Lecture 4: Public key encryption

4.1 Classical crypto

The classical setup for secure communication sees **A**lice trying to communicate with **B**ob, while an eavesdropper called **E**ve tries to listen or look at the communication learning anything she can. One setup with some historical justification is a "book code", which will be used here principally to introduce vocabulary.

Alice and Bob are given copies of the same book. They are told that messages from one to the other will be made secure by looking at a page on the book and adding each letter in a message with a corresponding letter on the page, mod 26. So if Alice wants to send Bob a message beginning "MAY WE MEET SOON ..." and the first words on the page are "ONCE UPON A TIME ..." then the first letter Eve would send would be B. That's because M is the thirteenth letter in the alphabet, O is the fifteenth, and $13 + 15 = 28 = 2 \bmod 26$, and B is the second letter in the alphabet.

In this case the entire arrangement (sending each letter, adding letter values mod 26, selecting pages, etc.) would be called a *cryptosystem*. We could think of Eve with various abilities. Minimally, Eve might just know the transmitted letter. Eve might be more powerful, and might know the entire system, and might even know the book used. She might not know the specific page of the book selected, and that would usually be called the *key* of the system. Generally it is assumed that an attacker knows the cryptosystem, but does not know the key. This assumption, formulated explicitly about a century ago, is called "Kerckhoffs maxim" and is named after a Dutch cryptographer who spent most of his life in France. The original work (from 1883) of Kerckhoffs is available on the web, just a click away!

Now communication is digital*. The assumption is frequently made that the cryptosystem is implemented via computer, and that Eve has much greater computer resources (both speed and storage) than Alice and Bob. This corresponds with some real situations, where Eve might be a government agency or a large corporation.

4.2 Public key encryption †

Much cryptography is done with high-speed machines using clever programs. A constant problem is to make sure that the machines can talk to each other. These machines should have the same keys. Key distribution has historically been a problem: in the 1500's,

---

* More than a century ago, Kerckhoffs had written another desirable attribute of cryptosystems: "Il faut qu'il soit applicable à la correspondance télégraphique." ("It must be applicable to telegraphic correspondence.")

† Some of this material follows notes written by Saša Radomirović, a math grad student. See `http://www.math.rutgers.edu/~greenfie/teaching/display/diary.html`. Links there have information about the course he helped with during spring 2000.

a country might need to tell its ambassadors which pages in a book to use on which days, or communication might be seriously impaired. A critical question might be phrased: how can keys be distributed "publically" without compromising security?

In the digital environment this is even more important. Hundreds of thousands of users might want to communicate securely with each other, and reality is that, effectively, there can be little prearrangement of keys. The users might agree on cryptosystems, but for a long time transactions were obstructed by a lack of ability to communicate keys securely.

The problem is: Alice wants to send Bob a message (the key). Eve will listen, Eve will know the protocols, and Eve has much greater computer resources than Alice and Bob. Is it possible for Alice and Bob to exchange information securely in spite of these obstacles? The surprising and perhaps unintuitive answer is "Yes". There are several solutions, and here we discuss RSA*. Another solution in widespread use is the Diffie-Hellman method, and there have been a number of other proposals.

4.3 Introducing RSA

If P and Q are distinct primes, Euler's generalization of Fermat's Little Theorem states

$$(\bullet) \quad a^{(\text{any integer})(P-1)(Q-1)} = 1 \bmod PQ$$

for certain integers, a. I'll forget temporarily about the restriction on a until later. It will turn out that no (operational) problem will occur. Notice that if T is a number with $T = 1 \bmod (P-1)(Q-1)$, then $a^T = a \bmod PQ$. Suppose Alice chooses a number e for **e**ncryption. The linear equation $ex = 1 \bmod (P-1)(Q-1)$ can be efficiently solved by the Euclidean algorithm. Suppose d is a solution of this equation. Then $(a^e)^d = a^{de} = a^{ed} = a^{(\text{an integer})(P-1)(Q-1)+1}$. This is a consequence of the mod equation $(\bullet)$. But mod PQ the complicated stuff in the exponent vanishes, and result of all the computation is just $a^1 = a$! So take a, compute $b = a^e \bmod PQ$, compute $b^d \bmod PQ$, and the result must be a. Here is a step-by-step description of RSA, a public key encryption protocol.

