Public Key Cryptography

Shai Simonson
Stonehill College

Introduction

When teaching mathematics to computer science students, it is natural to emphasize constructive proofs, algorithms, and experimentation. Most computer science students do not have the experience with abstraction nor the appreciation of it that mathematics students do. They do, on the other hand, think constructively and algorithmically. Moreover, they have the programming tools to experiment with their algorithmic intuitions.

Public-key cryptographic methods are a part of every computer scientist's education. In public-key cryptography, also called trapdoor or one-way cryptography, the encoding scheme is public, yet the decoding scheme remains secret. This allows the secure transmission of information over the internet, which is necessary for e-commerce. Although the mathematics is abstract, the methods are constructive and lend themselves to understanding through programming.

The mathematics behind public-key cryptography follows a journey through number theory that starts with Euclid, then Fermat, and continues into the late 20th century with the work of computer scientists and mathematicians. Public-key cryptography serves as a striking example of the unexpected practical applicability of even the purest and most abstract of mathematical subjects.

We describe the history and mathematics of cryptography in sufficient detail so the material can be readily used in the classroom. The mathematics may be review for a professional, but it is meant as an outline of how it might be presented to the students. "In the Classroom" notes are interspersed throughout in order to highlight exactly what we personally have tried in the classroom, and how well it worked. The use of history is primarily for context and for interest. There are great stories to tell, and the results are better appreciated in context. Plenty of references are given for those who would like to extend our work and design their own labs.

The material presented here was taught in a *learning-community* course at Stonehill College [5]. The course is a three-way collaborative of three courses: Discrete Mathematics, Data Structures, and Mathematical Experiments in Computer Science. Students register for all three courses. The first two courses are standard lecture and discussion. The third course is a *closed* lab: a scheduled time when students work on specially prepared laboratory assignments and interact with the computer in order to discover solutions and principles. The course has five 3-week units, one of which is Cryptography. In each unit: the first week is spent analyzing, proving theorems, and writing programs; the second week is spent using the programs and theorems of the first week for experimenting and exploring; and the third week is used for "enrichment", which usually means videos, stories, and related material of a lighter nature. The course is interactive, with only short impromptu lectures during which we can react on the spot to the student's questions and experiments. The labs appear at the Stonehill College computer science department's website [24].

A Motivating Puzzle

In the 1995 movie *Die Hard: With a Vengeance (aka Die Hard III)*, Bruce Willis and Samuel L. Jackson play a bomber's deadly game as they race around New York trying to prevent explosions. In order to stop one explosion, they need to solve the following puzzle:

> Provided with an unlimited water supply, a 5-gallon jug, and a 3-gallon jug, measure out precisely 4 gallons, by filling and emptying the jugs. (See photo in figure 1).

Figure 1: Bruce Willis and Samuel L. Jackson Solving a Mathematical Puzzle.

This puzzle is useful for studying public-key cryptography, because the solution embodies the two major related number-theoretic results: Euclid's algorithm and Fermat's Little Theorem. Of course this puzzle was not invented for this movie. Most references attribute the puzzle and its variations to Tartaglia [29], but the earliest known version of the problem occurs in the Annales Stadenses compiled by Abbot Albert of the convent of the Blessed Virgin Mary in Stade. Stade is a small city on the west side of the Elbe estuary a bit downriver from Hamburg. The date of compilation is uncertain, but seems to be 1240 [27]. The puzzle is also discussed in detail at the wonderful educational mathematics site created by Alex Bogolmony [4]. Bogolmony retells the story of how the famous French mathematician Simeon Poisson (1781-1840) was pressured against his will by his father to study medicine. Poisson stumbled upon the puzzle, solved it, and decided to abandon medicine in favor of mathematics.

> In the Classroom: The Poisson story allows us to digress and discuss what kinds of mathematics are inspiring to what kinds of people. We wonder whether there are any budding mathematicians who are inspired by viewing Die Hard III.

A Solution to the Puzzle

When solving this puzzle, it doesn't take the students long to realize that the only thing worth doing is to repeatedly pour one container into the other, emptying the second container when it gets filled. For example, to solve the case presented in the movie, where 3-gallon and 5-gallon jugs must be used to measure out four gallons, we fill the 3-gallon jug, pour it into the 5-gallon jug, fill 3 again, pour it into 5, empty 5, finish pouring 3 into 5, yielding 1 gallon in the 5-gallon jug, fill 3 once again, and pour into 5, yielding four gallons. In this way, we could cycle through the following values in the 5-gallon jug: 3, 1, 4, 2, and 0. One could also pour in the opposite direction. That is, from the 5-gallon into the 3-gallon jug, giving the cycle of values: 2, 4, 1, 3, and 0. The students discover that every value modulo 5 is obtainable and that the two cycles are the reverse of each other.

> In the Classroom**:** We screen the scene in the movie and compare it to the solution the class discovered on its own. Afterwards, we try to solve the problem in its general form given jugs of sizes *a* and *b*. We do this with an informal class discussion, allowing the class to follow dead ends up to a point, and eventually coming to some conclusions.

Using Programs to Experiment and Understand

Gerry Sussman, one of the authors of the award winning text *Structure and Interpretation of Computer Programs*, once explained [28] that one grasps a mathematical idea if and only if one is able to write a program to illustrate it.

> In the Classroom**:** In our closed labs, the students write a program to generate possible values for the quantities in the jugs. The programming is used to reveal and reinforce an understanding of this puzzle. It is an exercise that supports Sussman's hypothesis.

When the students write a program to generate the cycles, it becomes clear that the sequence of values obtained by repeatedly pouring 3 into 5, is simply the successive multiples of 3 modulo 5:

$$1\times3 \bmod 5 = 3$$
$$2\times3 \bmod 5 = 1$$
$$3\times3 \bmod 5 = 4$$
$$4\times3 \bmod 5 = 2$$
$$5\times3 \bmod 5 = 0$$

The students notice that if x appears in this cycle then $x = 3u - 5v$, for some u and v. For example, when measuring four gallons with 3-gallon and 5-gallon jugs, we calculate: $4 = 3\times3 - 5\times1$.

