The science of encryption: prime numbers and mod n arithmetic

Go check your e-mail. You'll notice that the webpage address starts with "https://". The "s" at the end stands for "secure" meaning that a process called SSL is being used to encode the contents of your inbox and prevent people from hacking your account. The heart of SSL – as well as pretty much every other computer security or encoding system – is something called a *public key encryption scheme*. The first article below describes how a public key encryption scheme works, and the second explains the mathematics behind it: prime numbers and mod n arithmetic.

1. A Primer on Public-key Encryption

Adapted from a suppliment to The Atlantic magazine, September 2002. By Charles Mann.

Public-key encryption is complicated in detail but simple in outline. The article below is an outline of the principles of the most common variant of public-key cryptography, which is known as RSA, after the initials of its three inventors.

A few terms first: cryptology, the study of codes and ciphers, is the union of cryptography (codemaking) and cryptanalysis (codebreaking). To cryptologists, codes and ciphers are not the same thing. Codes are lists of prearranged substitutes for letters, words, or phrases – i.e. "meet at the theater" for "fly to Chicago." Ciphers employ mathematical procedures called algorithms to transform messages into unreadable jumbles. Most cryptographic algorithms use keys, which are mathematical values that plug into the algorithm. If the algorithm says to encipher a message by replacing each letter with its numerical equivalent (A = 1, B = 2, and so on) and then multiplying the results by some number X, X represents the key to the algorithm. If the key is 5, "attack," for example, turns into "5 100 100 5 15 55." With a key of 6, it becomes "6 120 120 6 18 66." (Nobody would actually use this cipher, though; all the resulting numbers are divisible by the key, which gives it away.) Cipher algorithms and cipher keys are like door locks and door keys. All the locks from a given company may work in the same way, but all the keys will be different.

In non-public-key crypto systems, controlling the keys is a constant source of trouble. Cryptographic textbooks usually illustrate the difficulty by referring to three mythical people named Alice, Bob, and Eve. In these examples, Alice spends her days sending secret messages to Bob; Eve, as her name indicates, tries to eavesdrop on those messages by obtaining the key. Because Eve might succeed at any time, the key must be changed frequently. In practice this cannot be easily accomplished. When Alice sends a new key to Bob, she must ensure that Eve doesn't read the message and thus learn the new key. The obvious way to prevent eavesdropping is to use the old key (the key that Alice wants to replace) to encrypt the message containing the new key (the key that Alice wants Bob to employ in the future). But Alice can't do this if there is a chance that Eve knows the old key. Alice could rely on a special backup key that she uses only to encrypt new keys, but presumably this key, too, would need to be changed. Problems multiply when Alice wants to send messages to other people. Obviously, Alice shouldn't use the key she uses to encrypt messages to Bob to communicate with other people – she doesn't want one compromised key to reveal everything. But managing the keys for a large group is an administrative horror; a hundred-user network needs 4,950 separate keys, all of which need regular changing. In the 1980s, Schneier says, U.S. Navy ships had to store so many keys to communicate with other vessels that the paper records were loaded aboard with forklifts.

Public-key encryption makes key-management much easier. It was invented in 1976 by two Stanford mathematicians, Whitfield Diffie and Martin Hellman. Their discovery can be phrased simply:

enciphering schemes should be asymmetric. For thousands of years all ciphers were symmetric – the key for encrypting a message was identical to the key for decrypting it, but used, so to speak, in reverse. To change "5 100 100 5 15 55" or "6 120 120 6 18 66" back into "attack," for instance, one simply reverses the encryption by dividing the numbers with the key, instead of multiplying them, and then replaces the numbers with their equivalent letters. Thus sender and receiver must both have the key, and must both keep it secret. The symmetry, Diffie and Hellman realized, is the origin of the key-management problem. The solution is to have an encrypting key that is different from the decrypting key – one key to encipher a message, and another, different key to decipher it. With an asymmetric cipher, Alice could send encrypted messages to Bob without providing him with a secret key. In fact, Alice could send him a secret message even if she had never before communicated with him in any way.

