduce the well known divisibility test for 3.

Here's a couple more useful properties of congruences:

5 If $x^2 \equiv y^2 \pmod{p}$ where p is prime, then either $x \equiv y \pmod{p}$ or $x \equiv -y \pmod{p}$.

6 For given integers a and m the congruential equation

$$ax \equiv 1 \pmod{m}$$

is solvable iff $(a, m) = 1$.

The proofs are pretty easy. For (5), from the congruence we know that $p|(x^2 - y^2)$. But $x^2 - y^2 = (x - y)(x + y)$, so $p|(x - y)(x + y)$. But if a prime p divides a product then it divides at least one factor, so either $p|(x - y)$ or $p|(x + y)$ (or both). In the first case $x \equiv y \pmod{p}$ and in the second $x \equiv -y \pmod{p}$.

To prove (6), once again write out what $ax \equiv 1 \pmod{m}$ means: $ax - 1$ is divisible by m, i.e., is a multiple of m, so

$$ax - 1 = km$$

for some integer k. But this is the same as $ax - km = 1$, which we know is solvable for it is equivalent to saying that $(a, m) = 1$. Euclid's algorithm would be an efficient means for finding a solution.

2.2 Fermat's Little Theorem; Primality Tests

Here it is.

Fermat's Theorem: *If p is a prime and a is any integer not divisible by p then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof:(From Giblin, *Primes and Programming*, Cambridge University Press) Since p doesn't divide a, $(a, p) = 1$. Look at the set of numbers

$$a, 2a, 3a, \ldots, (p-1)a.$$

None of these is a multiple of p ($p|(ja)$ implies $p|j$ or $p|a$, both impossible) so none is congruent to 0 mod p. Also, no two of these numbers are congruent to each other mod p, for if $ja \equiv ka \pmod{p}$ then $ja - ka = (j-k)a$ would be divisible by p, implying $p|a$ or $p|(j-k)$, both impossible. Thus these numbers are, mod p, just the numbers 1,2, $p-1$ in some rearranged order. If we multiply all of them together we must have

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p},$$

or, with a little simplification

$$a^{p-1} 1 \cdot 2 \cdot 3 \cdots (p-1) \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p}.$$

But since p is relatively prime to each of $1,2,\ldots,(p-1)$ we can divide these factors from both sides to conclude that

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

■

Of course, Fermat's little theorem may not (and usually isn't) true if p isn't prime. For example, if $n = 16$ and $a = 6$ then $a^{n-1} = 6^{15} \equiv 0 \pmod{16}$, not 1.

A Simple Test for Primality; Repeated Squaring

Here's an easy way to test whether a number n might be prime (although we'll shortly make some refinements). Pick a random integer a (and you may as well choose $1 < a < n$.) Compute $a^{n-1} \pmod{n}$. If it's not 1 then n

CANNOT be prime. If the result is 1, then n might be prime.

Example 1: Suppose that $n = 22$; we want to determine whether n could be prime, and yet we're too dumb to tell at a glance (but smart enough to know Fermat's little theorem!?). Take $a = 3$ and compute $a^{21} \pmod{22}$. There are (at least) 3 ways to do this. A very, very inefficient method, a very inefficient method, and an efficient method.

1. The very very inefficient method is to raise 3^{21} by brute force, then reduce mod 22.

2. Slightly better is to do the following: $3 \equiv 3 \pmod{22}$, so $3^2 \equiv 9 \pmod{22}$, so $3^3 \equiv 27 \equiv 5 \pmod{22}$, so $3^4 \equiv 3 \cdot 5 \equiv 15 \pmod{22}$, so $3^5 \equiv 3 \cdot 15 \equiv 1 \pmod{22}$, and so on. In short, to compute $3^{k+1} \pmod{22}$, take the previously computed value of $3^k \pmod{22}$, multiply by 3, then immediately reduce mod 22. You never have to work directly with huge numbers like 3^{21}.

3. The efficient way is to notice that 21 can be decomposed into powers of two, as $21 = 16 + 4 + 1$. Then compute that $3 \equiv 3 \pmod{22}$, $3^2 \equiv 9 \pmod{22}$, and $3^4 \equiv (3^2)^2 \equiv 81 \equiv 15 \pmod{22}$. Also $3^8 \equiv (3^4)^2 \equiv 15^2 \equiv 5 \pmod{22}$, and $3^{16} \equiv (3^8)^2 \equiv 25 \equiv 3 \pmod{22}$. By squaring repeatedly we can easily compute 3^{2^k} for any k. In the end we have
$$3^{21} = 3^{16}3^4 3^1 \equiv 3 \cdot 15 \cdot 3 \equiv 135 \equiv 3 \pmod{22}.$$
By repeated squaring I've reduced computing $3^{21} \pmod{22}$ to the problem of computing $3^1, 3^2, 3^4, 3^8$, and 3^{16} mod 22. A little analysis shows that computing a^n this way takes $\log(n)$ multiplications modulo n (as opposed to order n for the second method).