The students look at another example. If the jugs held 17 and 7 gallons then the cycle would be: 7, 14, 4, 11, 1, 8, 15, 5, 12, 2, 9, 16, 6, 13, 3, 10, 0. Because 1 is the fifth number in the list, $7\times5 = 1 \bmod 17$, and $1 = 7\times5 - 17\times2$. Naturally then:

$$2 = 7\times10 - 17\times4$$
$$3 = 7\times15 - 17\times6$$
$$4 = 7\times20 - 17\times8$$
$$\ldots$$

Therefore, after they can calculate 1, they can also calculate the rest of the numbers by using multiples of the values used for 1. Looking at the cycle, the numbers 1 through 16 can be counted in order, by starting with 1 and moving five to the right each time, cycling around when you hit the end of the list. In general, given two jugs of sizes a and b, $a > b$, the values that can be obtained are the values between 0 and a that are multiples of the smallest positive value that can be obtained.

What is this smallest number that can be obtained from pouring water back and forth between jugs of sizes a and b? In other words, given a and b, $a>b>0$, how can we find the smallest positive integer x such that $au + bv = x$?, where u and v are integers.

Euclid's algorithm gives the answer: $x = \gcd(a,b)$, the greatest common divisor of a and b. Hence, a complete solution to the two jug puzzle for sizes a and b, $a > b$, is that we can measure all multiples of the greatest common divisor of a and b. In particular, when a and b are relatively prime, all values less than or equal to a can be obtained.

Euclid's Algorithm

The students' first stop in mathematical history is Euclid. Euclid's method provides a fast recursive algorithm, given a and b, to calculate u and v, such that $au+ bv = gcd(a,b)$. Euclid's algorithm is described in The Elements, Book VII, Proposition 2 [13].

To find the greatest common measure of two given numbers… Let AB and CD be the two given numbers not relatively prime. It is required to find the greatest common measure of AB and CD. If now CD measures AB, since it also measures itself, then CD is a common measure of CD and AB. And it is manifest that it is also the greatest, for no greater number than CD measures CD. But, if CD does not measure AB, then, when the less of the numbers AB and CD being continually subtracted from the greater, some number is left which measures the one before it…

In the Classroom**:** We take this opportunity to briefly discuss Euclid's Elements and its place in the history of mathematics. We also spend some time discussing Euclid's style of writing and in particular the absence of any algebra or modern notation. Eventually, we interpret the paragraph above well enough to deduce the famous algorithm, that given $a>b$, $\gcd(a,b)$ can be computed recursively as follows: **gcd(a,b): if *b* divides *a* then return(*b*) else return(gcd(*b*, *a* mod *b*)).**

The students try some examples. For 28 and 123, the tail recursive algorithm (tail recursion means that the recursion does nothing on the way back but pass the result upwards) computes: gcd(123, 28) = gcd(28, 11) = gcd(11, 6) = gcd(6, 5) = gcd(5, 1) = 1.

Working backwards the students recursively calculate u and v such that $123u + 28v = 1$. We start at the penultimate recursive call by calculating a linear combination of 6 and 5 that equals the greatest common divisor:

(A) $1 = 6 – 1\times5$.

Now, using $5 = 11 – 1\times6$, and substituting for 5 in equation (A) gives:

(B) $1 = 6 – 1\times(11 – 1\times6) = 2\times6 – 1\times11$.

Continuing backwards, using $6 = 28 – 2\times11$, and substituting for 6 in equation (B), we get:

(C) $1 = 2(28 – 2\times11) – 1\times11 = 2\times28 – 5\times11$.

Similarly, using $11 = 123 – 4\times28$, and substituting for 11 in equation (C), we get:

(D) $1 = 2\times28 – 5\times(123 – 4\times28) = 22\times28 – 5\times123$.

Euclid's Algorithm Extended

This idea can be used to extend Euclid's algorithm, so that given $a>b$, the algorithm below, CalculateUV(a,b), calculates u and v, such that $au + bv = \gcd(a,b)$. This algorithm is not tail recursive, and the results are changed as they are passed back up. Note that the division is integer division, and truncates the remainder.

```
CalculateUV(a, b):
if a mod b = gcd(a,b) then return (u,v) = (1, –a/b).
else let (u', v') = CalculateUV(b, a mod b), and return(v', u' –(a/b)v').
```

In the last example, when a = 123 and b = 28:
CalculateUV(123, 28) calls CalculateUV(28, 11), which calls CalculateUV(11, 6), which calls CalculateUV(6,5), which returns (1, –1).
Winding back from the recursion:
CalculateUV(11, 6) returns (–1, 1 – 1×(–1)) = (–1, 2).
CalculateUV(28, 11) returns (2, –1 – 2×2) = (2, –5).
CalculateUV(123, 28) returns (–5, 2 – 4×(–5)) = (–5, 22).
This means that gcd(123, 28) = 123×(–5) + 28×22 = 1.

In the Classroom**:** As short as this algorithm may be, we find that students always have some trouble remembering exactly how to do the calculations. In order to make the idea more concrete, they are asked to code the algorithm, and to print out and interpret the values at each level of recursion.

The Complexity of Euclid's Algorithm

The following brief discussion regarding the time complexity of Euclid's algorithm will be important later when we talk about public key cryptography. Euclid's algorithm is usually not taught to middle school children, perhaps because the reason why it works is not obvious, or perhaps because for small numbers other methods seem more intuitive, simpler, and faster. Here are two intuitive methods for calculating greatest common divisors that *are* usually found in middle school textbooks.

Given $a>b$, gcd(a,b) can also be computed by:

1. Factoring:
Factor a and b into prime factors, and take the intersection of the prime factors.

2. Brute Force:
Try all the numbers from b down to 1, and return the first one that divides both a and b evenly.

Of course, middle school children should learn how to factor, and should understand why these two methods work. From that point of view, these algorithms are worth teaching, however, both algorithms have a horribly large time complexity. The time complexity of the first algorithm is proportional to the size of $a+b$, and that of the second is proportional to the size of b.