Step 1: Alice prepares Alice selects distinct primes P and Q. She computes $N = PQ$, selects e and d so that $ed = 1 \bmod (P-1)(Q-1)$. Alice makes public only N and e, so *everyone* knows N and e. Please note: here again there may appear to be an operational difficultly – selecting e and finding d. I'll address this later, also. It is not a serious problem.

Step 2: Bob encrypts Bob wants to send a message, a, to Alice. He computes $b = a^e \bmod N$ and sends that to Alice.

Step 3: Alice decrypts Alice computes $b^d \bmod N$ and gets the message, a.

That's it, a communications revolution, one which has made governments very annoyed for several decades. In *Privacy on the line*, Diffie and Landau remark that both hardware and software changes have created obstacles to "codebreaking": the use of fiber-optic

* R=Ron Rivest, S=Adi Shamir, and A=Len Adleman, and, yes, this is the same Shamir as in the secret sharing section. They published a description of this system in 1977. It had likely been discovered by other people earlier. See the bibliography.

communications has led to a reduced need to broadcast communcations by radio, and that's been combined with common use of public key encryption.

4.4 An example

In "the real world" nobody would ever use numbers as small as what is here. This is just a toy which I hope will help you understand the steps above.

Step 1: Alice prepares Alice chooses 11 and 5 as her primes, and computes $N = 11 \cdot 5 = 55$. She chooses $e = 3$ and finds that $d = 27$. She finds d by solving $3 \cdot x = 1 \bmod 10 \cdot 4$. Alice "publishes" $N = 55$ and $e = 3$.

Step 2: Bob encrypts Bob knows $N = 55$ and $e = 3$. Suppose Bob's message is $a = 9$. He computes $a^e \bmod N$, which is $9^3 \bmod 55 = 14$. He sends $b = 14$ to Alice.

Step 3: Alice decrypts Alice gets 14 from Bob. She computes $14^{27} \bmod 55$. Her answer is 9, which is Bob's message. That's no surprise, since $(9^3)^{27} = 9^{81} = 9^{4 \cdot 20+1} = \left(9^{20}\right)^4 \cdot 9^1 = 1^4 \cdot 9 = 9 \bmod 55$ by Euler's Theorem.

4.5 How to break it

Is RSA any good? Any serious cryptosystem should be evaluated critically, and such evaluation such probably be redone fairly often. What does Eve know and can she "break" RSA? We assume that Eve knows that Alice and Bob are using this system. In the example above, Eve knows $N = 55$, $e = 3$, and the encrypted message, $b = 14$. Eve also knows the Alice and Bob are using RSA. Solving $a^3 = 14 \bmod 55$ might take some time. There's an easier way, as Eve tries to imitate Alice's decryption.

Eve must know d to decrypt this easily. But to find d efficiently, she has to know P and Q! In this example, Eve has little problem factoring 55 as $5 \cdot 11$. Then her computer solves $3x = 1 \bmod 4 \cdot 10$ rapidly, and gets the answer 27 for d. Eve can now decrypt every message sent to Alice the same way that Alice does.

Much effort has been devoted to finding ways to break RSA. Some special cases can be attacked easily, but there are defenses against these. Factoring the modulus is the best known general attack on RSA, and some people believe it is the only general attack.

4.6 The real world

In the real world the primes P and Q which are currently used each have about 170 digits. Many people are trying to factor numbers which are the product of two big primes. The largest such number factored so far is a product of two 78 digit primes and is known as RSA-155. This result was announced on August 22, 1999 by Dr. H. J. J. te Riele at the CWI in Amsterdam (which is the national research institute for mathematics and computer science in the Netherlands) and was a result of joint work involving many organizations and people.

The factoring process for RSA-155 took about 7 months and more than 300 workstations were involved. At the end, a Cray supercomputer was needed. The amount of computer time needed was about the equivalent of 800 years on a standard PC. This comparison is not useful because the amount of memory needed was almost 20 times as much

as a standard PC has. Some details: 2048 MB of RAM and up to 3.7 GB of hard disk space were needed. The factoring process had various stages. At the end a huge system of linear equations had to be solved. This part required an enormous amount of memory, because there were about 6,700,000 equations with 6,700,000 unknowns to be solved. The primes are 102 63959 28297 41105 77205 41965 73991 67590 07165 67808 03806 68033 41933 52179 07113 07779 and 106 60348 83801 68454 82092 72203 60012 87867 92079 58575 98929 15222 70608 23719 30628 08643. respectively.