Since $3^{21} \neq 1 \pmod{22}$, 22 is definitely NOT prime. If, on the other hand it DID turn out that $3^{21} \equiv 1 \pmod{22}$, we could then try computing $a^{21} \pmod{22}$ for another choice of a, and repeat this for different a as many times as we like. If it ever turns out that $a^{21} \neq 1 \pmod{22}$, we conclude 22 isn't prime.

What if an integer n passes the test for every a we try—does that prove that n is prime? Unfortunately there are integers n which are NOT prime and yet will pass this test for any choice of a. If a composite integer n satisfies $a^{n-1} \equiv 1 \pmod{n}$ for some a relatively prime to n then we say that n is

a *pseudoprime* to base a. Even more unfortunately, there are nasty integers which are pseudoprimes in EVERY base a (with $(a, n) = 1$, of course). They are called *Carmichael* numbers. For example, 561 is a Carmichael number. It is composite ($561 = 3 \cdot 11 \cdot 17$) and yet for every integer a with $(a, 561) = 1$ we find that $a^{560} \equiv 1 \pmod{560}$.

A Refinement: Miller's Test for Primes

Let's use Fermat's Theorem to test whether $n = 341$ might be prime. We compute $2^{340} \pmod{341}$. The result is 1, so 341 is either prime or a pseudoprime to base 2. But now recall a result we proved earlier: if $x^2 \equiv y^2 \pmod{p}$ where p is prime, then $x \equiv y \pmod{p}$ OR $x \equiv -y \pmod{p}$. In the special case $y = 1$ we have $x^2 \equiv 1 \pmod{p}$ implies $x \equiv 1 \pmod{p}$ or $x \equiv -1 \pmod{p}$. So IF 341 is prime then $2^{340} = (2^{170})^2 \equiv 1 \pmod{341}$ implies

$$2^{170} \equiv \pm 1 \pmod{341}.$$

In fact we find that $2^{170} \equiv 1 \pmod{341}$, which is consistent with 341 being prime. But $2^{170} = (2^{85})^2$, so if 341 is prime then $2^{85} \equiv \pm 1 \pmod{341}$. Compute it; you find $2^{85} \equiv 32 \pmod{341}$. So 341 cannot be prime!

This is the basis of Miller's Test: you want to test whether an integer n is prime. Pick some number a and compute $a^{n-1} \pmod{n}$. If it's not 1, n is not prime. If the result is 1, AND if $n - 1$ is divisible by 2 then compute $a^{(n-1)/2} \pmod{n}$. If n is prime the result must be ± 1. If it's not, n is composite. If it's -1 then you can't go any farther with this base a, and so n is either prime or, if n is composite, n is said to be a *strong pseudoprime* to base a. If the result is 1, but $(n-1)/2$ is NOT again divisible by 2 then n, if not actually prime, is also said to be a strong pseudoprime to base a. If $a^{(n-1)/2} \equiv 1 \pmod{n}$ AND $(n-1)/2$ is again divisible by 2 then compute $a^{(n-1)/4} \pmod{n}$. It must be ± 1 and we do it all again. We continue until we obtain a result which is NOT ± 1 (in which case n is composite) or until we obtain -1 or find that the exponent is no longer divisible by 2, in which case n must be prime or is a strong pseudoprime to base a.

Here's some concrete examples: Let's first look at 561 in base 2. Compute

$$\begin{aligned} 2^{560} &\equiv 1 \pmod{561}, \\ 2^{280} &\equiv 1 \pmod{561}, \\ 2^{140} &\equiv 67 \pmod{561}. \end{aligned}$$

So 561 must be composite. On the other hand, look at $n = 2047$ to base 2:

$$\begin{aligned} 2^{2046} &\equiv 1 \pmod{2047}, \\ 2^{1023} &\equiv 1 \pmod{2047}. \end{aligned}$$

We can't go any farther, since 1023 isn't divisible by 2, and so 2047 is either prime or a strong pseudoprime to base 2. However, in base 3 the number 2047 reveals its compositeness immediately, for $3^{2046} \equiv 1013 \pmod{2047}$.