"If this sounds ridiculous, it should," Schneier wrote in Secrets and Lies (2001). "It sounds impossible. If you were to survey the world's cryptographers in 1975, they would all have told you it was impossible." One year later, Diffie and Hellman showed that it was possible, after all. (Later the British Secret Service revealed that it had invented these techniques before Diffie and Hellman, but kept them secret – and apparently did nothing with them.)

To be precise, Diffie and Hellman demonstrated only that public-key encryption was possible in theory. Another year passed before three MIT mathematicians – Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman – figured out a way to do it in the real world. At the base of the Rivest-Shamir-Adleman, or RSA, encryption scheme is the mathematical task of factoring. Factoring a number means identifying the prime numbers which, when multiplied together, produce that number. Thus 126,356 can be factored into 2 x 2 x 31 x 1,019, where 2, 31, and 1,019 are all prime. (A given number has only one set of prime factors.) [1] Surprisingly, mathematicians regard factoring numbers – part of the elementary-school curriculum – as a fantastically difficult task. Despite the efforts of such luminaries as Fermat, Gauss, and Fibonacci, nobody has ever discovered a consistent, usable method for factoring large numbers. Instead, mathematicians try potential factors by invoking complex rules of thumb, looking for numbers that divide evenly. For big numbers the process is horribly time-consuming, even with fast computers. The largest number yet factored is 155 digits long. It took 292 computers, most of them fast workstations, more than seven months.

Note something odd. It is easy to multiply primes together. But there is no easy way to take the product and reduce it back to its original primes. In crypto jargon, this is a "trapdoor": a function that lets you go one way easily, but not the other. Such one-way functions, of which this is perhaps the simplest example, are at the bottom of all public-key encryption. They make asymmetric ciphers possible.

To use RSA encryption, Alice first secretly chooses two prime numbers, p and q, each more than a hundred digits long. This is easier than it may sound: there are an infinite supply of prime numbers. Last year a Canadian college student found the biggest known prime: $2^{13466917} - 1$. It has 4,053,946 digits; typed without commas in standard 12-point type, the number would be more than ten miles long. Fortunately Alice doesn't need one nearly that big. She runs a program that randomly selects two prime numbers for her and then she multiplies them by each other, producing pq, a still bigger number that is, naturally, not prime. This is Alice's"public key." (In fact, creating the key is more complicated than I suggest here, but not wildly so.)

---

[1] Pop quiz: which one of our theorems from class says this?

As the name suggests, public keys are not secret; indeed, the Alices of this world often post them on the Internet or attach them to the bottom of their e-mail. When Bob wants to send Alice a secret message, he first converts the text of the message into a number. Perhaps, as before, he transforms "attack" into "5 100 100 5 15 55." Then he obtains Alice's public key – that is, the number pq – by looking it up on a Web site or copying it from her e-mail. (Note here that Bob does not use *his own* key to send Alice a message, as in regular encryption. Instead, he uses Alice's key.) Having found Alice's public key, he plugs it into a special algorithm invented by Rivest, Shamir, and Adleman to encrypt the message.

At this point the three mathematicians' cleverness becomes evident. Bob knows the product pq, because Alice has displayed it on her Web site. But he almost certainly does not know p and q themselves, because they are its only factors, and factoring large numbers is effectively impossible. Yet the algorithm is constructed in such a way that to decipher the message the recipient must know both p and q individually. Because only Alice knows p and q, Bob can send secret messages to Alice without ever having to swap keys. Anyone else who wants to read the message will somehow have to factor pq back into the prime numbers p and q.[2]

