

Good Old Mathematics

The Basis of Cryptography

Mike Knee

Snell & Wilcox, UK

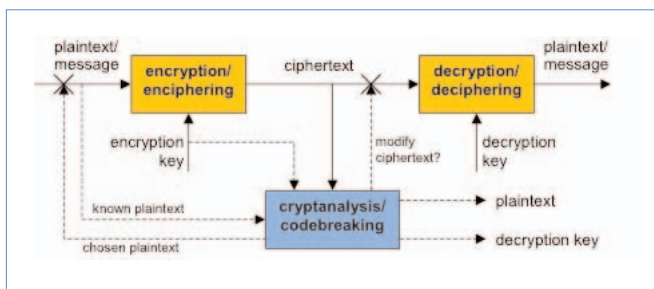
Abstract: This tutorial paper gives a fairly gentle introduction to the basic mathematics required to understand encryption systems. Three main areas are covered: modular arithmetic, prime numbers and probability theory. These should help the reader to understand how conventional and public-key encryption systems work, how they might be attacked and how to improve their security.

Introduction

This paper introduces the basic mathematics required to understand encryption systems. The science of *cryptology* (consisting of *cryptography*, the study of the design of cipher systems, and *cryptanalysis*, the study of how to break them) is at least 2,000 years old. Modern computing has made it possible to encipher huge amounts of data very securely in a reasonable time, while also making it easier for the cryptanalyst to break cryptographic systems. Cryptology is a fast-moving subject, not least in the exciting area of quantum cryptography where the first computer network secured by quantum techniques has been unveiled this year. It might be thought that the underlying maths would have to move equally fast. Happily for me, that is not the case. Modular arithmetic was formalized by Gauss at the end of the 18th century. The study of prime numbers and factorization has been around for at least 2,200 years, with the extra results used in public-key cryptography being provided (relatively speaking) in the nick of time (by Fermat in the 17th century and Euler in the 18th). Fermat and Pascal gave us the foundations of probability theory in the 17th century. Perhaps the most recent major mathematical contribution, especially important in quantum cryptography, is Claude Shannon's *Mathematical Theory of Communication* from 1948. But most of the contents of this paper are at least 200 years old!

The following diagram should help to define most of the cryptological terms and scenarios encountered in this paper. The main information paths are shown as bold lines, while the dotted lines show additional kinds of attack that might be available to the cryptanalyst.

Figure 1: Cryptography and cryptanalysis



Modular Arithmetic

Modular arithmetic is also known as “clock arithmetic” but might better have been called “calendar arithmetic” as it has been around for at least as long as we have had calendars. It is what we naturally do when we calculate with days of the week. When we work out that three days from today (Friday) is a Monday and that six days ago it was a Saturday, we are performing *modulo-7* addition and subtraction. We do modulo-

12 arithmetic when counting months and when doing key changes in written music, and modulo-360 arithmetic when calculating angles. To do modular arithmetic, you simply do normal arithmetic, divide the result by the chosen *modulus* (7, 12, 360 etc.) and take the remainder. Alternatively (and this can be more convenient), do normal arithmetic and, if necessary, keep subtracting or adding the modulus until the result is positive and is less than the modulus. Note that in both cases the result is between 0 and one less than the modulus.

For example, in modulo-7 arithmetic, to calculate $6 + 5 + 4$, do the normal sum to get 15. Divide by 7 to get 2 remainder 1, or subtract 7 to get 8 and subtract 7 again to get 1. Either way, the answer is 1.

Modular addition and subtraction

An early use of modular arithmetic in encryption was the *Caesar Cipher*, in which each letter of a message is encrypted by replacing it with the letter three places further along in the alphabet. So “IBC” would be encrypted as “LEF”.

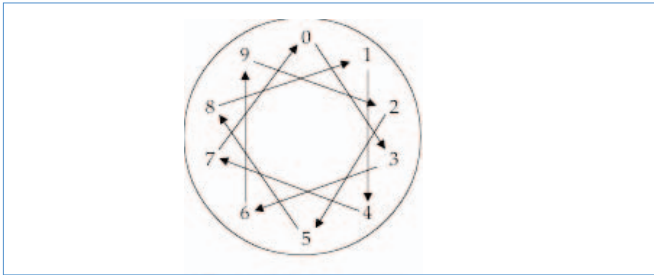
To make this process look more “mathematical” we could represent each letter by a number between 0 and 25 (starting from 0 might seem to be for the sake of computer programmers, who don’t know how to count, but it also helps when it comes to modular arithmetic). The encryption key would be denoted +3, and the operation of encrypting a letter is then to add the key to the letter’s corresponding number. For example, to encrypt E=5, we add 3 to get 8=H.