This illustrates one way to test whether an integer n is prime. Pick some base a with $1 < a \leq n-1$ and apply Miller's Test. If n fails, n is composite. If n passes then try another base, and another, etc. If n ever fails, n is composite. If n passes every test then it's highly likely that n is prime. In fact it can be proved that

Theorem: *If n is odd and composite then n passes Miller's test for AT MOST $(n-1)/4$ bases a with $1 \leq a \leq n-1$.*

For a proof, see K.H. Rosen, *Elementary Number Theory and Its Applications*, Addison-Wesley, 1988, or Koblitz, *A Course in Number Theory and Cryptography*, Springer Verlag, 1985.

If we pick a "random" base and n is composite then n has only a $1/4$ chance of passing Miller's Test. If we pick k different random bases independently then n has only a $1/4^k$ chance of masquerading as a prime number. Of course, a true prime will never fail the test.

3 Euler's Function

Suppose n is a positive integer. We'll define the function $\phi(n)$ by

$\phi(n)$ equals the number of integers k with $1 \leq k \leq n$ and $(k, n) = 1$.

The function ϕ is called *Euler's function* and it has many interesting properties.

Example 1: If p is prime then $\phi(p) = p-1$, since every integer $1, 2, \ldots, p-1$ is relatively prime to p.

Example 2: Let's compute $\phi(p^n)$ where p is prime and n is some positive integer. An integer k is relatively prime to p^n iff k is not a multiple of p. Which numbers in the range $1 \leq k \leq p^n$ ARE multiples of p? These are the numbers $p, 2p, 3p, \ldots (p^{n-1})p$ Thus there are exactly p^{n-1} numbers which are NOT relatively prime to p^n, leaving $p^n - p^{n-1}$ numbers which are relatively prime to p. For example, $\phi(7^3) = 7^3 - 7^2 = 294$.

Example 3: Let's compute $\phi(pq)$ where p and q are DISTINCT primes. List the integers from 1 to pq:

$$1, 2, 3, 4, \ldots, pq.$$

There are of course pq integers. Strike out all of the integers which are multiples of p; there are q such integers, $p, 2p, 3p, \ldots, qp$. Similarly strike out all the multiples of q: $q, 2q, 3q, \ldots, (p-1)q$, but don't strike out pq again; there are $p-1$ numbers eliminated. Notice that we didn't strike out anything twice—there is no number $a < pq$ which is both a multiple of p and a multiple of q if $p \neq q$. How many numbers are left? Exactly $pq - p - (q-1) = pq - p - q + 1$. This can be simplified slightly by noticing that $pq - p - q + 1 = (p-1)(q-1)$. In fact, since $\phi(p) = p - 1$ and $\phi(q) = q - 1$, we've proved that for distinct primes p and q it's true that $\phi(pq) = \phi(p)\phi(q)$.

It's not coincidence that $\phi(pq) = \phi(p)\phi(q)$. This is merely a special case of

Theorem: If $(m, n) = 1$ then $\phi(mn) = \phi(m)\phi(n)$.

Proof: Let $r_1, r_2, \ldots, r_{\phi(m)}$ denote the $\phi(m)$ numbers in the range $1, 2, \ldots, m$ which are relatively prime to m, and similarly $s_1, s_2, \ldots, s_{\phi(n)}$ the number between 1 and n which are relatively prime to n. Consider a number x with $1 \leq x \leq mn$ with $(x, mn) = 1$. We are going to establish a one-to-one correspondence between each such x and pairs of the form (r_i, s_j). Since the number of such pairs is $\phi(m)\phi(n)$, this will prove the theorem.

Let x be given, with $(x, mn) = 1$. Clearly then $(x, m) = 1$ and $(x, n) = 1$. The first claim is that

$$x \equiv r_i \pmod{m}, \tag{6}$$
$$x \equiv s_j \pmod{n}. \tag{7}$$

for one of the r_i and s_j. If this wasn't true—if say $(r_i, m) = d > 1$—then we have $d|(x - r_i)$ which would imply that $d|x$ so that $(x, m) \geq d > 1$, a

contradiction. A similar argument shows that the congruence (7) holds for some s_j. Thus each number x with $1 \leq x \leq mn$ with $(x, mn) = 1$ can be associated with a pair (r_i, s_j).