In the real world, public-key encryption is practically never used to encrypt actual messages. The reason is that it requires so much computation – even on computers, public-key is very slow. According to a widely cited estimate by Schneier, public-key crypto is about a thousand times slower than conventional cryptography. As a result, public-key cryptography is more often used as a solution to the key-management problem, rather than as direct cryptography. People employ public-key to distribute regular, private keys, which are then used to encrypt and decrypt actual messages. In other words, Alice and Bob send each other their public keys. Alice generates a symmetric key that she will only use for a short time (usually, in the trade, called a session key), encrypts it with Bob's public key, and sends it to Bob, who decrypts it with his private key. Now that Alice and Bob both have the session key, they can exchange messages. When Alice wants to begin a new round of messages, she creates another session key. Systems that use both symmetric and public-key cryptography are called hybrid, and almost every available public-key system, such as PGP is a hybrid.[3]

[2]The next article will give you an indication of how amazingly difficult this is

[3]Or SSL. PGP is the encryption process used for most secure computer databases, whereas SSL is typically used over the internet. It is also a hybrid: Encoding and decoding the contents of an entire webpage using public-key encryption would slow down your internet browser too much. Instead, a public-key is used to send a temporary private key that lets you decode the encrypted data from the website. Every time you visit facebook or gmail, the private key changes, and your information is kept secure.

2. The math behind RSA encryption

Adapted from a text by math educator Tom Davis. You can find this material, and more, at http://mathcircle.berkeley.edu/BMC3/rsa/node4.html

It is very simple to multiply numbers together, especially with computers. But it can be very difficult to factor numbers. For example, if I ask you to multiply together 34537 and 99991, it is a simple matter to punch those numbers into a calculator and 3453389167. But the reverse problem is much harder.

Suppose I give you the number 1459160519. I'll even tell you that I got it by multiplying together two integers. Can you tell me what they are? This is a very difficult problem. A computer can factor that number fairly quickly, but (although there are some tricks) it basically does it by trying most of the possible combinations. For any size number, the computer has to check something that is of the order of the size of the square-root of the number to be factored. In this case, that square-root is roughly 38000.

Now it doesn't take a computer long to try out 38000 possibilities, but what if the number to be factored is not ten digits, but rather 400 digits? The square-root of a number with 400 digits is a number with 200 digits. The lifetime of the universe is approximately 10^{18} seconds - an 18 digit number. Assuming a computer could test one million factorizations per second, in the lifetime of the universe it could check 10^{24} possibilities. But for a 400 digit product, there are 10^{200} possibilities. This means the computer would have to run for 10^{176} times the life of the universe to factor the large number.

It is, however, not too hard to check to see if a number is prime–in other words to check to see that it cannot be factored. If it is not prime, it is difficult to factor, but if it is prime, it is not hard to show it is prime.

So RSA encryption works like this. I will find two huge prime numbers, p and q that have 100 or maybe 200 digits each. I will keep those two numbers secret (they are my private key), and I will multiply them together to make a number $N = pq$. That number N is basically my public key. It is relatively easy for me to get N; I just need to multiply my two numbers. But if you know N, it is basically impossible for you to find p and q. To get them, you need to factor N, which seems to be an incredibly difficult problem.

But exactly how is N used to encode a message, and how are p and q used to decode it? Below is presented a complete example, but I will use tiny prime numbers so it is easy to follow the arithmetic. In a real RSA encryption system, keep in mind that the prime numbers are huge.

In the following example, suppose that person A wants to make a public key, and that person B wants to use that key to send A a message. In this example, we will suppose that the message A sends to B is just a number. We assume that A and B have agreed on a method to encode text as numbers. Here are the steps:

(1) Person A selects two prime numbers. We will use $p = 23$ and $q = 41$ for this example, but keep in mind that the real numbers person A should use should be much larger.
(2) Person A multiplies p and q together to get $pq = (23)(41) = 943$. 943 is the "public key", which he tells to person B (and to the rest of the world, if he wishes).
(3) Person A also chooses another number e which must be relatively prime to $(p-1)(q-1)$. In this case, $(p-1)(q-1) = (22)(40) = 880$, so we could choose the number $e = 7$. This

number e is also part of the public key, so B also is told the value of e. [See footnote[4] for a remark on why we're using the number $(p-1)(q-1)$.]