What happens to X, Y and Z? Y=24, for example, would become 27, for which we have no letter. It is fairly obvious in the Caesar cipher that X, Y and Z will be encrypted as A, B and C respectively, so we need a kind of arithmetic in which 26, 27 and 28 become 0, 1 and 2. This is modulo-26 arithmetic. To perform it, we can use the rule that we do normal arithmetic but subtract 26 whenever the result is 26 or greater.

Decryption in the Caesar cipher would be done by subtracting 3, and in this case we have to be careful about negative numbers. If our answer is negative, we should *add 26*. This may seem obvious, but a horrendous fact is that many computer languages do not do modular arithmetic correctly when negative numbers are involved! Of course, in this case we could also decrypt by *adding 23*.

Figure 2 illustrates another example of addition in modular arithmetic. In this example, I am working with numbers modulo-10, partly because I am too lazy to write out 26 symbols, and partly because I would like to underline the number theory basis of modern encryption.

Figure 2: Adding 3 modulo-10



Modulo-10 arithmetic would delight schoolchildren, because it's the same as ordinary arithmetic but without having to remember to carry anything over.

Modular arithmetic as a one-to-one mapping

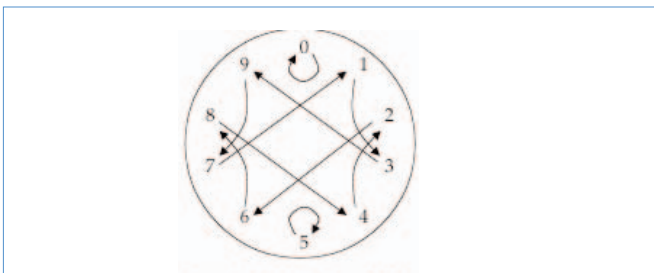
The "XYZ" problem above gives us a hint as to why modular arithmetic is so important in encryption. As a general rule, we do not want encryption to have a big effect on the size of a message. Making a message smaller is the domain of compression, while making it bigger is the domain of error protection. In encryption, we would like a *reversible mapping* between the message (plaintext) and its encrypted version (ciphertext) and we don't want to add too much redundant information. When we come to multiplication and exponentiation using integers, the problem with ordinary arithmetic is that the range of outputs from fairly small input numbers can be huge. The nice thing about modular arithmetic is that everything is kept within the range of the chosen modulus.

Modular multiplication

Again, modular *multiplication* is just like ordinary multiplication with the added step of dividing the result by the modulus and taking the remainder. For example, in modulo-7 arithmetic, $3 \times 6 = 4$, because 4 is the remainder on dividing 18 by 7.

Modular multiplication has a pleasing symmetry about it. For example, figure 3 illustrates multiplication by 3 in modulo-10 arithmetic.

Figure 3: Multiplying by 3 modulo-10

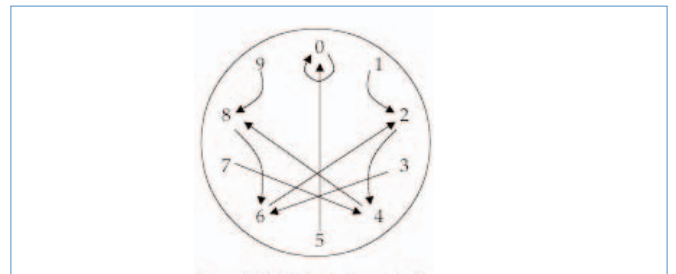


Modular multiplication could be the basis of a slightly more sophisticated version of the Caesar cipher in which we modulo-26 multiply the numbers by a key. But we have to be careful when we consider how this is decoded. Just as the inverse of modular addition is modular subtraction, so the inverse of modular multiplication is modular division. Unfortunately, modular division does not always work, as we shall see next.