The converse is also true: given a pair (r_i, s_j) we can associate this pair with a unique number x between 1 and mn with $(x, mn) = 1$ by solving the simultaneous congruences (6)-(7) for x. Why are these solvable? Well, since $(m, n) = 1$ there are some integers x_1 and y_1 such that $nx_1 + my_1 = 1$. But this implies that

$$nx_1 \equiv 1 \pmod{m}.$$

Similar reasoning shows that there is some integer x_2 such that $mx_2 \equiv 1 \pmod{n}$. Now define $\tilde{x} = r_i n x_1 + s_j m x_2$. It's easy to check that $\tilde{x}$ satisfies the congruences (6)-(7). Of course $\tilde{x}$ may not lie in the range 1 to mn, but by taking $x = \tilde{x} \pmod{n}$ we find that x still satisfies congruences and $1 \leq x \leq mn$. Also, the number x which satisfies the congruences (6)-(7) is unique, for if we could find two solutions x and y then we would have $x - y$ divisible by both m and n, and since $(m, n) = 1$ we can conclude that $mn|(x-y)$, impossible if both x and y lie in the range 1 to mn, unless $x = y$. Finally, we can check that $(x, mn) = 1$; if not then we'd have, for example, $(x, m) > 1$, which would make equation (6) impossible. Similar remarks show that $(x, n) = 1$, so $(x, mn) = 1$.

So in the end we've proved that there is a one-to-one pairing between each x with $1 \leq x \leq mn$ with $(x, mn) = 1$ and each pair (r_i, s_j). But there are exactly $\phi(m)\phi(n)$ such pairs (r_i, s_j), meaning there are exactly that same number of x. So $\phi(mn) = \phi(m)\phi(n)$.■

The theorem easily extends to the case in which arbitrarily many pairwise prime integers are multiplied together, i.e., $\phi(mnr) = \phi(m)\phi(n)\phi(r)$ if $(m, n) = (m, r) = (n, r) = 1$. From this we get a nice formula for $\phi(n)$ in terms of the prime power factorization of n. If

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

then

$$\phi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1 - 1})(p_2^{\alpha_2} - p_2^{\alpha_2 - 1}) \cdots (p_k^{\alpha_k} - p_k^{\alpha_k - 1}). \tag{8}$$

This can also be written as

$$\phi(n) = n \left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

For example, if $n = 300$ then $n = 2^2 \cdot 3 \cdot 5^2$ and so $\phi(n) = (2^2-2)(2)(5^2-5) = 80$.

Euler's Theorem

Recall Fermat's Theorem: if p is prime and $(a,p) = 1$ then $a^{p-1} \equiv 1 \pmod{p}$. Of course this isn't usually true if p is not prime. What is true is Euler's Theorem:

For any integers a and n with $(a,n) = 1$ we have $a^{\phi(n)} \equiv 1 \pmod{n}$.

Notice that this really is a generalization of Fermat's Theorem, for if we take $n = p$ a prime then $\phi(n) = p - 1$.

Proof of Euler's Theorem: This is virtually identical to the proof of Fermat's Theorem. Let $r_1, r_2, \ldots, r_{\phi(n)}$ be the $\phi(n)$ positive integers less than n which are relatively prime to n. Consider the integers

$$ar_1, ar_2, \ldots, ar_{\phi(n)} \pmod{n}.$$

I claim that these are simply the numbers $r_1, r_2, \ldots, r_{\phi(n)}$ in some rearranged order. To see this first notice that $(ar_i, n) = 1$, since both $(a,n) = 1$ and $(r_i, n) = 1$. So each of the numbers $ar_i \pmod{n}$ is one of the r_j. Also, it is impossible to have $ar_i = ar_j$ unless $i = j$, for then we would have $a(r_i - r_j) \equiv 0 \pmod{n}$; but since $(a,n) = 1$ this forces $r_i \equiv r_j \pmod{n}$, impossible since $1 \leq r_i, r_j < n$ (unless $i = j$). Thus each integer ar_i is matched with a unique r_j, and the numbers $ar_1, ar_2, \ldots, ar_{\phi(n)} \pmod{n}$ are just $r_1, \ldots, r_{\phi(n)}$ in some rearranged order, as claimed. This makes it clear that

$$(ar_1)(ar_2)\cdots(ar_{\phi(n)}) \equiv r_1 r_2 \cdots r_{\phi(n)} \pmod{n}$$

or that

$$a^{\phi(n)} r_1 r_2 \cdots r_{\phi(n)} \equiv r_1 r_2 \cdots r_{\phi(n)} \pmod{n}.$$

But since each of the r_i are relatively prime to n, we can divide both sides of this congruence by each of the r_i and the result is exactly Euler's Theorem. ■

4 Applications to Cryptography

Below are detailed two cryptosystems which use simple ideas from number theory. The first is an *exponentiation cipher* and the second is the RSA public key algorithm.