Surely writing RSA-155 itself would be a silly adolescent display.

10941 73864 15705 27421 80970 73220 40357 61200 37329 45449 20599 09138 42131 47634 99842 88934 78471 79972 57891 26733 24976 25752 89978 18337 97076 53724 40271 46743 53159 33543 33897

Let me return to commenting on the operational difficulties disclosed earlier. We need a, the "message", to be an integer between 0 and PQ which is *not* divisible by P or Q. In practice, P and Q are primes, each with many digits. In order to guarantee that the message a is not divisible by P or Q require that a be *less than* both P and Q. For example, require that a be less than 10^{150} when P and Q are in the range mentioned above. That allows many possible messages.

In Alice's step 1, there's another possible difficulty. Alice must select e and d so that $ed = 1 \bmod (P-1)(Q-1)$. Generally, the linear equation (constant)$x = 1 \bmod N$ has a unique solution if the constant and N have no common factors. Some condition is necessary: for example, the equation $3x = 1 \bmod 6$ has *no* solution. But Alice needs e and d so that $ed = 1 \bmod N$. She can select e at random and try to solve the equation $ex = 1 \bmod N$ using the Euclidean algorithm, which is very rapid (polynomial time: see section 5.5). If e and N do have a common factor, the algorithm will terminate unsatisfactorily. Another choice of e can be tried repeatedly until one having no common factor with N is found.

4.7 Authentication

A public key system like RSA can serve other social goals in addition to confidential communication. Here's one: Bob gets the message, "I love you." It is supposedly signed by Alice. Is there some way he can be sure she sent it? If you are less romantic, you can contemplate financial transactions or, perhaps, diplomatic or military scenarios where the issue of authentication is important: securely identifying the sender as well as the message.

Alice could take her message, a, and compute $b = a^d \bmod N$. She then sends b to Bob. Recall that no one else knows d or could discover it easily. Bob receives b and computes $b^e \bmod N$. This is possible since e and N are known to everyone. But $(a^d)^e = a^{de} = a^{ed} = a \bmod N$ just as before. So Bob gets the message $a =$"I love you." Since he decrypted it using Alice's public information (e and N), he is fairly sure it was sent by Alice, using Alice's private information, d.

If Alice wished to keep her message private and share it only with Bob, she could encrypt it with Bob's public key. She could then take the resulting message and "sign it" as described above, by using her d. This double layer method then sends the message securely and authenticates it as well. But things can get very complicated rapidly when using these methods. For all of these uses, a collection of of public keys has to be available,

and somehow these public keys must themselves be truthful, and users must somehow be guaranteed correctness.

The type of message authentication discussed here is a source for what are called "digital signatures", now permitted by U.S. law. It also allows good verification for such processes as voting and auctions. Maybe Lessig's quote given earlier ("encryption ... the most important technological breakthrough") isn't that exaggerated!

4.8 Bibliography

The original paper on RSA:

[1] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, February 1978 issue of the Communications of the ACM (this is the Association for Computing Machinery), volume 21, pages 120–126. You can get it on the web.

You can read about factoring RSA-155 and also see further factoring challenges here:

[2] `http://www.rsasecurity.com/rsalabs/challenges/factoring/index.html`.

Members of British secret government communications agencies found the commonly used public key systems before the public academic community. Here's a link to some information. Also see section 0.5's reference [2].

[3] `http://www.research.att.com/~smb/nsam-160/`

The National Security Agency (NSA) is the U.S. government's chief agency charged with communications security. Their website is

[4] `http://www.nsa.gov`

Twice I've given courses to introduce students with little mathematical background to the math and computer science of cryptography, emphasizing some of the social and legal questions surrounding this subject. A good high-school math background is sufficient to understand the math parts of the course. Look at my homepage

[5] `http://www.math.rutgers.edu/~greenfie`

and follow the links to the two sections of Math 103. Two of our graduate students, Nina Fefferman and Saša Radomirović, helped me teach these courses. There are links to a wide variety of web pages discussing crypto and society. Even if the technical side is not attractive, I'd hope that some of the issues discussed would be interesting to you. Some discussion of these courses is also available at

[6] `http://www.math.rutgers.edu/~greenfie/teaching/display/diary.html`

A source of information on the policy and implementation problems regarding digital signatures is

[7] `http://www.epic.org/crypto/dss/`.

