RSA Encryption 2

At the end of Part I of this article, we stated that RSA encryption works because it's impractical to factor n, which determines P_1 and P_2, which determines our private key, d, which can be is given by the formula $d = \frac{k \cdot (P_1 - 1)(P_2 - 1) + 1}{e}$ and can be used to decrypt our message. But why is it so hard to factor n? To see why, consider the following; consider the problem of trying to figure out what numbers you have to multiply together to get another number.

First, let's start with a couple of definitions. The numbers that we're going to be dealing with in this problem are natural numbers: positive whole numbers like you're used to dealing with: 1, 2, 3, 4 …. Not fractions, decimals, square roots, and so forth. Second, the numbers you multiply together to get another number are called factors. A prime number is one in which the only factors you can multiply together to get it are 1 and the number itself. A composite number is a number that has factors other than 1 and itself. So 29 is a prime number because the only numbers you can multiply to get it are 1 and 29. On the other hand, 6 is a composite number because you can multiple 2 x 3 to get 6, as well as 1 x 6. Euclid proved about 2300 years ago that every composite number can be expressed as multiple prime numbers, multiplied together. The string of prime factors multiplied together to make an integer are unique. For example, the number 12 has prime factors 2 x 2 x 3. No other number has this combination of prime factors.

Now for the specific problem we want to examine. We want to talk about multiplying two prime numbers together to get a composite number. Let's say you're given the number 203. To determine its prime factors, you'd start with 2 and divide it into 203. It doesn't go evenly so you go to the next prime integer 3. It doesn't divide 203 evenly either. Neither does the next prime integer, 5. How about 7. 7 into 203 goes 29 times. When you get two prime factors that, when multiplied together, give you the number, you're done. So the prime factorization of 203 is 7 times 29.

That's easy. Now suppose you're asked to factor 8,162,821. You wouldn't start with 2 because you know that it won't divide an odd number without a remainder. It turns out that all prime numbers are odd, so you don't have to bother to test 2 or any other even number. So you'd start at 3 and test prime numbers until you found 2 prime factors. Fortunately, you don't have to test

every prime number less than the number you're trying to factor. You only have to test prime numbers up to the greatest integer that is less than or equal to the square root of the number you're trying to factor. (Recall that the square root of a number is a number that, when multiplied by itself, gives you the number you're trying to take the square root of.) Two examples will help to illustrate this.

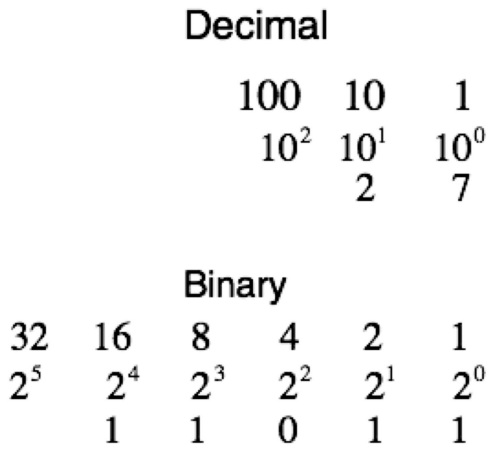
Suppose you're going to factor 49. You start with 2 and try to divide it into 49. It doesn't divide evenly. Same with 3, and 5. You don't need to try 4 or 6 because they're not prime numbers. 7 x 7 works. That's all you need to do. Going any further is redundant. That's because, for every number you test greater than that square root, the number you multiply with it will be less than the square root and will already have been tested. In the worst case, the furthest you'll ever have to go is to an integer that, when multiplied by itself, equals the number (better known as the number's square root). For example, if you want to find prime factors for 119, the furthest you would possibly need to go would be to 10 because the square root of 119 is approximately 10.9. Although, in the case of 119, you wouldn't even need to go that far because 7 x 17 = 119.

Now back to 8,162,821. The square root of 8,162,821 is 2,857 plus a decimal. So you might need to try dividing up to 2,857 numbers into 8,162,821 to figure out if it has prime factors. Of course, the only numbers you would need to divide into 8,162,821 would be prime numbers, so if you knew which numbers less than 2,857 are prime, you would only have to try those numbers and the number of steps to solve the problem would be considerably less than 2,857.

The prime factors that happen to give 8,162,821 are 3011 and 2711. To find those factors, you would have to perform 2711 division steps. "2711 steps?" you say. "That's a lot of steps." And it would be if you were carrying out those steps by hand or with a calculator. However, a computer can still handle checking 2711 numbers in a small fraction of a second. But what if the number you're trying to factor is 617 decimal digits long, like the 2048 bit keys that are often used in RSA encryption. How long would that take? Before we answer this question, we should first find out what a bit is since the term will be used frequently in this discussion.

A bit is the basic information unit with which a computer works. A computer is basically a long string of logic gates hooked together. Each gate can give an answer of yes or no, represented numerically by a 1 or 0, respectively. Thus, computer represent all information as strings of 0's

and 1's. That includes numbers. When numbers are represented in this form, they're called binary or base 2 numbers. The usual numbers we're used to are decimal or base ten numbers. Basically, that means that each digit in the number can be one of ten possibilities, 0-9. With binary numbers, there are only 2 possibilities for each number: 0 or 1. You can represent the same number in either binary or decimal form. The following diagram illustrates an example that, hopefully, will explain this:



Take the decimal number 27. You remember exponents. That's the number of times you have to multiply the base number (in this case, 10) together. So, in the diagram, $10^2 = 10 \times 10 = 100$, $10^1 = 10$, and $10^0 = 1$ (any number to the zero power is 1). 27 in decimal form is 2 groups of tens and 7 groups of ones so you put a 2 in the tens column and 7 in the ones column. There are no hundreds or higher digits so you leave them blank.