Whatever cryptosystem is used, one must first translate the message into numbers. You can probably imaging a million ways to do this, but for these simple examples we'll pick something very easy: we'll pretend that the messages are computer files and we'll translate the message byte-by-byte using the ascii values of the characters. So for example, "a" will become 97, "b" will become 98, etc. Of course this would be terrible in practice, but it will serve for these examples. In reality one would encode larger blocks of characters into integers, and it's easy to imagine incorporating compression, error correction, or other pre- or post-processing into the scheme. But our messages will all consist of strings of integers from 0 to 255.

4.1 Exponentiation Ciphers

Here is a cryptosystem which is closely related to the RSA algorithm, but a bit simpler. It allows persons (traditionally "Alice" and "Bob") to send secret messages to each other. They choose a large prime number p and an "enciphering key" e with the property that $(e, p-1) = 1$. Let X be the *plaintext* message, that is, the unencrypted message that Bob wants to send to Alice. X will be a sequence of numbers (from 0 to 255 in our simple encoding scheme). Bob encrypts the message X by applying the enciphering operator

$$E(X) = X^e \pmod{p}$$

to X, byte-by-byte, to produce a new string of integers $E(X)$. Alice, upon receiving the *ciphertext* $E(X)$, decrypts it by using the deciphering operator

$$D(Y) = Y^d \pmod{p}$$

where d, the *deciphering exponent*, is chosen so that $de \equiv 1 \pmod{p-1}$. Such a d can always be found, since $(e, p-1) = 1$; all we have to do is solve $ed + k(p-1) = 1$ with Euclid's Algorithm. It's easy to check that

$$\begin{aligned} D(E(X)) &\equiv X^{ed} \pmod{p} \\ &\equiv X \cdot (X^{p-1})^k \pmod{p} \\ &\equiv X \end{aligned}$$

where we've used $de = 1 + k(p-1)$ for some k and $X^{p-1} \equiv 1 \pmod{p}$ by Fermat's Theorem. Thus D is an inverse for E. Of course applying Fermat's Theorem requires $(X, p) = 1$, so in our case we'd choose $p > 256$.

In the above scheme Bob and Alice must agree upon the prime p and enciphering key e. It's clear that they cannot reveal p and e to anyone else, for anyone who knows p and e can easily solve $de \equiv 1 \pmod{p}$ and so find the deciphering key. Thus this is an example of a *private key* encryption scheme.

However, they could conceivably reveal the value of p alone without compromising the security of the system. Consider a case in which someone discovers the value of p AND knows a piece of plaintext X and the corresponding ciphertext $Y = X^e \pmod{p}$. That person seeks to discover the enciphering exponent e, and so compute d. What this person must do is find an e such that

$$X^e \equiv Y \pmod{p}. \tag{9}$$

Consider the equation $X^e = Y$ where X, Y, and e are real numbers. In this case we easily solve for e as $e = \log_X(Y)$. There isn't any such simple procedure modulo p (more generally, in a finite field). The problem of solving for e in equation (9) with X and Y known is called the *discrete logarithm* problem. It is comparable in difficulty to factoring. The security of an exponentiation cipher rests on the difficulty of solving this problem in general. It's not hard to see that the discrete log problem is easy if p is small—brute force would work—but if p is large the problem becomes intractable. HOWEVER, even if p is large there are attacks that can solve this in special cases, e.g., if $p-1$ has all small prime factors.

The Massey-Omura Cryptosystem

To use a simple exponentiation cipher as described above, Alice and Bob have to agree on a prime p and an enciphering key e. Although the prime can be made public, e cannot, so Bob and Alice will have to agree upon e in some secure setting.

There is a way to use this system which does not require Bob and Alice to exchange or develop e in a secure setting. Here's how it works. Let the prime p be agreed upon, publicly. Alice chooses and enciphering key e_A and computes the corresponding deciphering key d_A by solving $e_A d_A \equiv 1 \pmod{p-1}$. She keeps both private, known only to herself. Bob chooses an enciphering key e_B and computes the corresponding d_B.

If Alice wants to send Bob a message X, she computes $X^{e_A} \pmod p$ and transmits this to Bob. No one who intercepts the message can read it—in fact, Bob himself can't even read it! He takes the received message and exponentiates it again, with e_B, to form $X^{e_A e_B} \pmod p$, which he transmits back to Alice. Alice hits the (doubly) encrypted message with d_A, to form $X^{e_A e_B d_A} \equiv X^{e_B} \pmod p$, and transmits this back to Bob. He then applies d_B to it, computing $X^{e_B d_B} \equiv X \pmod p$, and so recovers the message Alice sent him.