Lecture 5: How hard is arithmetic?

5.0 How many H_20's ...

If students help, I will overestimate the number of molecules of water on earth.

5.1 Adding numbers; multiplying numbers

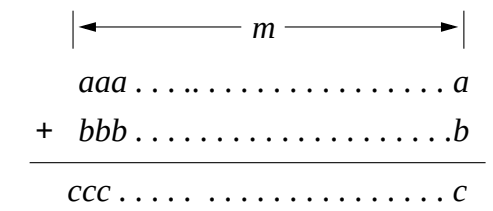
Let's return to traditional integer arithmetic, not modular arithmetic. Most of what will be said is correct with minor modifications for modular arithmetic. We begin by considering carefully two difficult problems.

```
  34 566          463
+ 788 381       × 219
  -------       ------
```

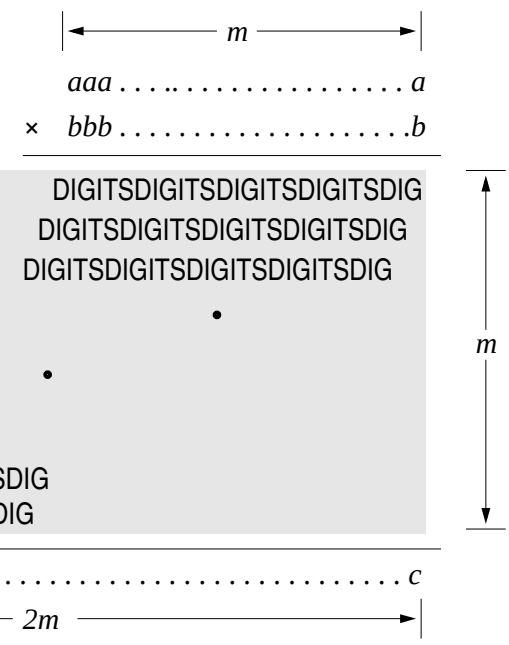
5.2 Counting operations

The specific answers to these computations are not important here. I'd like to analyze how many "elementary operations" are needed. In this case, I think of elementary operations as the number of references to the standard base 10 one digit addition and multiplication tables. This is one way to measure how "hard" a computation is. I assume that the + and × tables are memorized, once and for all. Only looking things up in these tables takes work.

Addition Let's overestimate the number of one digit operations required for addition. If A and B have length m, the addition scheme for $A+B$ might look approximately as shown here. A is written $aaa\ldots a$, B is $bbb\ldots b$, and $A+B$ is $ccc\ldots c$. There may be m direct additions of digits of A and B. There may also be carries. Maybe there are m additions from carries. The total number of one digit additions is about $2m$.



Multiplication Now to get an overestimate for the number of one digit operations required for multiplication. This is a much more complicated scheme. The idea of explaining it to students is tremendously intimidating! Now A and B are just as before, but $A\cdot B$ is $ccc\ldots c$, the product. There are less than $2m^2$ single digit multiplications in the box shown. There could addition carries inside the box: $2m^2$ is an overestimate of the possible single digit additions (as carries). The number of arithmetic operations is less than $4m^2$.



The number of digits seems to determine the work done. That is, multiplying 26 by 34 is about as much work as multiplying 45 by 37, even though both factors in the second product are greater numbers. If A is an integer, let $\#(A)$ be the number of decimal digits of A, the *length* of A. So $\#(427)$ is 3. $\#(A)$ is the log of A to the base 10, rounded up to the nearest integer. $\#(A) = 3$ occurs exactly when $10^2 = 100 \leq A \leq 999 = 10^3 - 1$. We've seen that if A and B both have length m, addition takes at most $2m$ elementary operations, and multiplication takes at most $4m^2$ such.