Modular division

Look at figure 4, which shows modulo-10 multiplication by 2. If we trace the arrows in the opposite direction, we have the inverse operation, which is *division* by 2. But we have a problem. Dividing an even number by 2 has two possible results, while dividing an odd number by 2 isn't possible. This does not fit our need for a one-to-one mapping. We need a unique result for every division, and we don't want to start introducing fractions to deal with the odd numbers. It turns out that we don't get this problem if the divisor and the modulus are *relatively prime* (this is defined later), for example in figure 3. It's even better if the modulus is prime, because then the condition will be met for all divisors. When we meet this condition, we can do modular division. This is one reason why modular arithmetic for encryption is done using prime or relatively prime numbers; more on this in the next chapter.

Figure 4: Multiplying by 2 modulo-10



Modular exponentiation

Modular *exponentiation* - taking a number to a power - is done by repeated multiplication, just as in normal arithmetic. Modular exponentiation is important in encryption because in the right conditions it is a good example of a supposed *one-way function*: fairly easy to do, but very hard to undo. There are a couple of tricks which make modular exponentiation "fairly easy". These are best described by an example. Suppose we wish to calculate 7^{18} in modulo-31 arithmetic.

The "brute force" approach would be to start with 1 and multiply by 7 eighteen times. The answer in normal arithmetic would be 1,628,413,597,910,449, which when divided by 31 gives a remainder of 2, the answer. It is a shame to have to deal with such large numbers in modulo-31 arithmetic!

A slightly better approach would be to take the remainders at intermediate stages in the repeated multiplication. The first few steps would be:

$1 \times 7 = 7$
 $7 \times 7 = 49$, remainder = 18
 $18 \times 7 = 126$, remainder = 2
 $2 \times 7 = 14$
 $14 \times 7 = 98$, remainder = 5
 $5 \times 7 = 34$, remainder = 3
 etc.

We have removed the need to handle large numbers but there are still many multiplications to do.

An even better approach is to recognize that $7^{18} = 7^{16+2} = 7^{16} \times 7^2$. These partial powers of 16 and 2 can be calculated easily by repeated squaring, and as usual we can deal with the remainder whenever we wish. $7^2 = 49$ which has a remainder of 18. To calculate 7^{16} , we square this result three times more, so:

$18^2 = 324$, remainder = 14
 $14^2 = 196$, remainder = 10
 $10^2 = 100$, remainder = 7

Finally, we multiply the two partial powers together:

$18 \times 7 = 126$, remainder = 2

So that was modular arithmetic, a powerful way of manipulating numbers within a given range in a reversible way. It is one of the basic tools of many encryption systems and we shall encounter it again. There remains in this chapter an important special case: *modulo-2 arithmetic*.

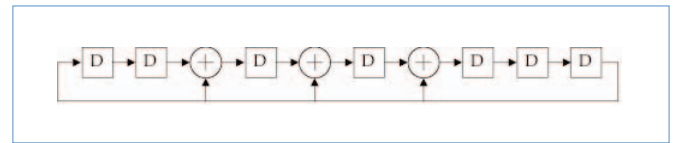
Modulo-2 arithmetic and Galois fields

This is the simplest possible kind of modular arithmetic but turns out to be remarkably powerful in encryption applications. We have only two numbers to deal with, 0 and 1, and the addition and multiplication rules are:

$0 + 0 = 0$	$0 \times 0 = 0$
$0 + 1 = 1$	$0 \times 1 = 0$
$1 + 0 = 1$	$1 \times 0 = 0$
$1 + 1 = 0$	$1 \times 1 = 1$

So modulo-2 addition is nothing more nor less than the Boolean *exclusive-OR* function, and modulo-2 multiplication is simply an AND function. Why is it so useful? Here is one example: a *linear feedback shift register*. Figure 5 shows a machine which uses seven delays and some modulo-2 adders to produce a long pseudo-random sequence of bits.

Figure 5: Linear feedback shift register



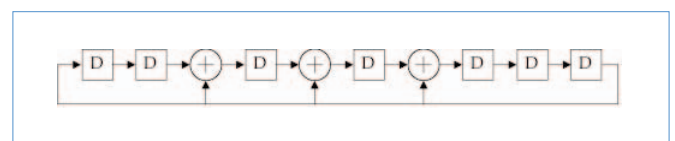
Careful choice of the arrangement of adders will ensure that the machine generates a *maximal-length* sequence, in this case $2^7 - 1 = 127$ bits. Each time the shift register is clocked within this period, it will contain a unique seven-bit word. (Exercise for the reader: can you design a shift register to give a maximal-length sequence as above but using only one adder?) Here we are in the realm of *Galois field theory*. The modulo-2 arithmetic system is known as a *Galois field* of order 2, or $GF(2)$. We can define polynomials with this arithmetic, and these can be mapped onto the shift register architecture. In the example of figure 5, the equivalent *generator polynomial* is $1 + X^2 + X^3 + X^4 + X^7$, and the shift register is actually doing arithmetic with polynomials of order 7, known as $GF(2^7)$, modulo the generator polynomial. It turns out that if the generator polynomial is irreducible (the equivalent of “prime” for polynomials) then the resulting sequence will be maximal-length.