One necessity in this cryptosystem is a good *signature* scheme, that is, a good way to identify with certainty the sender of a message. We'll discuss how to do this later, but suppose the system above were used without any signatures, and that intruder C intercepts the encrypted message X^{e_A} that Alice sent to Bob. The intruder (who knows p—it's public) could then apply his own enciphering exponent e_C and return $X^{e_A e_C}$ to Alice, who, not knowing that C isn't Bob, exponentiates again with d_A and returns X^{e_C} to intruder C. C of course can compute d_C and so $X^{e_C d_C} = X \pmod p$, and so decrypt the message intended for Bob.

4.2 RSA Public Key Encryption

Public key encryption provides a means for anyone to send you a secret message, yet no one (not even the person who encrypts the message) can decrypt it. The basis of the RSA public key scheme is Euler's Theorem. Here's how it works.

As with any type of encryption, we convert the message into a string of numbers; as above I'll simply assume that the message is a computer file and we've converted byte-by-byte, so plain English text would be converted into the ascii equivalent.

In order to provide you the means to send me secret messages, I first find two large prime numbers p and q, and compute $n = pq$. I keep p and q secret, known only to me, but I make n public. I also choose an integer e with the property that $(e, \phi(n)) = 1$. Note that $\phi(n) = (p-1)(q-1)$ is easy to compute if you know p and q, and it's also easy to find an appropriate e. In fact, you could just guess at e and check if $(e, (p-1)(q-1)) = 1$ very quickly with Euclid's Algorithm. If the first guess doesn't work, try again, etc., until it does. The number e is called the *enciphering key*. I make this number public also. In fact, ideally both e and n would be published in some easy to access central directory.

As with the exponentiation cipher I must compute the deciphering key. To do this I solve the congruence

$$ed \equiv 1 \pmod{\phi(n)} \tag{10}$$

for d. I know there's a solution because I chose e so that $(e, \phi(n)) = 1$, and so there are integers s and t such that $se + t\phi(n) = 1$, and I can find them using Euclid's Algorithm. Then $d = s$ will solve equation (10). The number d is the *deciphering key* and must be kept private, known only to me.

Here's how you send me an encoded message. Assume that the message has been converted into integers in the manner above (which would be public knowledge). Let X be a typical integer piece of the message, and let's assume that n is large enough so that $X < n$ in all cases. To encipher the message you compute $E(X)$ where

$$E(X) = X^e \pmod{n}.$$

Anyone can do this since n and e are public knowledge. You transmit $E(X)$ to me; you could even post it in a public place. No one else (not even you!) can easily decrypt the message. But I can. Recall that $ed \equiv 1 \pmod{\phi(n)}$, or equivalently, $ed = 1 + k\phi(n)$ for some integer k. This is the fact that let's

me decrypt the encrypted message. Define the decryption operation $D(S)$ by

$$D(S) = S^d \pmod{n}.$$

I take $E(X)$ that you sent me and compute

$$D(E(X)) = (X^e)^d \equiv X^{ed} \equiv X \cdot (X^{\phi(n)})^k \equiv X \pmod{n}$$

where the last step follows since $X^{\phi(n)} \equiv 1 \pmod{n}$ by Euler's Theorem. D undoes what E did and so I can recover the original message X. Note that this will only work if we have $(X, n) = 1$, or equivalently, $(X, p) = (X, q) = 1$. The odds of this being true are overwhelming if the primes p and q are large. Also note that this scheme requires that we pick n large enough so that we always have $X < n$; otherwise the message must be broken into smaller blocks or integers.

Here's an example. First, I will find two "large" primes. For this example let's take $p = 71$ and $q = 101$. The product is $n = 7171$, and $\phi(n) = 70 \cdot 100 = 7000$. I'll choose an enciphering key $e = 37$; you can easily check $(37, 7000) = 1$. Now I have to compute the deciphering key d, i.e., I have to find a solution to

$$37d + 7000t = 1.$$

You can check (chase Euclid's algorithm backwards) that $d = -3027$ works (with $t = 16$). Actually, we'd like d to be positive. Taking $d = -3027 + 7000 = 3973$ doesn't mess up $ed \equiv 1 \pmod{n}$. You should think about that. So the deciphering key is $d = 3973$.