In contrast, Euclid's algorithm is very fast. Gabrielle Lame (1795-1870), using Fibonacci numbers, proved that the complexity of Euclid's algorithm, including the extended version CalculateUV, is proportional to the number of digits in b [23]. This time complexity is exponentially faster than the middle school algorithms described in the last paragraph! On 100-digit numbers, the two slow algorithms will take hundreds of centuries even on the world's fastest computer, while Euclid's algorithm will appear to work instantaneously even on the hand-me-down PCs often donated to middle schools!

In the Classroom**:** Students write programs for the greatest common divisor in three different ways. Sure, one way would be enough, and the *right* way is Euclid's simple recursive formula. However, by doing it the other two ways, we introduce a side lesson about computational complexity, which parallels the discussion of the complexity for factoring that is so crucial later on. Simple measurements are done to illustrate the speed differences between these algorithms. We focus on the preoccupation that theoretical computer scientists have had with exponential versus polynomial time complexity since the beginnings of computational complexity in the 1960s [10].

Fermat's Little Theorem

The next stop through mathematical history that relates to modern cryptography is Fermat's Little Theorem, not to be confused with Fermat's Last Theorem. Fermat's Little Theorem was stated in a letter from Fermat to the amateur mathematician Frenicle de Bessy (1605-1675) dated October 18, 1640.

Fermat's Little Theorem:
Let p be a prime that does not divide the integer a, then $a^{p-1} = 1 \bmod p$.

Fermat wrote "I would send you the demonstration, if I did not fear its being too long" [19]. Frenicle de Bessy was an excellent amateur mathematician but not the equal of Fermat, and he was not able to provide a proof. He wrote angrily to Fermat and although Fermat gave more details in his reply, Frenicle de Bessy felt that Fermat was teasing him. Bessy was able to solve many other problems posed to him by Fermat, but it was Euler who first published a proof of Fermat's Little Theorem in 1736.

In the Classroom**:** As with Euclid, we spend a brief time reviewing the life and influence of Fermat. We emphasize how Fermat published almost nothing in his lifetime, and gave no systematic explanation of his methods. Instead, he left his results on the margins of works that he had read and annotated, or in letters to mathematicians of his day. He almost always described his results without proof, leaving the details of the proofs to other mathematicians, who usually supplied them. We make sure the class knows about the famous Last Theorem, and contrast it with Fermat's Little Theorem which is less famous but more applicable. One interesting note is that the same Gabrielle Lame, who proved the logarithmic complexity of Euclid's algorithm, also proved the special case of Fermat's Last Theorem when $n = 7$. For both Fermat and Euclid, there are dozens of good references, and the reader can follow the lists in [16].

Euler's proof, essentially identical to an unpublished version by Leibniz around 1680, can be understood and motivated by the same water jug puzzle that helped us with Euclid. Consider again the case of a 17-gallon jug and a 7-gallon jug. The sequence of quantities that end up in the 17-gallon jug as we repeatedly fill and empty the 7-gallon jug into it are: 7, 14, 4, 11, 1, 8, 15, 5, 12, 2, 9, 16, 6, 13, 3, 10, 0.

In the Classroom**:** Students use the program they wrote earlier that generates the successive possible values for the quantities in the jugs, and experiment to see what happens when the sizes of the two jugs are relatively prime versus when they are not. Examples and special cases are always helpful when trying to motivate the statement and proof of a theorem.

The students notice how the 16 multiples of 7 are all distinct modulo 17. This would be true for any pair of relatively prime numbers a and p. If any two of the $p–1$ multiples of a were equal modulo p then their difference, call it ax, would be divisible by p. However, p does not divide *ax*. By assumption, *p* does not divide *a*, and *p* does not divide *x* because *x* is less *p*. Hence *p* does not divide *ax*, because:

> Euclid (VII.30)
> *If two numbers, multiplied by one another make some number, and any prime number measures the product, then it also measures one of the original numbers.*

Now if the $p-1$ multiples of a are all distinct modulo p then their product, $a\times 2a\times 3a\times\ldots\times(p-1)a$ equals the product $1\times 2\times\ \ldots\ \times (p-1)$, modulo p. Fermat's Little Theorem follows by dividing both sides of the equality by $1\times 2\times\ \ldots\ \times (p-1)$. With the proof of Fermat's Little Theorem in hand, we are ready to study public-key cryptography.

Cryptography: A Brief History

The history of cryptography covers thousands of years with dozens of interesting stories [14, 25], including:

- The story of Lysander of Sparta in 404 BC who decoded a message written on a belt by winding the belt around a wooden staff, and reading the letters that appeared adjacent to each other, (see figure 2). He learned that Persia was planning an attack, and he was able to thwart the attack.

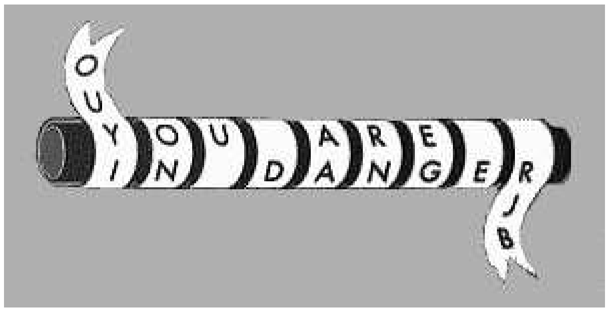

Figure 2: Lysander Decoding the Transposition Code

- The story of Mary, Queen of Scots, who was executed after her encoded correspondence regarding her plan to murder Queen Elizabeth I was intercepted and decoded.
- The strange story of the Beale Ciphers, describing the hidden location of a fortune in gold, buried somewhere in Virginia in the nineteenth century and still not found.
- The cracking of German codes by Alan Turing in Bletchley Park, England, during World War II. This story is brought to life by Sir Derek Jacobi in the Broadway play and later PBS TV special *Breaking the Code*.

A complete history of cryptography is beyond our scope here, and the reader is referred to [14, 25]. We will review only enough of cryptographic history to give context to the work of Rivest, Shamir, and Adelman [21].

> In the Classroom**:** These stories are all very colorful and fun. They catch the interest of the class, who are about to get hit with some much less colorful and more difficult mathematics.