5.3 What's an algorithm?

A precise definition of **algorithm** is difficult, which is interesting since the concept is central to much of mathematics and computer science during the last quarter century. It is as vital and important to such study as the **sonnet** is to the history and practice of poetry. Here are some quotes from Knuth's *The Art of Computer Programming.*

From page 1:

> The word "algorithm" itself is quite interesting; at first glance it may look at though someone intended to write "logarithm" but jumbled up the first four letters. ... the true origin of the word ... comes from the name of a famous Persian textbook author, Abu Ja'far Mohammed ibn Mûsâ al-Khowârismî (*c.* 825) – literally, "father of Ja'far, Mohammed, son of Moses, native of Khowârizm." Khowârizm is today the small Soviet city of Khiva. Al-Khowârizmî wrote the celebrated book *Kitab al jabr w'al-muqabala* ("Rules of restoration and reduction"); another word, "algebra", stems from the title of his book, although the book wasn't really very algebraic.

From pages 4, 5, and 6:

> The modern meaning for algorithm is quite similar to that of *recipe, process, method, technique, procedure, routine,* except that the word "algorithm" connotes something just a little different. Besides merely being a finite set of rules which gives a sequence of operations for solving a specific type of problem, an algorithm has five important features:
>
> **1) Finiteness.** An algorithm must always terminate after a finite number of steps. ...
>
> **2) Definiteness.** Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case. ...
>
> **3) Input.** An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects. ...
>
> **4) Output.** An algorithm has one or more outputs, i.e., quantities which have a specified relation to the inputs. ...
>
> **5) Effectiveness.** An algorithm is also generally expected to be *effective.* This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time ...

Knuth continues on the same page to contrast his definition of algorithm with what could be found in a cookbook:

> Let us try to compare the concept of an algorithm with that of a cookbook recipe: A recipe presumably has the qualities of finiteness (although it is said that a watched pot never boils), input (eggs, flour, etc.) and output (TV dinner, etc.) but notoriously lacks definiteness. There are frequently cases in which the definiteness is missing, e.g., "Add a dash of salt." A "dash" is defined as "less than $\frac{1}{8}$ teaspoon"; salt is perhaps well enough defined; but where should the salt be added (on top, side, etc.)? ...

He concludes his comparison by writing:

> ... a computer programmer can learn much by studying a good recipe book.

A description of the basic algorithms of arithmetic is usually done in the primary school grades. Addition, multiplication, and subtraction are relatively easy to teach. Division ("long division") is complicated to show and analyze.

5.4 Factoring

Let's describe an algorithm which would decide if an integer were prime or not, and, if it were not, would report factors of the integer. What is written here is emphatically not the fastest or easiest (in terms of total work) way to answer this question. But it can be described simply, and analyzed with what we've already done.

My factoring algorithm

> For each pair of numbers (i, j) where i and j are any integers between 2 and N, compute the product ij. If this is equal to N, report that i and j are factors. If this is never equal to N, report that N is a prime.

If $N = 6$, this algorithm would require 25 (that is, 5^2) multiplications of numbers between 2 and 6. To decide if a number N is prime or can be factored, the algorithm would require at most $(N-1)^2$ multiplications of numbers between 2 and N. The algebra is simpler if we estimate that at most N^2 multiplications of pairs of numbers at most N will be required.

5.5 Polynomial time algorithms

If $\#(A) = \#(B) = m$, computing $A + B$ will require about $2m$ operations, computing $A \cdot B$ will require about $4m^2$ operations, and detecting factors of A will require about $4m^2A^2$ operations. Let's try to do some proportions. Suppose A and B both have about 100 digits, and suppose the timing of solution of these three problems is as follows:

Computing $A + B$ takes .0001 seconds;
Computing $A \cdot B$ takes .001 seconds;
Deciding **Factoring** for A takes 1 second.

How long would it take to decide these questions for 200 digit numbers? m gets doubled, so $2m$ gets doubled. $4m^2$, however, gets multiplied by 4. This still isn't too bad. We can report the following, based on our analysis and our assumptions:

Computing $A + B$ takes .0002 seconds;
Computing $A \cdot B$ takes .004 seconds.