The bit sequence at the output will be in many senses statistically random or noise-like, a useful feature in many encryption systems. But there is a danger in relying on the apparent randomness of such sequences. Suppose we tried to use the long sequence as a *one-time pad* (described later). The ciphertext would look pretty well scrambled, but in fact it would only be necessary for a cryptanalyst to recover seven consecutive bits of the one-time pad to determine all of it and break the code.

Cyclic redundancy checks

Linear-feedback shift registers are important not only in pure encryption but also in *message authentication*. An important aspect of secure communication is the assurance that the message has not been tampered with in some way. This can be tackled partially within the cipher itself by ensuring that it has an element of *diffusion* (a property whereby changing a small part of the ciphertext affects a large amount of plaintext). Another approach is to use techniques borrowed from error detection, such as the *Cyclic Redundancy Check* (CRC). A CRC can be thought of as nothing more than a linear feedback shift register with an additional input for the message to be checked, as shown here.

Figure 6: Simple cyclic redundancy check



The contents of the shift register after the message has been passed through are a 7-bit *checksum* which is appended to the message. An equivalent checksum is generated at the receiving end. If the message is corrupted, it is very likely that the received and locally generated checksums will disagree.

We now turn to the second big area of mathematics that is useful in cryptography - prime numbers

Prime Numbers

This section discusses the mathematics of *prime numbers*, a hugely important part of number theory which has exercised mathematicians for centuries, with no idea that their work would one day have practical applications, for example in enabling us to buy pure maths books securely over the Internet.

A few definitions

A positive integer greater than 1 is *prime* if its only divisors are 1 and itself. Such a simple definition, but so much difficult maths follows on from it. It is difficult to count prime numbers, to find out if a given large number is prime, and to generate prime numbers. Nobody knows a simple formula that will generate *all* the prime numbers, and there are some absurdly simple statements about primes which have never been proved, the most famous being *Goldbach's conjecture* which states that every even number greater than 2 can be expressed as the sum of two primes. Thankfully, it *is* easy to prove that we will never run out of primes (but once again Euclid beat us to it), and this is just as well, as a never-ending supply seems to be essential for encryption.

Primes are the building blocks of all numbers. Every number greater than 1 can be written as the product of prime numbers, and the *Fundamental Theorem of Arithmetic* tells us that this *prime factorization* of a given number is unique - there isn't another set of primes that can be multiplied together to get the same result.

The *highest common factor (hcf)*, or *greatest common divisor*, of two numbers hardly needs further definition. For example, the hcf of 30 and 18 is 6; no higher number will divide exactly into the two numbers. A 2,200-year-old algorithm from Euclid enables us to calculate the hcf quite easily - you simply take remainders when dividing one number by the other, retaining the two smallest numbers as you go, and stopping just before you get to 0. For example:

$$30/18, \text{ remainder} = 12$$

$$18/12, \text{ remainder} = 6$$

$$12/6, \text{ remainder} = 0, \text{ so answer} = 6.$$

If you have the prime factors of the numbers, you can also easily calculate the hcf by multiplying all the prime factors that the numbers have in common. For example:

$$30 = 2 \times 3 \times 5$$

$$18 = 2 \times 3 \times 3$$

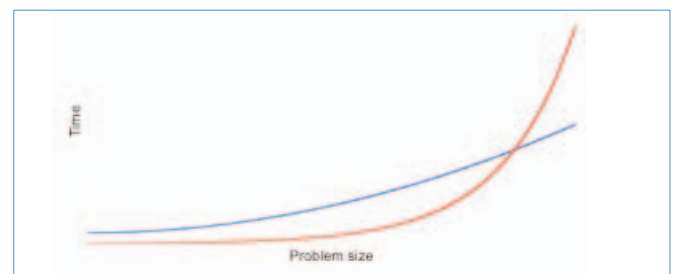
$$\text{Common prime factors: } 2 \times 3 = 6$$

Another definition: two numbers are *relatively prime* if their hcf is 1 - so they have no prime factors in common.