The code of Lysander was a *transposition* code where the letters transmitted were correct but scrambled out of order. The winding of the belt put them back in order. Modern cryptographic methods are all *substitution* ciphers in which letters or numbers are replaced by other letters or numbers.

One of the oldest substitution ciphers is called the Caesar cipher, invented by the Roman Emperor Julius Caesar. Caesar's method is similar to the decoder rings given away in cereal boxes in the 1960s (see figure 3).

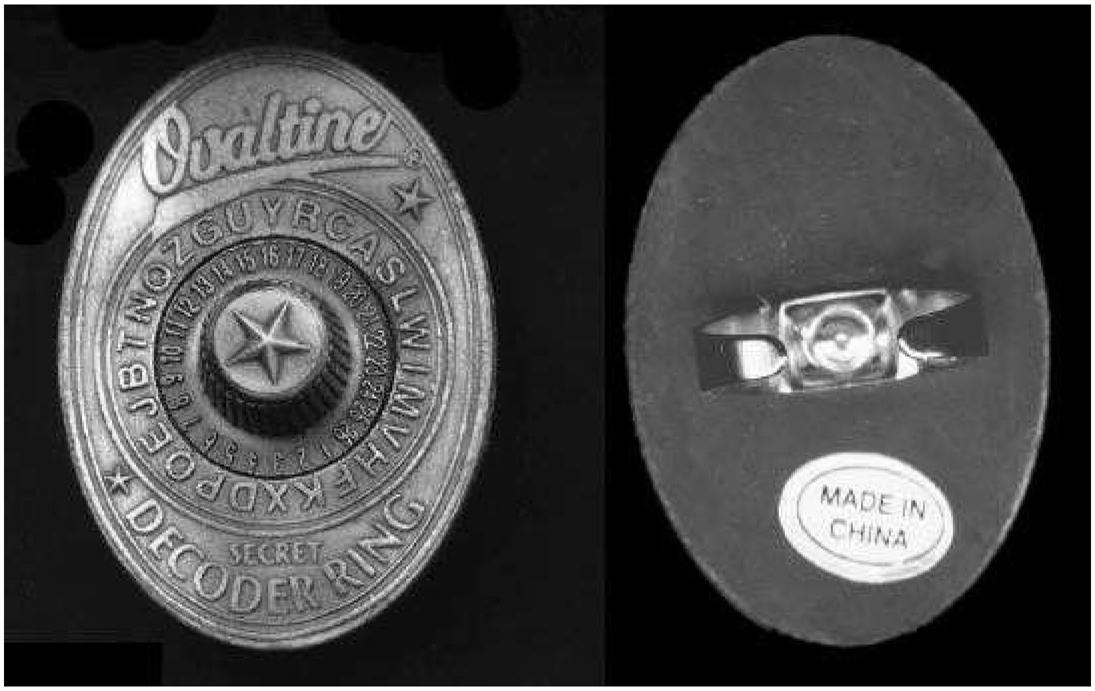

Figure 3: Ovaltine Nostalgia Decoder Ring

The alphabet is rotated by some number of letters, so for example if we rotate five places, "a" becomes "f", "b" becomes "g", …, and finally "z" becomes "e", wrapping all the way around. The substitutions for this rotation (call it the "f-shift") are shown below:

abcdefghijklmnopqrstuvwxyz
fghijklmnopqrstuvwxyzabcde

For example, Caesar's famous phrase "veni vidi vici" ("I came I saw I conquered") would read "ajsn anin anhn" using the f-shift. Actually, Caesar himself was partial to the c-shift. A codebreaker only has to try 25 possibilities to break an encoding like this. By translating the message 25 times using the reverse of each possible letter shift, until one translation looks like a real message, he will eventually find the real message.

A better substitution method that is much more difficult to decode is called the Vigenere Cipher. Blaise de Vigenere (1523-1596), from the court of Henry III described his method in a book titled *A Treatise on Secret Writing*. Vigenere was also known for his 1578 rationalist treatise debunking superstitions about comets. "In this little book various statements regarding comets as signs of wrath or causes of evils are given, and then followed by a very gentle and quiet discussion, usually tending to develop that healthful skepticism which is the parent of investigation." [31]

The Vigenere cipher is an example of a *polyalphabetic substitution* cipher, in which each subsequent letter uses a different shift. After a while these shifts cycle. The sequence of shifts is represented by a codeword consisting of letter shifts. For example, if the codeword is "remarkable", then the first letter of the message would shift by 17, the second letter by 4, …, the 7th letter by 0, etc. The 11th letter would shift by 17 again like the first and the cycle would repeat.

For example, using the codeword "remarkable", the sentence "we meet in new york for a rendezvous" would be encoded as "ni yevd io yin carb pos l vvrpeqfovd". Note that spaces between words are not actually transmitted or encoded. For a codeword like "remarkable" of length ten, there are 26^{10} different possible encodings. The longer the codeword, the more possibilities there are.

In the Classroom**:** Students write a program that encodes and decodes a Vigenere cipher. The program takes plaintext along with a codeword and produces encrypted text. In this exercise as well as others that follow, we have the students compete against each other by creating messages for each other to decode. This is a lot of fun, and the students love it.

Breaking the Vigenere Cipher: Determining the Length of the Codeword

The first step in breaking a Vigenere cipher is to determine the length of the codeword. There are a number of ad hoc ways of doing this including the work of Kasiski (1863) and Kerckhoff (1883). The former searches for commonly recurring *bigrams* (two letter combinations) that might encode words like "me" or "it", hoping that the distances between repeated occurrences will be the length of the codeword. The latter tries different lengths, groups the letters into disjoint sets each of which contains letters that use the same substitution, and sees if the sorted frequencies of the encoded letters match the sorted expected English frequencies. This latter method also suggests guesses of the actual key word. However, neither of these methods is as reliable as the more systematic method of Philip Friedman published in 1922 [8].

Friedman's ingenious method to determine the length of the codeword is described by David Kahn [14] as "the most important single publication in cryptology." Friedman defines the *index of coincidence*, a statistical measure of text, that for normal English is about 6.6%, but for a random collection of letters is only about 3.8%. He uses this measurement to calculate the length of the codeword.