What about factoring? The time/work is proportional to $4(\#(A))^2A^2$. When $\#(A) = 100$, the time is 1. If K is the constant of proportionality here, $K\left(4(100)^2(10^{99})^2\right)$ is 1. Doubling

the length of A changes $\#(A)$ to 200 and A itself to 10^{199}. The amount of time should be approximately computable with the following ratio:

$$\frac{1}{4(100)^2(10^{99})^2} \cdot 4(200)^2(10^{199})^2$$

and therefore when A changes from 100 to 200 digits,

Deciding **Factoring** for A takes about $4 \cdot 10^{200}$ seconds.

This is a very large number. A century has about 10^8 seconds. Of course we made some overestimates to come to this conclusion, but this number is unreasonably large. Maybe from the human point of view it is effectively eternity.

In terms of the length of the input, the algorithms we have given for integer addition and multiplication have work/time requirements which are polynomial functions of the length of the input, and the miserable and silly algorithm we have given here for factoring has work/time demand which is actually *exponential* in terms of the length of the input. That is because A itself is approximately $10^{\#(A)}$, and $4m^2A^2$ is about $4m^2 10^{2m}$.

A now-classical source (*The Design and Analysis of Computer Algorithms* by Aho, Hopcroft, and Ullman, Addison-Wesley, 1974) declares (p. 364):

> How much computation should a problem require before we rate the problem as being truly difficult? There is general agreement that if a problem cannot be solved in less than exponential time, then the problem should be considered completely intractable. The implication of this "rating scheme" is that problems having polynomial-time-bounded algorithms are tractable. ... we say that a problem is *intractable* if all algorithms to solve that problem are of at least exponential time complexity.

5.6 Computational complexity

Algorithms for arithmetic depend on polynomial multiples of their input length. The specific solution to the factoring algorithm given here requires time which is exponential in its input length. But there is *no known* polynomial-time algorithm for factoring.

Factoring seems to be very hard

Since the only known general attack on RSA depends on factoring, RSA seems reasonably secure if the size of the modulus is large enough. Therefore, the security of RSA depends on the long record of *failure* to invent fast factoring algorithms!

The systematic study of the difficulty of computations only began in the late twentieth century and is called *Computational Complexity.*

The factoring problem has the following interesting structure. It seems to be hard to solve (exponential time). If a candidate for a solution is given (that is, two numbers whose product is supposed to be the integer considered) then the answer can be checked in polynomial time by multiplication. Factoring is a specific case of the fundamental problem of Computational Complexity, called the **P versus NP problem**. Here is an approximate statement:

> Is there a problem whose suggested solutions all can be checked in time which is polynomial in the input size, but which cannot be solved in polynomial time?

No one knows the answer, and many people have worked on this problem. An ideal crypto protocol would involve such a problem. Both Alice and Bob "solve" the problem in polynomial time with their encryption and decryption steps, but Eve must check more than polynomial number of alternatives (hopefully, an exponential number!).

win ... win ... win ... win ... win

One million dollars!

Deciding if P and NP are the same or different is one of the seven problems which the Clay Mathematical Institute has called "Millennium Prize Problems". Each of these problems has a $1,000,000 prize for solution. You can read both an official problem description and a "popular article" connecting P/NP to the minesweeper computer game at the link `http://www.claymath.org/Millennium_Prize_Problems/P_vs_NP/` . The P/NP problem is probably the only one of the seven Clay Institute problems which can be understood without an extravagant amount of background.*

5.7 Big exponentiating

Part of the RSA scheme requires computation of A^B mod C where A, B, and C are large numbers. It seems that $B-1$ multiplications of numbers between 0 and $C-1$ are needed. This isn't polynomial time in $\#(B)$. But there may be more than one way to solve a problem (that's why P versus NP is not "clear"!). Even though one algorithm to solve a problem may not be polynomial time, another may be.

Let me do an example first, omitting the mod which doesn't really matter here. The idea is to drastically reduce the number of multiplications by using previous computations.