Difficult calculations

A cornerstone of modern public-key encryption is the (unproved) supposition that it is very difficult to find the prime factors of very large numbers. What do we mean by such imprecise phrases as "very difficult" and "very large"? It's not absolute size or difficulty that matters, but how fast the difficulty increases as the size of the numbers increases. Some calculations can be performed in *polynomial time*. This means that the computation time can be expressed as a polynomial function (a sum of finite powers) of the "size" of the problem (typically the number of digits in the input). Other calculations (our "very difficult" ones) are thought not to be solvable in polynomial time, even if the polynomial involves huge powers. Some of these are *exponential-time* problems. For some sizes of numbers, they may be easier than some polynomial-time problems, but eventually as the size of the problem increases (the numbers become "very large") the exponential-time problem will become slower than the polynomial-time problem. Figure 7 gives an example of how an exponential-time problem can always be made slower than a polynomial-time problem if the numbers get big enough.

Figure 7: Polynomial and exponential time



We would like the difficulty of cracking encryption systems to at least keep up with *Moore's law*, which is the observation that computing power is increasing exponentially with time. If an encryption system could be broken in polynomial time, no matter how difficult it may seem to be at the moment, Moore's law would eventually catch up with it unless we increased the sizes of the numbers involved by ridiculous amounts.

Fermat and Euler

Between them and unwittingly, these two gentlemen laid the foundations for modern public-key encryption. A very important theorem is *Euler's theorem*, which is a bit difficult to explain here. I shall just give two corollaries of the theorem, one of which was actually proved a century or so earlier than Euler's theorem.

Fermat's Little Theorem says that in modulo- p arithmetic, if p is a prime number and a is non-zero, then $a^{p-1} = 1$. For example, $4^6 = 4,096$ which is 1 in modulo-7 arithmetic.

Euler's Corollary (to his Theorem) says that if a is relatively prime to two primes p and q , then in modulo- pq arithmetic, $a^{(p-1)(q-1)} = 1$. For example, $4^{2 \times 6} = 16,777,216$ which is 1 in modulo- $(3 \times 7 = 21)$ arithmetic.

Public Key Encryption

We are now in a position to see how the maths we have looked at so far can be used in a *public-key* encryption system. The goal of a public-key system is to find a pair of functions, represented by numerical keys, that are inverses of each other and where the second function (decryption) cannot easily be deduced from the first function. The most celebrated and practically useful public key encryption system so far developed is the *RSA algorithm*, named after its inventors Rivest, Shamir and Adelman.

The RSA algorithm

The RSA algorithm works in modulo- pq arithmetic, where p and q are large prime numbers. If Alice is sending a message to Bob (they're nearly always called Alice and Bob), Alice uses Bob's public key e (and pq , which is actually part of Bob's public key) to encrypt her message which has been expressed as a number $m < pq$. The encryption is very simple; it's just raising m to the power e (modulo pq , of course). When Bob receives the message m^e , he raises it to the power of his private decryption key d to obtain a result m^{ed} .

In order for this to work as an encryption system, two conditions have to be met. First, it must be "very difficult" to calculate the private decryption key d from knowledge of the public encryption key e and pq . Second, the decrypted result m^{ed} must of course be equal to the original message m . How is this ensured?

The public key e is selected so that it is relatively prime to $(p-1)(q-1)$. This makes it possible to find a private key d which is the "reciprocal" of e , i.e. $ed = 1$ in modulo- $(p-1)(q-1)$ arithmetic. The actual calculation of this reciprocal can be done using a version of Euclid's algorithm, presented above, and is easy provided that p and q are known. However, it is

very difficult if only the product pq is known; this number would first have to be factorized and we are working on the assumption that it is very difficult to factorize large numbers. So we have met our first condition. It remains to show that $m^{ed} = m$. Choosing d as the reciprocal of e means that there must be an integer t with $ed = 1 + t(p-1)(q-1)$ (in normal arithmetic). So $med = m^{1+t(p-1)(q-1)} = m(m^{(p-1)(q-1)})^t$

Now in modulo- p arithmetic, Fermat's Little Theorem says that $(m^t)^{(p-1)(q-1)} = 1$, so $m(m^{(p-1)(q-1)})^t = m$. This will also be true in modulo- q arithmetic, and it is then not hard to show that it will be true in modulo- pq arithmetic. So we have shown that decryption is the inverse of encryption.