Let the message be the string "Number theory is fun." Converting this to integers using the ascii equivalence gives

$$\begin{aligned} X &= 78, 117, 109, 98, 101, 114, 32, 116, 104, 101, 111, 114, 121, 32, 105, 115, 32, \\ &\quad 102, 117, 110, 46. \end{aligned}$$

We take each integer, raise it to the 37th power, and take the result mod 7171. This results in the string

$$\begin{aligned} E(X) &= 6128, 227, 3361, 6977, 3030, 4689, 5213, 5930, 6153, 3030, 5363, 4689, \\ &\quad 7115, 5213, 3530, 1117, 5213, 2021, 227, 3094, 5552. \end{aligned}$$

This can be safely transmitted to me; anyone who intercepts it will have to factor 7171 to decrypt it. I now apply the decryption key by raising each number to the 3973rd power and taking the result mod 7171 to find

$$\begin{aligned} D(E(X)) &= 78, 117, 109, 98, 101, 114, 32, 116, 104, 101, 111, 114, 121, 32, 105, \\ &\quad 115, 32, 102, 117, 110, 46 \end{aligned}$$

and so recover the message.

4.3 Digital Signatures

An important part of any transmitted message is the *digital signature*, by which one can verify the sender's identity. Suppose someone sends me an encrypted message that says "Dr. Bryan, this is General MacArthur speaking. Attack at dawn." I'd want to be sure that it really was from MacArthur, since ANYONE could have encrypted that message with my publicly known keys. Here's how the identity verification works.

Like me, General MacArthur would his own public encryption keys, known to the whole world (but not his deciphering key). Let's say that I've chosen primes p_b and q_b to form $n_b = p_b q_b$, I have enciphering key e_b, and deciphering key d_b. General MacArthur has corresponding keys $n_g = p_g q_g$, e_g, and d_g. For a message X, thought of as an integer, let my enciphering operation be by denoted by

$$E_b(X) = X^{e_b} \pmod{n_b}$$

and similarly for $E_g(X)$ for MacArthur's enciphering operation. The deciphering operations are D_b and D_g, respectively. General MacArthur could verify his identity as follows: suppose his plaintext orders are the integer X_1. To the end of his orders he appends a string like $X_2 =$"This really is General MacArthur", but he first applies his *deciphering* key to X_2 i.e., he appends $D_g(X_2)$ to the end of his plaintext orders—the result, $X_1 \cdot D_g(X_2)$, is his plaintext message followed by a string of gibberish. He then enciphers the whole thing by applying E_b and then sends it to me. I receive $E_b(X_1) \cdot E_b(D_g(X_2))$. When I decode the message by applying D_b I recover X_1, his plaintext orders, followed by $D_g(X_2)$, a string of apparent nonsense. But when I hit this string of nonsense with E_g (which I can do—n_g and e_g are public knowledge) I obtain $E_g(D_g(X_2)) = X_2^{e_2 d_2} \equiv X_2 \pmod{n_g}$ and can now read the original string $X_2 =$ "This really is General MacArthur". This verifies that the message came from MacArthur, because whoever appended that string of gibberish to the end of the message must have known MacArthur's deciphering key, a number d_g which solves $d_g e_g \equiv 1 \pmod{\phi(n_g)}$. The only way to do that is to know the value of $\phi(n_g)$, which is more or less equivalent to knowing the factorization of n_g. That is either MacArthur or someone who was able to factor n_g. The latter possibility is small if p_g and q_g are large.

One fine point: this method for sender ID will only work if $n_g < n_b$. The reason is that when MacArthur computes $D_g(X_2)$ the result is an integer between 0 and n_g; if this integer is larger than n_b then it can't be transmitted

in a single block. If we have $n_g > n_b$ then the digital signature can still be done—MacArthur just has to reverse the order of application of D_g and E_b to X_2. Of course, since both n_g and n_b are publicly known, I'd know the order in which he applied them and decrypt accordingly.

4.4 More on Signatures; Hash Functions

The signature scheme as presented above isn't very secure. For example, suppose that I, after receiving the encrypted message from MacArthur, decrypt it to obtain $X_1 \cdot D_g(X_2)$. I remove X_1 and now I have MacArthur's digital signature $D_g(X_2)$. I could append this to any message I want to and so pass myself off as MacArthur! In fact, anyone who receives a message which has been electronically signed by MacArthur could do this.

Alternatively, the encoded message $E_b(X_1) \cdot E_b(D_g(X_2))$ might be intercepted by an enemy in transit. If the enemy could identify that part of the message corresponding to the encrypted plaintext, $E_b(X_1)$, then the enemy could replace that portion of the message with $E_b(Y_1)$ (where Y_1 is the false order "Attack at sunset"), reappend $E_b(D_g(X_2))$, and send the message on its way to me. Note that the enemy can easily compute $E_b(Y_1)$ since this uses public information. We need a way to prevent these kinds of tampering.