How could we compute A^{46}? Naively we could compute $A^2 = A \cdot A$, $A^3 = A^2 \cdot A$, $A^4 = A^3 \cdot A$, etc. So A^4 requires three multiplications. But $A^4 = (A^2)^2$, so that computing A^4 can be done with two multiplications. But repeated squaring is the clue to a more efficient method. Consider the powers of 2: $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, and $32 = 2^5$. Since $46 = 32 + 8 + 4 + 2$, $A^{46} = (A^2)^5 \cdot (A^2)^3 \cdot (A^2)^2 \cdot A^2$. Computing A^2 takes one multiplication, $(A^2)^2$ takes one additional multiplication, $(A^2)^3 = (A^2)^2 \cdot (A^2)$ takes one additional multiplication, and $(A^2)^5 = ((A^2)^2)^2 \cdot (A^2)$ takes two additional multiplications. Assembling the product (counting the $\cdot$'s) takes three additional multiplications. So computation of A^{46} can be done in 8 multiplications, much less than 46 multiplications.

This repeated squaring implies that the exponent (here 46) should be written as a sum of powers of 2. This is called *binary notation*. So $46_{10} = 100110_2$ where $101110_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^0$. The method described here repeatedly squares A until the power corresponding to the highest 1 in the binary expansion of A is reached. Then assemble A from what's already been computed by including any powers of A which correspond to 1's in the binary expansion of A. Actually things can even be done more efficiently, relying on questions of parity (even/odd) of A and $A/2$ recursively.

Why does this method allow us to compute A^B in polynomial time? It certainly allows the number of multiplications which are each polynomial time to be reduced essentially to

* See `www.cs.umd.edu/~gasarch/papers/poll.ps` which reports a recent (2002) poll of professionals: their opinions of P/NP and when the problem might be solved (and how!).

the twice the length of the binary expansion of B. But the length of the binary expansion of B is between three and four times the length of the base 10 expansion of B since $2^3 < 10 < 2^4$. The number of multiplications needed is no more than $8\#(B)$.

Exponentiation is at the core of many encryption algorithms, so there has been much research in how to exponentiate *fast*. The bibiliography contains a link to a recently written discussion of this problem.

5.8 Finding primes

How can we get prime numbers? How can we get BIG prime numbers? For example, $2^{240\,36583} - 1$, which has over seven million digits, is prime*. Large prime numbers are useful for both RSA and secret sharing. It would be silly in the latter application to use a small prime, since then "brute force search" would be good enough to reveal the secret.

An old time † proof

There are an infinite number of primes. This is because if there were only a finite list of primes, $\{p_1, p_2, \ldots, p_t\}$, the number $p_1 \cdot p_2 \cdots p_t + 1$ gotten by multiplying the contents of this list together and adding 1 would have to be divisible by some *other* prime number. This contradicts the original assumption of a finite list of primes.

Current practice

There are many prime numbers, but how can they be found? Many cryptographic applications need big prime numbers. For example, PGP ("pretty good privacy") is a widely used collection of programs implementing RSA-like algorithms. The original developer, Phil Zimmermann, wanted to allow everyone to generate their own public and private keys. But large primes are needed for such applications. Advanced number theory suggests that eight and a half percent of 5 digit numbers are prime, more than four percent of 10 digit numbers are prime, and more than two percent of 20 digit numbers are prime. The primes do thin out as numbers get larger, but still more than .4% of 100 digit numbers are prime. What people do in practice is based on the existence of fast (polynomial time!) tests that report if a number is *probably* prime. Choose a random 100 digit number. Check a few times with independent tests to see if it is probably prime. If it fails a test, take another number. Continue to do this until success results. This "bit flipping" may be unsatisfactory to those who love certainty, but it works in the real world.

PRIMES is in P

The problem of testing whether a specific integer is prime is called PRIMES by theoretical computer scientists. The mathematical world was surprised recently (the summer of 2002!) when Manindra Agrawal, Neeraj Kayal, and Nitin Saxena ("AKS") of the Indian Institute of Technology in Kanpur, India, found a polynomial time algorithm for testing primality. Their algorithm is described in a remarkably brief (9 pages!) paper available on the web. I certainly believe that you would understand the introduction. Please see the bibliography for precise references.

---

* See `http://www.utm.edu/research/primes/` or `http://www.mersenne.org` for further information.

† Euclid's *Elements* again, written more than two thousand years ago.

The method announced does not itself threaten the security of RSA encryption. Many people believe breaking RSA is equivalent to the difficulty of *factoring* as previously discussed. The discovery of the AKS algorithm has not yet led to major changes in how encryption is done. The probabilistic primality algorithms still seem to be a lot faster for the range of primes in use.