(4) Now B knows enough to encode a message to A. Suppose, for this example, that the message is the number $M = 35$.
(5) B calculates the value of $C = M^e (\text{mod } N) = 35^7 (\text{mod } 943)$.
(6) $35^7 = 64339296875$ and $64339296875 (\text{mod } 943) = 545$. The number 545 is the encoding that B sends to A.
(7) Now A wants to decode 545. To do so, he needs to find a number d such that $ed = 1 (\text{mod } (p-1)(q-1))$, or in this case, such that $7d = 1 (\text{mod } 880)$. A solution is $d = 503$, since $7 \times 503 = 3521 = 4(880) + 1 = 1 (\text{mod } 880)$.
(8) To find the decoding, A must calculate $C^d (\text{mod } N) = 545^{503} (\text{mod } 943)$. This looks like it will be a horrible calculation, and at first it seems like it is, but notice that $503 = 256 + 128 + 64 + 32 + 16 + 4 + 2 + 1$ (this is just the binary expansion of 503). So this means that

$$545^{503} = 545^{256+128+64+32+16+4+2+1} = 545^{256} 545^{128} \cdots 545^{1}.$$

The line above just uses basic rules about how exponents work. Now since we only care about the result (mod 943), we can calculate all the parts of the product (mod 943). By repeated squaring of 545, we can get all the exponents that are powers of 2. For example, $545^2 (\text{mod } 943) = 545 \cdot 545 = 297025 (\text{mod } 943) = 923$. Then square again: $545^4 (\text{mod } 943) = (545^2)^2 (\text{mod } 943) = 923 \cdot 923 = 851929 (\text{mod } 943) = 400$, and so on. We obtain the following table:

$$\begin{aligned}
545^1 (\text{mod } 943) &= 545 \\
545^2 (\text{mod } 943) &= 923 \\
545^4 (\text{mod } 943) &= 400 \\
545^8 (\text{mod } 943) &= 633 \\
545^{16} (\text{mod } 943) &= 857 \\
545^{32} (\text{mod } 943) &= 795 \\
545^{64} (\text{mod } 943) &= 215 \\
545^{128} (\text{mod } 943) &= 18 \\
545^{256} (\text{mod } 943) &= 324
\end{aligned}$$

So the result we want is:

$$545^{503} (\text{mod } 943) = 324 \cdot 18 \cdot 215 \cdot 795 \cdot 857 \cdot 400 \cdot 923 \cdot 545 (\text{mod } 943) = 35.$$

Using this slightly tedious (but simple for a computer) calculation, A can decode B's message and obtain the original message $N = 35$. The fact that this actually works – that the process in step (8) gives you the answer – depended on us using the number $(p-1)(q-1)$ and the reason behind that is some beautiful mathematics involving the ϕ function.

[4] Why the number $(p-1)(q-1)$? It has to do with the ϕ function that we talked about in class. When p and q are prime numbers, $\phi(pq) = (p-1)(q-1)$.

Exercises

(1) Summarize, in non-mathematical terms, how public-key encryption works.

(2) Is public-key encryption more secure or at least less risky than private-key encryption? What are the main advantages and disadvantages of each?

(3) Follow through all the steps of RSA encryption as outlined in Davis' article, using the prime numbers $p = 17$, $q = 19$ and $e = 11$ to encode the message "81" as some other number, and then decode it back. Hint: to save you some time in step (7), a number d such that $ed = 1 (\text{mod } (p-1)(q-1))$ is the number $d = 131$. But you must check this to make sure that it works, i.e. show that 1 is the remainder when you divide $(p-1)(q-1)$ by ed.
Be sure to show all of your work and write down all of the steps.