To determine the index of coincidence of a particular piece of text, take the text, rotate it by some random number of places, and write the rotated text underneath the original text. For example, the sentence below is rotated by 58 characters and the rotated version appears beneath it.

alanturingbreakscodeslikenobodysbusinessbuthispersonallifesadlybecameeverybodysbusiness
dysbusinessbuthispersonallifesadlybecameeverybodysbusinessalanturingbreakscodeslikenobo

The index of coincidence is the number of places in which the same letter occurs in both strings of text. In this example, there are 6 coincidences for 87 characters of text, a rate of 6.9% and close to the 6.6% of normal English text. An important point to notice is that when a text is encrypted with a Caesar cipher, where every letter is shifted by the same value, the index of coincidence remains constant.

For a Vigenere encryption, we would expect to see an index of coincidence more like the 3.8% of random letters, *unless* we happen to have rotated by a multiple of the Vigenere cycle length. In the case where the rotation is a multiple of the cycle length, the letters that are lined up underneath each other are encoded using the same shift, and the usual English index of coincidence would be expected, because the index of coincidence is invariant under a Caesar cipher. Hence the way to determine the Vigenere cycle length is to rotate the encrypted text by 1, 2, 3, etc. symbols, until we see an index of coincidence that looks more like 6.6% than 3.8%.

The Vigenere cipher itself did not catch on, but variations of it did, including the Gronsfeld cipher, (which is essentially a Vigenere cipher with a codeword of digits), and the more complex ciphers of the Germans during World War II. Until the work of Kasiski, Kerkehoff, and later Friedman, the Vigenere cipher and its variants, were for 300 years considered unbreakable, especially if long codewords were used and short messages were encoded.

> In the Classroom**:** Students write a program using Friedman's technique to determine the cycle lengths of messages encrypted with different codewords. Seeing the correct percentage pop up is perhaps the best way to appreciate Friedman's amazing contribution to code breaking.

Breaking the Vigenere Cipher Given the Length of the Codeword

Once one knows the length of the cycle, 10 for the "remarkable" example, there are a number of techniques for breaking the code without having to try all the possibilities. One idea is to divide the letters of the encoded message up into 10 groups, one for each shift in the cycle. The 1st, 11th, 21st, etc. make the first group. The 2nd, 22nd, 32nd, etc. make the second group. And so on. The letters in each group are encoded using the same shift. This method is very much like Kerckhoff's method but without having to guess the length of the codeword.

We try to decode the letters a group at a time. For each group, we try all 26 possible shifts, but since the letters in each group are scattered throughout the message, we are unlikely to learn anything by doing this. The chance of the partially decoded message looking familiar will be very small.

Nevertheless, there is a way to learn something about the message. Every language has a characteristic statistical frequency for each letter. For example, in English the letter "e" is the most frequently used letter. For a given group, we compare the frequencies of the letters in each of the 26 decodings to the expected frequencies. Matching the two sets of frequencies helps identify the shift or at least narrows down the number of possibilities from 26 to just a few. Of course this method requires a large text sample.

> In the Classroom**:** We have not yet designed a lab to experiment with this method. It requires a linear least squares regression and our students have not yet studied this in their math classes. Instead, we have the students design more ad hoc methods for guessing the letters, which allow for computer-human cooperation. These methods force the students to be creative and unique. The methods are also true to the history of breaking codes. The kind of programming that went on at Bletchley Park during World War II was exactly this ad hoc style of combining analytical methods with practical necessity. It was the combined power of machine and human problem solving that ultimately cracked the German codes.

In World War II, the Germans used an encoding scheme similar to the Vigenere cipher but more complex. They used a machine to generate the letter substitutions, and the cycles were extremely long. The machine the Germans used was called the Enigma. An online simulator of such a machine can be found at [6]. Breaking Nazi codes required the sophisticated computer-aided decoding effort led by Alan Turing.

The important thing to note about the Vigenere Cipher and all its variants is that *if* one knows the encoding method (via espionage for example), then the decoding is trivial. At first thought, a kind of encryption where the encoding method is not easily reversible might appear impossible. However, if such a scheme were possible, then even if the encoding method were to fall into the wrong hands, the enemy could still not easily decode messages! Now we are ready to jump ahead to the late twentieth century to the work of Rivest, Shamir and Adelman.

Public Key Cryptography: The RSA Breakthrough of 1978

Cryptographic methods do not have to be reversible! Today a new kind of encoding is used which is called *public-key* cryptography (or *one-way*, or *trapdoor*). With this new method, the whole world is able to encode messages, but unless Sam Hacker has more information, he still cannot decode a message.

This new method is what allows e-commerce to flourish without fear of a security breach. Suppose I want to send my credit card number to Amazon.com. I encode my number with a publicly published method that anyone else could use called the *public key*, but only Amazon.com can decode it because only they have the *private key*!

Authentication

What if I don't trust that I am actually talking to Amazon.com? That is, I suspect that Sam Hacker posing as Amazon, sent me a fake public key, and that he is planning to decode my reply with his own private key and get my credit card number! In that case, we do the process in reverse. I ask Amazon to send me a message encoded with their private key. If I decode their message with their public key and it states, "Hi I am Amazon.com", I know that the message had to come from Amazon.com, because nobody else would have known how to encode it correctly. This is called authentication and is described in nice detail by VeriSign, the largest digital signature provider [30].

The students often wonder what prevents Sam Hacker from using Amazon's private signature. After all, everyone who decodes "Hi I am Amazon.Com" knows what those characters look like before they are decoded. Sam could decode their signature just like anyone does, and start signing messages with it! This is a serious problem. The solution is to run the whole message sent by Amazon through a *hash* function – and then encode the result of the hash function with Amazon's private key. (A hash function H is a function that takes a message M of any size and computes $H(M)$ a fixed size output, with the property that it is computationally infeasible to find another message N, where $H(N) = H(M)$.) Then all Sam could do is retransmit the very same message Amazon meant to send, but he could not send his own messages with Amazon's signature.