5.9 Bibliography

Here's the major reference for algorithms in computer science. The second volume has a discussion of long division extending over 4 pages, with further problems later – the algorithm *is* intricate!

[1] Donald Knuth, *The Art of Computer Programming*, Addison-Wesley, published over the last 30 years or so. (Three volumes, lots and lots of pages, priced at $135 and worth every penny.)

Another classical reference about algorithms:

[2] Alfred Aho, John Hopcroft, and Jeffrey Ullman, *The Design and Analysis of Computer Algorithms* 1974, Addison-Wesley ($55 and 470 pages).

Here is a link to an English translation of the major work of "the leading mathematics teacher of antiquity or perhaps of all time" (Euclid). I believe the results used here appear in Books VII, VIII, and IX.

[3] `http://aleph0.clarku.edu/~djoyce/java/elements/elements.html`

Much has been learned about algorithms in the last 20 years, and about teaching them. More contemporary references follow. The first is usually used as a graduate level text. The next two are used in undergraduate courses.

[4] Thomas Cormen, Charles Leiserson, and Ronald Rivest, *Introduction to Algorithms*, MIT Press (available in paperback in September 2001 for about $50, over a thousand pages).

[5] Robert Sedgewick, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996 (about 500 pages, $45).

[6] Robert Sedgewick, *Algorithms in C++ : Parts 1-4 : Fundamentals, Data Structures, Sorting, Searching*, Addison-Wesley, 1998 (about 730 pages, $50).

Want to find a big prime? Here's a Java implementation of a fancy factoring algorithm.

[7] `http://www.alpertron.com.ar/ECM.HTM`

Or you can join GIMPS, the Great Internet Mersenne Prime Search.

[8] `http://www.mersenne.org/prime.htm`

[9] Richard Crandall and Carl Pomerance *Prime Numbers: A Computational Perspective*, Springer Verlag, 2001 (545 pages, $50). Learn the latest about primes, factoring, etc. One

review on Amazon remarks: "While graduate-level, much of it should be accessible by an undergraduate. ... It's a good reference - no need to read the whole thing."

The classical view of "computational complexity" was presented here. Within the last decade, a different model of computation has been investigated: quantum computing (and, of course, quantum cryptography). This model uses some of the wonderful phenomena of quantum mechanics to make NP computations more tractable. Much of this approach is still hypothetical, however: people can't get it to work satisfactorily yet! This web page has references under the `Tutorials` link.

[10] `http://www.qubit.org/`

If you want to exponentiate *fast* look at

[11] D. Gordon, *A survey of fast exponentiation algorithms, Journal of Algorithms*, volume 27 (1998), pages 129–146, or look at `http://www.ccrwest.org/gordon/jalg.ps` on the web.

The AKS paper:

[12] M. Agrawal, N. Kayal, and N Saxena, *Primes in in P*, which can be obtained at `http://www.cse.iitk.ac.in/primality.pdf`. The entire abstract of the paper follows:

> We present a deterministic polynomial-time algorithm that determines whether an input number n is prime or compositve.

The link `http://www.cse.iitk.ac.in/news/primality_v3.pdf` has a more recent (March 2003) version of the paper.

Here is a commentary on the AKS discovery:

[13] F. Bornemann, *PRIMES is in P: A Breakthrough for "Everyman"*, Notices of the American Math. Soc., volume 50 (2003), pages 545–552. You can also get this on the web at `http://www.ams.org/notices/200305/fea-bornemann.pdf`.

5.10 Appendix: finite fields

We decided that we should be able to add, subtract, multiply, and divide. It was also very convenient that the standard algebraic rules hold, such as associativity, commutativity, and distributivity, and that there were additive and multiplicative identities (0 and 1) and inverses for addition and multiplication. A set with addition and multiplication obeying all these rules is called a *field*. Examples of fields are the real numbers and the rational numbers, but not the integers (you can divide by a non-zero integer!). Modular arithmetic (with a prime modulus) allowed us to construct **finite fields**. These are not the only finite fields. There are others whose rules of addition and multiplication are not quite as simple, but which are really useful in certain signal processing applications. For example, there is exactly one finite field having 49 elements, and maybe that's the size needed for a certain application. We'll discuss this more when we get to coding theory.