

Session 15, July 27

Public-Key Encryption using the RSA Algorithm

Until the 1970s, encryption was done using only algorithms which are called *private-key* algorithms. Basically, these algorithms are advanced Secret Decoder Rings – both parties know the encoding and decoding algorithms, and keep both secret. Communication with any party requires both parties to know both algorithms, and if either algorithm slips into enemy hands, the code is compromised, and the entire system must be revised *by everyone involved*, a difficult task for large groups who are using the same coding scheme.

Algorithms known as *public-key* algorithms were first developed in the 1970s, and rely on what are called *back-door functions*, functions which have non-obvious inverse functions. This allows anyone to encode a message, since the encoding method doesn't give any hints to the decoding method. The encoding algorithm is made public (a *public key*) and the inverse function, the decoding algorithm, is kept secret. In public-key encryption, each user has an individual public and private key set, so if you want to send an encoded message to Philip over the phone, you would look up his public key, encode your message with it, and send it, with confidence that only Philip would be able to decode the message.

Furthermore, public-key algorithms are much more difficult to compromise; if someone's decoding algorithm slips into enemy hands, only that person's code is compromised, and it can quickly be replaced without affecting any other users of a large group.

While not the first public-key system to be discovered, the RSA Algorithm (named for the three MIT professors who created it) is the one most frequently implemented on the Internet for security and identity verification. RSA can be used by itself, or as part of a larger encryption like PGP ("Pretty Good Privacy"). RSA is high-quality encryption, and its implementation relies on inverse functions, Fermat's Little Theorem, *and* the Magic Box. How about that!

So, let's figure out how this works, then use the thing. First, some practice with Fermat.

1. What is the value of 17^{28} in mod 29? How can you answer so quickly?
2. In mod 29, what is the value of 17^{56} ? of 17^{29} ? of 17^{57} ? of 17^{28k} ? of $17^{(28k+1)}$? Here, k is any positive integer.
3. In mod 29, what is the value of 13^{85} ? What is the value of $(13^5)^{17}$?
4. In mod 65, what is the value of 11^{48} ? How can you answer so quickly?
5. In mod 65, what is the value of 11^{48k} ? of $11^{(48k+1)}$? of u^{48k} ? of $u^{(48k+1)}$? Here, k is any positive integer, and u is any member of \mathbf{U}_{65} .

Remember that the goal of any decoding algorithm is to be the inverse of the encoding algorithm. In other words, if $E(m)$ is the encoding algorithm, and $D(m)$ is the decoding algorithm, we want $D(E(m)) = m$.

6. Find the smallest value of b for which $(17^5)^b = 17 \pmod{29}$. Look back at your work in Problem 2. What format will the exponent have to have for this to work?

Make sure you know how to do Problem 6 before proceeding. I think the Magic Box is calling.

7. Find the smallest value of b for which $(11^7)^b = 11 \pmod{65}$. Look back at your work in Problem 5. What format will the exponent have to have for this to work?
8. Find the smallest value of b for which $(m^{11})^b = m \pmod{31}$. What format will the exponent have to have for this to work?

In Problem 8, the solution did not rely on the value of m (other than that m is in \mathbf{U}_{31}). This is the key to RSA. In RSA, the encryption algorithm is a *power cipher*, $\mathbf{C} = \mathbf{P}^a \pmod{n}$. Restrictions: \mathbf{P} must be in \mathbf{U}_n , otherwise there will be no way to “undo” the operation. Additionally, the value of a must be relatively prime to $\varphi(n)$.

The decryption algorithm is *also* a power cipher, $\mathbf{P} = \mathbf{C}^b \pmod{n}$. How do we find the value of b ?

Well, the two algorithms are supposed to undo each other, so $(\mathbf{P}^a)^b = \mathbf{P} \pmod{n}$ must be true. Basically, this says if you run a number through BOTH algorithms, you will get the original number back again.

A demonstration is helpful, but you may have found earlier that the new exponent (ab) must be in the form $ab = \varphi(n)k + 1$. Numerical examples: if we were in mod 31, then $ab = 30k + 1$. If we were in mod 65, then $ab = 48k + 1$.

Once you know the value of a , then you find the value of b by the Magic Box method. As a matter of fact, this is the very calculation and setup you performed in Problems 6–8.

One more thing before we proceed.

9. What is $\varphi(5 \times 13)$? What is $\varphi(7 \times 11)$? What is $\varphi(11 \times 3)$? What is $\varphi(5 \times 7)$? What is $\varphi(3 \times 23)$? Use these examples (and others, if necessary) to answer the Big Question: if p and q are distinct prime numbers, what is $\varphi(pq)$?