Another objection often arises in class: How do we know that the public key of Amazon is correct? Perhaps an adversary has published fake Amazon.com public keys all over the Internet. The solution is to have a company like VeriSign guarantee the authenticity of Amazon's public key by the same process. Of course someone could pretend they are VeriSign, but that is hard to do since VeriSign's raison d'etre is to make sure that nobody masquerades as them. Their business depends on it.

The Mathematics Behind Public Key Cryptography

How do we construct and use these private and public keys? Interestingly, the method is based on number theory, one of the oldest branches of pure mathematics, more famous for its beauty and elegance than its practical applications.

We start by describing a simple version of the Rivest, Shamir, and Adelman (RSA) algorithm that is *not* public-key cryptography because the private key can be computed from the public key using the Extended Euclid's Algorithm described earlier. This simpler version isolates the main ideas from the public-key part, and helps one better appreciate the contribution of RSA.

Public key cryptographic methods encode integers into integers, so we assume that our messages are first converted somehow to a sequence of integers. The exact method of conversion uses hash functions and is not trivial but that won't concern us here.

To encode a number, we will need the *public key*. This consists of two integers, for example 5 and 17. The second integer must be prime, and the first must be relatively prime to the second integer minus one. In this case, 17 is prime, and 5 is relatively prime to 16.

For example, to encode 6 using this key, we calculate 6^5 mod 17. You can check that this equals 7. To decode 7 back into 6, a brute force approach requires trying all possible values from 0 to 16 to see which one would encode into 6. This is computationally prohibitive when the prime has 40 or more digits. However, we can decode quickly if we calculate the *private key*. The private key also consists of two numbers, one of which is part of the public key, namely the prime 17, and one of which is private, in this case 13. To decode, we calculate 7^{13} mod 17, which you can verify equals 6.

How is the private key, 13 in our example, calculated? It is the solution to the equation $5u = 1 \bmod 16$. This solution can be computed efficiently with Euclid's Extended Algorithm, by finding u and v such that $5u + 16v = \gcd(5, 16) = 1$.

Why does the private key decode correctly? In our example, why does $6^{5(13)} \bmod 17 = 6 \bmod 17$? It all comes down to Fermat's Little Theorem. Fermat's Little Theorem implies that $6^{16} = 1 \bmod 17$. Since $5(13) = 1 \bmod 16$, we can write $5(13) = 16(4) + 1$. Thus $6^{5(13)} = 6^{16(4) + 1}$. Finally, since $6^{16} = 1 \bmod 17$, $6^{16(4) + 1} = 6 \bmod 17$.

In the Classroom**:** We have found that isolating the RSA idea from the part that requires the factoring enables students to more easily understand the algorithm. Furthermore, there is some historical justification for this pedagogy because the results were discovered in this layered way.

The RSA Algorithm

The real RSA algorithm is very similar to the algorithm described in the previous few paragraphs with one important difference: with the real RSA it is *not* easy to compute the private key from the public key.

This time we start with two prime numbers, p and q, say 2 and 17, and compute their product $pq = 34$. We calculate $(p–1)(q–1) = 16$, and then choose a value that has no common factors with 16, let's try 5. The public key becomes the pair of numbers 34 and 5.

The encoding and decoding is done just like before. For example, to encode 6, we compute $6^5 \bmod 34 = 24$, and to decode 24 we compute $24^{13} \bmod 34 = 6$. As before, 13 is the solution to the equation $5u = 1 \bmod 16$.

The difference between the real RSA idea and our first attempt is that previously 16 was calculated simply by subtracting 1 from the common public prime. The equation, $5u = 1 \bmod 16$, was then solved by Euclid's Extended Algorithm. But now the only way to calculate 16 is to factor the number 34 into 2 and 17 and compute $(2–1) \times (17–1)$. And the factoring part is hard! Nobody knows how to factor numbers quickly. The best methods are exponentially slower than the time complexity of the Extended Euclid's Algorithm.

Of course, anybody can factor 34, but in practice the two primes that are chosen for the public key are on the order of 100 digits each. This makes all currently known factoring algorithms take years. If you come up with an efficient algorithm that can factor numbers, you will be famous!

In the Classroom**:** Students write programs to encode integers using the RSA algorithm. As with the Vigenere cipher, they compete by trying to break each other's codes. The importance of choosing large enough prime numbers comes to life, as otherwise their messages are easily cracked. As the large prime factors defeat the cracking attempts, the students appreciate first hand just why it is safe to send a credit card number over the internet.

The Problem of Factoring

What makes the RSA algorithm a one-way, or *trapdoor* method, is that decoding requires factoring, and encoding does not. There is currently no better way to factor an integer n than to try all possible prime factors, up to $\sqrt{n}$. This is an exponential time algorithm. At first, a lot of students mistakenly think that this *is* a polynomial time algorithm, because the process takes about $\sqrt{n}$ operations for an integer n. However, computer scientists naturally measure time complexity as a function of the *size* of the input, and the *size* of an integer n is the number of bits it takes to store the integer, which is proportional to $\log n$. Letting $m = \log n$, the best factoring algorithm takes time proportional to $2^{m/2}$ operations.

Is there a faster way to factor numbers? There is, after all, no proof that factoring is inherently hard. The problem of factoring is not even known to be NP-Complete. Perhaps one day someone will come up with a polynomial time factoring algorithm.

In 2002, a trio of Indian computer scientists, Agrawal, Kayal, and Saxena invented a polynomial time algorithm [1] to determine whether or not a number is prime. Let *m* be the number of bits representing the number to be factored. Their algorithm runs in time proportional to m^{12}. This was a huge breakthrough, since up until then the best result was $m^{c \log\log m}$ discovered in 1983 [20]. This latter result is not polynomial in *m* because of the *log log m* factor in the exponent.