Here's a simple example using ridiculously small numbers. Suppose $p = 5$, $q = 11$, so $pq = 55$. We just have to pretend that 55 is hard to factorize! Then $(p-1)(q-1) = 40$. The public key e must be selected to be relatively prime to 40, for example 27. Then the private key d will be equal to 3 (because $27 \times 3 = 81$ which gives a remainder of 1 when divided by 40). Suppose the message is represented by a number $m = 6$. By the repeated squaring process, we can encrypt this "message" in our modulo-55 arithmetic:

$$6^2 = 36$$

$$6^4 = 36^2 = 1,296, \text{ remainder} = 31$$

$$6^8 = 31^2 = 961, \text{ remainder} = 26$$

$$6^{16} = 26^2 = 676, \text{ remainder} = 16$$

So $6^{27} = 6^{16+8+2+1} = ((16 \times 26) \times 36) \times 6 = (31 \times 36) \times 6 = 16 \times 6 = 41$, taking remainders whenever we need to. So the ciphertext is the number 41, and we decrypt it by raising it to the power of 3, again in modulo-55 arithmetic: $(41 \times 41) \times 41 = 31 \times 41 = 6$. Our original message!

Signature authentication

One weakness of a public-key system is that there is nothing to stop an impostor from claiming to be Alice. Happily, Alice can use the RSA algorithm in reverse to provide *authentication* of her identity. This is because of the interchangeability of the encryption and decryption processes; in the above example, d and e can be exchanged and the overall result will be the same. So to provide a signature, all Alice has to do is to encrypt her name using her *private* key and attach it to her message. Then Bob decrypts the attachment using Alice's *public* key and recovers Alice's name. Nobody else could fake the encrypted signature because it has been produced using Alice's private key, known only to Alice.

Diffie-Hellman key exchange

This is another approach to public-key encryption, which differs from RSA because the encryption and decryption keys are identical, and in fact any *symmetric* encryption algorithm (having the same keys for encryption and decryption) can be used. It is also much easier to understand than RSA!

Instead of looking up public keys in a directory, Alice and Bob establish a trusted (but not necessarily secret) communication channel. They choose a prime modulus p for their arithmetic and an arbitrary number $a < p$. Alice then chooses a secret number e and transmits $a^e \pmod{p}$ to Bob. Given this information, an eavesdropper would find the value of e “very difficult” to calculate. Meanwhile, Bob also chooses a secret number d and transmits $a^d \pmod{p}$ to Alice. Likewise, an eavesdropper would find d “very difficult” to calculate.

Alice and Bob then each take what they have received to the power of their own secret number. So Alice calculates $(a^d)^e$ and Bob calculates $(a^e)^d$. Both these numbers are equal to a^{ed} and this common number is used as the encryption and decryption key.

A simple example: $p = 37$, $a = 12$. Alice chooses $e = 17$ and sends $12^{17} = 34$. Bob chooses $d = 14$ and sends $12^{14} = 7$. Alice calculates the key, $7^{17} = 16$. And Bob calculates the key, $34^{14} = 16$. They both have the key without having exchanged it!

We now turn from the very precise field of number theory to the somewhat woollier field of probability theory.

Probability Theory

This is another old, but perhaps more familiar, area of mathematics which comes into play in cryptology, particularly in cryptanalysis. One important point to make about probability with regard to encryption is that it is really all about counting how many times things happen, and need not be any more complicated than that.

A few basics

The notion of *probability* is actually quite hard to define rigorously, but we do not need that level of rigour for the purposes of this paper. We can stick to the “obvious” notion that the probability of an outcome can be expressed as the average number of times an event will have that outcome, expressed as a fraction of the total number of events. For example, with a fair dice we should find that $1/6$ of the throws will produce a 5, so we say that the probability of throwing a 5 is $1/6$. It should go without saying that the measured fraction should approach $1/6$ more closely as we increase the number of throws.

We can manipulate probabilities with arithmetic but we have to be careful. We can *add* probabilities together, but only if the corresponding outcomes are *mutually exclusive*. For example, the probability of throwing 1 or 2 with a dice is $1/6 + 1/6 = 1/3$. We can add the probabilities of throwing a 1 and of throwing a 2 because the two outcomes cannot both occur together. But suppose we have two dice and want to calculate the probability of throwing a 5 on at least one of them. We can't just add the two $1/6$ s together, because throwing a 5 on one dice does not stop us throwing a 5 on the other.