You separate binary numbers into columns in a similar fashion. $2^5 = 32$, $2^4 = 16$, $2^3 = 8$, $2^2 = 4$, $2^1 = 2$ and $2^0 = 1$. To express the decimal number 27 in binary form, start by taking the column with the largest value that can be divided into 27, in this case, 16. Divide 16 into 27. It goes once so you have 1 group of 16 in 27. Therefore, put a 1 in the sixteens column. When you divide 16 into 27, you get a remainder of 11. Go to the next column, 8. 8 goes into 11 once so put a 1 in the eights column. You're left with 3. Go to the next column, 4. 4 doesn't go into 3 so put a 0 in the fours column. You still have 3 left over. Go to the twos column. 2 goes into 3 once so put a 1 in the twos column. You've got 1 left over. Go to the ones column. 1 goes into 1 once. Put a 1 into the ones column. You've got no more numbers left so you're done. So 27 in

binary form is 11011. Each column is a bit so 27 is a 5 bit number. It's stored in a computer with five little logical units. Notice that it takes more digits to express a given number in binary form than in decimal form: two digits in decimal form and five digits in binary form in the case of 27. A 2048 bit key like the one that guards directions to the key is a 2048 digit binary number. The decimal equivalent is 617 digits long.

To factor a number this big (call it n), as I described previously, you might need to carry out as many as the square root of n calculations. The square root sign is $\sqrt{\ }$. Taking the square root of a number is the same as the number raised to half its exponent. So

$\sqrt{10^{617}} = 10^{617 \cdot \frac{1}{2}} = 10^{308.5}$. Let's round off and make it 10^{309}.

So you might need to carry out as many as 10^{309} calculations, i.e. divide as many as 10^{309} numbers into n to find the answer. Each of these divisions probably takes perhaps 1000 (i.e. 10^3) computer steps or FLOPs (floating point operations) as they're called. That's probably a conservative estimate, but as you'll see, it really won't matter. A decent computer can carry out about a teraflop each second. That's 10^{12} FLOPs, or computer steps, per second. So a decent computer will be able to carry out 10^9 divisions per second. I got that by doing the following:

$$\frac{10^{12} \cdot \frac{steps}{second}}{10^3 \cdot \frac{steps}{division}} = 10^9 \cdot \frac{steps}{second} \cdot \frac{divisions}{step} = 10^9 \frac{divisions}{second}$$

To figure out how many seconds it'll take to carry out 10^{308} divisions, you divide that by the number of divisions per second your computer can carry out:

$$\frac{10^{309} divisions}{10^9 \frac{divisions}{second}} = \frac{10^{309}}{10^9} \cdot \frac{divisions}{\frac{divisions}{second}} = 10^{300} seconds$$

There are 60 seconds per minute, 60 minutes per hour, 24 hours per day and 365 days per year, so there are 31,536,000 seconds per year:

$$60 \frac{seconds}{minute} \cdot 60 \frac{minutes}{hour} \cdot 24 \frac{hours}{day} \cdot 365 \frac{days}{year} = 31,536,000 \frac{seconds}{year} = 3.1536 \times 10^7 \frac{seconds}{year}$$

To figure out how long it would take—in years—to carry out all 10^{308} division steps that could be necessary to find the secret key, you divide the length of time to do this—in seconds—by the number of seconds in a year:

$$\frac{1\times10^{300}\,seconds}{3.1536\times10^{7}\,\frac{seconds}{year}} = 0.31709792 \times 10^{293} years = 3.1709792 \times 10^{292} years$$

Even if you had 100 computers, it would still take $3.1709792 \times 10^{290}$ years. The universe is about 13.7 billion years old. A billion is 10^9. 13.7×10^9 years equals 1.37×10^{10} years = the age of the universe. $\frac{3.1709792\times10^{290}}{1.37\times10^{10}} \cdot \frac{years}{\frac{yrs}{age\ of\ universe}} = 2.31 \times 10^{280}$ times the age of the universe.

Of course, we only need to test prime. Unfortunately, that doesn't make much difference. An estimate of the number of prime numbers less than a number, n, (referred to as $\pi(n)$) is given by the expression $\pi(n) \approx \frac{n}{ln(n)}$ [1]. In our case, $n = 10^{309}$. I can't find an online calculator that will calculate $ln\left(10^{309}\right)$. However, $ln\left(10^{307}\right) = 706.893623549$ and $ln\left(10^{308}\right) = 709.196208642$. So $ln\left(10^{309}\right)$ is probably in the range of 712. Then $\pi(n) \approx \frac{10^{309}}{712} \approx .014 \times 10^{309} \approx 1.4 \times 10^{307}$. We'd still have to test as many as 1.4×10^{307} numbers - still not practical! There are algorithms other than trial division that can be used to factor large numbers. The most effective of these is the general number field sieve (well beyond the scope of this article; maybe we can tackle it in a future article). However, this still won't make a dent in the problem.

In fact, the largest number ever factored (using the general number field sieve, finished in 2009) had 232 decimal digits (768 bits)[2]. It took hundreds of computers approximately 2 years.

I think you're starting to get the picture. Factoring large numbers with two large prime factors is an extremely difficult problem. And that's why, at the time of this writing, the RSA method is still a safe encryption algorithm.

References

1. Count of prime numbers less than some number, n
https://en.wikipedia.org/wiki/Prime_number_theorem

2. Overview of factoring integers
https://en.wikipedia.org/wiki/Integer_factorization