For determining whether or not an integer is prime, it is possible that the exponential time algorithm, $m^{c \log\log m}$, will run faster than the provably polynomial time algorithm, m^{12}, but that is a question of engineering. Even though *log log m* is theoretically not a constant, it is less than 4 for all input sizes currently being used for secure message transmission, and less than 10 for all input sizes ever likely to be used. A joke from Carl Pomerance [20] summarizes "that although it has been *proved* that *log log m* tends to infinity with *m*, it has never been observed doing so". Nevertheless, from the complexity theorists' point of view, the polynomial time breakthrough of Agrawal, Kayal, and Saxena [1] is a milestone in the way the 4-minute mile was. It opens possibilities, debunks impossible barriers, and confirms what most thought would be true and what some suspected would not.

This breakthrough in the complexity for determining whether a number is prime or composite is reminiscent historically of the 1980s breakthrough in linear programming. Linear programming is an optimization problem that is used in all sorts of practical situations, such as determining the best way to schedule airline flights so as to minimize costs and maximize profits. The simplex algorithm of Dantzig (1947) had been used for years to successfully solve linear programming problems, despite its theoretical worst-case exponential time complexity. In practice, the exponential time behavior was not observed, although theoretically it was not clear why this was the case. Then in 1979 [17], a polynomial time algorithm was invented. It took another breakthrough in 1984 [15] and years of software engineering and testing before this polynomial time algorithm was competitive with the exponential time simplex algorithm. Complexity theory, a 20th century field of study, has a long way to go before it completely models practical computer science problems.

> In the Classroom**:** This historical parallel between linear programming and factoring makes an impression on the students. They are often suspicious of theory and its practical uses. Although there are plenty of examples to show them the applicability of theory in computer science, here we show them examples where the theory is weak! They learn that the critique of theory in science is a historical process that forces the theory to improve.

Can this new breakthrough for determining whether or not a number is prime be used to help factor numbers quickly? Not yet as far as anyone knows, and most people suspect it cannot. However, in many cases, including the well-known Traveling Salesman Problem, the *decision* version of a problem is polynomially related to the *optimization* version. [10]. That is, if you can decide whether or not something is possible, then you can also figure how to do it. If this were the case for factoring, then a polynomial time algorithm for deciding whether or not a number is prime would hint at a polynomial time algorithm for determining the factors.

Historical Notes on RSA Encryption

Ron Rivest [22] was kind enough to share with me his own recollections of the RSA discovery and his thoughts about the future directions of cryptography. Rivest credits the original motivation for the RSA work to the 1976 seminal ground-breaking paper **New Directions in Cryptography** by Diffie and Hellman [7]. The paper explains how two users can exchange a secret key over an insecure medium without any prior secrets. The challenge of public-key cryptography was directly proposed in this paper, and was brought to the attention of Rivest by a graduate student. Rivest, Shamir, and Adelman were assistant professors at MIT at the time, Rivest in computer science, Shamir and Adelman in mathematics. The departments in MIT overlap through the many independent research labs, and the three colleagues collaborated on this problem at LCS, the lab for computer science.

The three tried a number of unsuccessful approaches, including a method using the *knapsack* problem as a way to thwart the *bad guys*. I suggested that it was ironic that the approach using knapsack, a known NP-Complete problem, failed, while the approach using factoring, a problem not known to be NP-Complete, succeeded. Rivest responded that NP-complete problems often have many different solutions, and for the purposes of public key cryptography, it is easier to work with factoring which has a unique solution.

Rivest is not currently working on the complexity of factoring, although he continues to work on new applications in cryptography related to internet voting and to radio frequency ID tags. When I asked him whether he believed that factoring was provably hard, he responded "I have a built-in bias in favor of hoping that factoring is hard, but to prove lower bounds for factoring – we'll need something a bit different than NP-completeness." Rivest does not expect that the new polynomial decision algorithm for prime numbers will yield a better factoring algorithm, but he is willing to be surprised. "I'm always in favor of the truth."

Did Rivest realize the broad applications and notoriety that their work would eventually yield? Rivest explained that RSA was discovered long before the days of the internet and just before the personal computer age. He had no inkling of the eventual supporting and crucial role his research would have on e-commerce. "It wasn't obvious, a priori, that our result would stand the test of time." Theoretically, there could be advances in factoring methods. Practically, the applications might not find enough uses.

After PCs came on the scene, Rivest et al started a company in 1983, called RSA Data Security, in an attempt to find commercial uses for the theory. By 1986, "we thought we would go bust", but then Lotus Notes signed a contract. The 1990s introduced the world-wide-web and RSA became ubiquitous. VeriSign was spun-off, in order to maintain its independence from the technology, and RSA Data Security was bought and renamed simply RSA Security (http://www.rsasecurity.com/).

Commercial success notwithstanding, Rivest seems most pleased with how cryptography as a field has evolved. "The rich interplay between theory and practice brought vitality to the subject." It has also brought controversy.

The National Security Agency (NSA) is a government agency with a heavy investment in secret cryptographic research. They do not publish their results, nor commercialize their implementations, but they do monitor what others are doing. At one time they threatened that the planned delivery of a paper related to RSA, at a Cornell IEEE conference, would violate the Export Control Act! The subject matter they claimed is classified. Eventually and sensibly, the government backed down. Rivest pointed out the absurdity of classifying research when one or more of the authors has no security clearance themselves. Today the NSA allows academic freedom in cryptographic research but still insists that software companies register their cryptographic products with the Bureau of Industry and Security (formerly the Bureau of Export Administration); see http://www.bxa.doc.gov/Encryption/Default.htm. The role of the NSA in monitoring cryptographic research and its commercial development will no doubt continue to evolve.

In the Classroom**:** The students learn where the history of this research is heading and what problems are still open. Examples of problems where the decision and optimization versions are *not* polynomially related is rare. It seems that this *may* be the case for factoring. When will we know?

Research on new cryptographic methods continues simultaneously with research on breaking the current codes. There are long-term strategies, like quantum computing and DNA computing, and short-term strategies using conventional computing. In all cases, mathematics will no doubt play a central role. In the meantime, e-commerce is still on secure ground.

In the Classroom**:** Quantum computing and DNA computing are very new fields that try to deal with the fundamental intractability that is inherent in hundreds of NP-Complete problems. They represent two parallel paths of future work that may or may not bear fruit. They are difficult fields but could be used to design labs, although we have not yet attempted to do this. The interested reader should consult [9].