What we want is a way "bind" MacArthur's signature and the original message together, in such a way that no one can strip off the signature and reuse it, and no one can tamper with the message itself (without me detecting it). There is such a scheme, based on the idea of a *hash* function.

Hash functions (or "hashing") have applications far beyond cryptography, and there is a huge literature on the subject. Without going into this, we can describe the essential properties that a hash function should have for cryptographic purposes. A good cryptographic hash function f should

1. Take an input integer string X of arbitrary length and produce a fixed length output $f(X)$ (typically a few tens to a few hundred bits).

2. Be easy and fast to compute.

3. It should be totally infeasible to find an input X that hashes to a given value h, i.e., solving $f(X) = h$ for X should be infeasible. Even finding two inputs X and Y that hash to the same value (so $f(X) = f(Y)$) should be infeasible.

4. Although this is in some sense implied by the previous property, any small change in the input X, even one bit, should change the output $f(X)$.

Here's how one uses a hash function to improve our signature scheme above and bind the message to the signature. Assume we have a good hash function f. Let the plaintext message be denoted by X. The signature is no longer an arbitrary string ("This is really MacArthur"), but rather the signature will be $f(X)$. MacArthur forms $X \cdot D_g(f(X))$ and then encrypts with my public key to form $E_b(X) \cdot E_b(D_g(f(X)))$. This he transmits to me. On receiving the message I

1. Apply D_b to recover the plaintext message X and $D_g(f(X))$.

2. Apply E_g to $D_g(f(X))$ to recover $f(X)$.

3. I apply the (publicly known) hash function to the plaintext X I received and compare this value to $f(X)$ as computed in step (2). If they match I can conclude that the message really was sent by MacArthur and was not altered in transit.

It's worth considering a few scenarios in which someone tries to steal MacArthur's signature or alter the message in transit. For example, what if someone tries to change the message meant for me, substituting a new message X' in place of X? I will then receive $E_b(X') \cdot E_b(D_g(f(X)))$. I use D_b to recover the altered message X'. I then apply E_g to the signature to recover $f(X)$. Finally, I apply f to the plaintext X' I received to compute $f(X')$ and compare to the decrypted signature $f(X)$. Almost certainly $f(X) \neq f(X')$ and I conclude that X' is not the message that D_g was applied to—so it can't be the message as MacArthur sent it.

In a similar vein, what if I stripped off the signature $D_g(f(X))$, tried to append it to a message X' of my choosing, and then send it to someone else (thereby attempting to impersonate MacArthur)? Assuming the message was encrypted with this person's public keys, this person uses his or her private key to recover $X' \cdot D_g(f(X))$. Applying E_g to the signature and f to the plaintext X' shows just as before that the message is a forgery, for $f(X) \neq f(X')$. The application of the hash function to the message followed by the application of one's private key inextricably binds the message and signature together.

4.5 Certifying Authorities and Certificates

How can you know anyone else's public key? Presumably such information could be posted in a directory somewhere, or you could get it directly from the person. In any case, such information is necessary for verifying someone else's identity in the signature scheme above, and of course the information must be reliable. But it is possible that some miscreant could deliberately propogate false information about other people's public keys—perhaps alter an insecure database, or intercept and alter in transit a request for someone's public keys. By doing so the culprit could make you believe that Alice's public key is given by the pair n_1, e_1, when in fact the correct key is n_2, e_2. The culprit can then masquerade as Alice (at least to you). How can public keys be distributed in a secure manner, so that no one can alter them and those who use them can be assured they are correct? This is the role of a *certifying authority*.

The certifying authority has its own public keys, say n_a and e_a, and private key d_a. The public keys are made VERY public—probably even built into encryption software, so that everyone knows what they are. The certification authority has the job of verifying that a given user has specified public keys, say n_b and e_b. Here's how: The key authority constructs a "message" X which contains n_b, e_b, the user's name, and any other relevant information about the user. The key authority then computes $C = X^{d_a} \pmod{n_a}$. The number C is a *certificate* which attests to the fact that the given user really has authorization to use the given keys. If the keys n_b, e_b belong to me, I can include the certificate C in any message I send to someone else. That person can hit C with the public key of the certifying authority to recover the information it contains, namely my keys, name, and whatever other information is in C. Only someone with the key authority's deciphering key d_a could have encrypted that information into C, and presumably that was the certifying authority. Thus the security of the whole system ultimately rests on trust in the certifying authority.