So... we are finally ready to finish describing RSA. Here is the full RSA algorithm:

- (a) Pick 2 prime numbers, p and q . Multiply them: $pq = n$. This number n will be the modulus.
- (b) According to Problem 9, $\varphi(pq) = (p - 1)(q - 1)$. This will be the mod of the exponents.
- (c) Pick *any* number a which is smaller than, and relatively prime to, $(p - 1)(q - 1)$. (This was the restriction on a mentioned above.)
- (d) Use the Magic Box to solve the Diophantine equation $ab = (p - 1)(q - 1)k + 1$. The value of k is not important, but the value of b is. It is this value of b that will “undo” the power cipher \mathbf{P}^a . Make sure you pick a positive value of b which is less than $(p - 1)(q - 1)$.

That’s all. The *public* key are the numbers a and n , which are made available; this way, anyone can run the encrypting algorithm:

$$\mathbf{C} = \mathbf{P}^a \pmod{n}.$$

But only you know the super-secret number b , which you then use to decode the message:

$$\begin{aligned} (\mathbf{P}^a)^b &= \mathbf{P}^{ab} = \mathbf{P}^{\varphi(pq)k+1} && \text{by the Diophantine equation} \\ \mathbf{P}^{\varphi(pq)k+1} &= \mathbf{P}^{\varphi(pq)k} \mathbf{P}^1 \pmod{pq} && \text{by law of exponents} \\ \mathbf{P}^{\varphi(pq)k} \mathbf{P}^1 &= 1^k \mathbf{P}^1 = \mathbf{P} && \text{by Fermat's Little Theorem} \end{aligned}$$

In other words, it's decoded.

That's the nitty-gritty. You may ask: couldn't people just figure out my super-secret value of b if I tell them what a and n are? The answer is yes. How long it takes them depends on your choice of 2 prime numbers: the larger the numbers, the more difficult it will be to crack. Why? Since a and b were based on the product $(p-1)(q-1)$, someone trying to crack RSA would need to figure out what p and q were. In other words, they need to know how to factor n .

And factoring n is difficult. No one has yet found a "fast factoring" technique that would be a leap above the Sieve of Eratosthenes (other than the guy in *Sneakers*). So, RSA is virtually uncrackable with large enough primes (say, 50 digits or so).

Let's have some practice with RSA. We'll need the TI-92: in particular, its modular arithmetic capability. A program like MATLAB or Mathematica would allow you to extend this exercise with larger numbers.

Typing $\text{mod}(29, 7)$ will give you 1, since $29 = 1 \pmod{7}$. Just type the letters in `mod`, then parentheses. For starters, we'll use $\text{mod } 247 = 13 \times 19$.

10. Encode the message I LOVE RSA using $\text{mod } n = 247$ and public key $a = 7$. Remember that A = 0, B = 1, etc.
11. Pretend that you received the encoded message in Problem 10. Decode it using the private key $b = 31$.

Now, try to exchange messages with a partner. We'll still work in $\text{mod } 247$. First, you'll need to create your own public and private keys.

12. Select a so that a is relatively prime to $\varphi(247) = 216$. Pick a different number than your partner.
13. Compute b by solving the Diophantine equation $ab = 1 + 216k$. Remember: b must be positive, and less than 216.
14. Now tell your partner the public key information (n and a) and have them encode a message. Then, decode the message using your private key.
15. Redo Problems 12–14, but pick your own primes p and q first. Warning: the calculator will not always properly handle numbers larger than about 250^{250} , so you should probably limit the product of primes to under 250.

You may have noticed that A = 0 always gets encoded as A, and B = 1 always gets encoded as B. In actual 128-bit encryption, 16 consecutive characters (8 bits per character) are encoded at once, and there is frequently a shift cipher applied as well as RSA's power cipher. In any case, the encryption is pretty powerful.

How powerful is the RSA algorithm? Currently, 512-bit numbers are used as moduli, a product of two 256-bit primes (2^{256} is about 80 digits). It takes at least a year for a high-end home computer using the fastest available algorithms to break RSA. A 576-bit number...

1881988129206079638386972394616504398071635633794173827007633564229
8885971523466548531906060650474304531738801130339671619969232120573
4031879550656996221305168759307650257059

... is a current Factoring Challenge. The first person to factor this number gets \$10,000 (see RSA.com), and no one has claimed the prize yet. Other, larger numbers, are worth up to \$200,000.