Cryptographic Decoding Challenges for Practice and Review

In the Classroom**:** In our labs, we not only have the students compete against each other by designing and cracking each other's codes, but also by seeing who can first decode messages that we created. There are usually cash prizes where the amount is proportional to the level of difficulty. The Vigenere ciphers pay more when we do not tell the students the length of the codeword. Here are some examples:

Vigenere Ciphers I
A two-cycle Vigenere encryption gives an encoding of a well-known cerebral song:
BQHIERPVBZXOPORHASACNFLQHBYSKFBBZKBHAHASYZHKXFLQHBLIEHBBZKB
HAHASKOBB
PWMVMVXHACNUAHLWWPXHAWGYBBBQHIERUSTBHHASKZBBVCEBBTBCGZRV
TRTPKOBB. What is the original text?

Vigenere Ciphers II
A five cycle Vigenere encryption gives an encoding of a Woody Allen math joke:
BOQWMNWOOQSSFHQKASRLAGOWRAADWRKAONCIOHKJKCYHVYWHLLC.
What is the original text?

RSA Ciphers I
An RSA encoding of nine ASCII values, using public key (10555, 21971), gives the name of a hunter: 16912 19531 20676 16912 6613 2348 17835 15770 15770. What is the original text?

RSA Ciphers II
An RSA encoding using public key (5555551, 118513313) gives: 80217189 107242213 96490860 79543571 25953566. What are the original numbers?

Conclusion

Cryptography is an ancient yet vibrantly active field for mathematicians and computer scientists. It is fun to teach because it contains a beautiful combination of elegant theory and practical application, and lends itself to exploration and learning through programming and experiments. The history of cryptography, while providing stories of intrigue and excitement, is a mathematical metaphor for the delicate balance between theory and practice.

Acknowledgments: Thanks to Ralph Bravaco, Tara Holm, Andrea Simonson, and Amy Shell-Gellasch for their suggestions on early versions of this article. A special thanks is due to Ron Rivest for sharing his personal thoughts and memories.

References

1. Agrawal, M., Kayal, N. and Saxena, N., *PRIMES in P*, August 2002. http://www.cse.iitk.ac.in/users/manindra/primality.ps http://www.cse.iitk.ac.in/news/primality.pdf
2. Albers, Donald J. and Reid, Constance, "An Interview with George B. Dantzig: The Father of Linear Programming," *College Mathematics Journal*, 17:4, (1986) 293-313.
3. Ball, W.W. Rouse and Coxeter, H.S.M., Mathematical Recreations and Essays, Dover, 1987.
4. Bogolmony, Alex, Cut-the-Knot, http://www.cut-the-knot.com/water.shtml.
5. Bravaco, R., Simonson, S., "Mathematics and Computer Science: Exploring a Symbiotic Relationship," to appear in *Mathematics and Computer Education*, 2004.
6. Carlson, Andy, Enigma Simulator, http://homepages.tesco.net/~andycarlson/enigma/enigma_j.html
7. Diffie, W. and Hellman, M.E., "New Directions in Cryptography," *IEEE Trans. Information Theory*, IT-22, 6, (1976) 644-654, http://citeseer.nj.nec.com/diffie76new.html.
8. Friedman, W., *The Index of Coincidence and Its Applications in Cryptography,* Publication No. 22, Geneva, IL, Riverbank Publications, 1922.
9. Gramss, T., Grob, M., Mitchell, M., et al., Eds. *Non-Standard Computation: Molecular Computation - Cellular Automata - Evolutionary Algorithms - Quantum Computers,* John Wiley & Sons, 1998.
10. Garey and Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman and Co., San Francisco, 1979.
11. Hodges, Andrew, *Alan Turing - The Enigma*, Walker and Company, New York, 2000.
12. The Imperial War Museum Online Exhibitions, *The story of Alan Turing and Bletchley Park*: http://www.iwm.org.uk/online/enigma/eni-intro.htm
13. Joyce, David, Euclid's Elements: http://aleph0.clarku.edu/~djoyce/java/elements/elements.html
14. Kahn, D., *Codebreakers: The Story of Secret Writing,* Macmillan, 1967.
15. Karmarkar, N., "A New Polynomial-time Algorithm for Linear Programming," *Combinatorica*, 4, (1984) 373-395.
16. Katz, Victor, *A History of Mathematics*, Harper Collins, 1993.
17. Khachian, L., "A Polynomial Algorithm in Linear Programming," Soviet Math. Dokl., 20, (1979) 191-194.
18. Menezes, Alfred J., van Oorschot, Paul C., Vanstone, Scott A., Handbook of Applied Cryptography, CRC Press, 2001.
19. O'Connor, J. J. and Robertson, E. F., *Biography of Pierre de Fermat*: http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Fermat.html

20. Pomerance, Carl, *A New Primal Screen*, FOCUS, Vol. 22, No. 8, The Newsletter of the MAA, (2002) 4.
21. Rivest, Ron, Shamir, Adi and Adleman, Len, *A method for obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, 21(2), (1978) 120-126.
22. Rivest, Ron, personal communication.
23. Rosen, Kenneth H., Discrete Mathematics and its Applications, McGraw Hill, 4th edition, (1999) 206.
24. Simonson, Shai, Cryptography Labs for Mathematical Experiments in Computer Science – a Learning Community Course Exploring Discrete Math and Data Structures, http://www.stonehill.edu/compsci/LC/Cryptography.html
25. Singh, Simon, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor Books, 2000.
26. Singh, Simon, Cryptography Links by Simon Singh, http://www.simonsingh.net/owtasite/Crypto_Links.html
27. Singmaster, David, personal communication.
28. Sussman, Gerry, personal communication.
29. Tweedie, M.C.K., *A Graphical method of Solving Tartaglian Measuring Puzzles,* Mathematical Gazette, 23, (1939) 278-282.
30. VeriSign, Authentication, http://www.verisign.com/docs/pk_intro.html.
31. White, Andrew Dickson, A History of the Warfare of Science with Theology in Christendom, D. Appleton and Co., 1896, (reprinted in the public domain at: http://www.santafe.edu/~shalizi/White/).