Probabilities can also be *subtracted* with the same care. Subtraction of probabilities most often occurs when we want to calculate the probability of something *not* happening. For example, the probability of *not* throwing a 5 on one dice is $1 - 1/6 = 5/6$. This is not hard to see if you rearrange the equation to turn it into an addition and if you note that the probability of “an outcome of any kind” is 1.

Likewise, we can *multiply* two probabilities together to find the probability of two outcomes both occurring, but only if the outcomes are completely *independent* of each other. For example, the probability of throwing a 5 on *both* dice is $1/6 \times 1/6 = 1/36$, because the two throws are independent (and note that this is true regardless of whether the dice are thrown together or one after the other).

We can now solve the problem of throwing a 5 on at least one of two dice by standing it on its head. Throwing a 5 on at least one of two dice is exactly opposite to “not throwing a 5” on *both* dice. The probability of not throwing a 5 with one dice is $5/6$, so for both dice it is $5/6 \times 5/6 = 25/36$ - we can do the multiplication because the two dice throws are independent. Then, the answer to our original question is $1 - 25/36 = 11/36$.

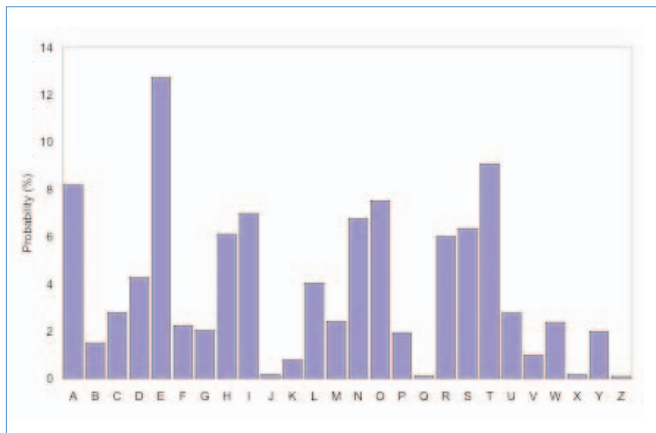
The last basic concept is that of *randomness*. This has various definitions, but I shall use the most common one, that *random numbers* have uniform probability distributions and are completely independent of one another.

We now look at why these notions of probability are important in cryptology, starting with a look at the distribution of letters in the alphabet.

Letter distributions

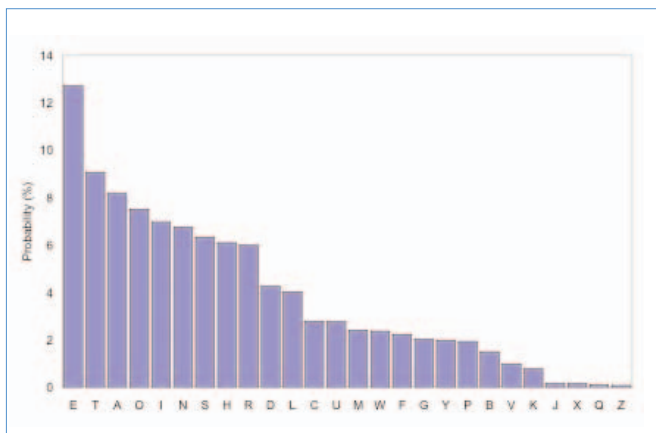
You don't need to look at much English text to see that the letter e occurs often and the letter z occurs far less often. The following histogram shows the approximate probabilities of the 26 letters of the alphabet.

Figure 8: Histogram of English letters



It is useful to order this histogram by probability, starting with the most common letter, as shown here.

Figure 9: Ordered histogram of English letters



Now suppose we receive a message that has been encrypted using a *substitution cipher*, in which every letter has been replaced by another letter, the same one each time it occurs. The Caesar cipher is a simple example of a substitution cipher, but any one-to-one mapping of the alphabet can be used provided information about the mapping can be transmitted to the intended recipient (this becomes the key). An eavesdropper could make an ordered histogram of the letters in the ciphertext. If she is lucky, she should obtain something with a similar shape to Figure 9, but with different letters. To some degree of accuracy, she should be able to match the letters one by one according to their relative frequency of occurrence, and break the cipher.

How can the cryptographer get over this problem and transmit messages more securely? There are two (overlapping) approaches: design more secure algorithms, or change the message.

More secure algorithms

One simple approach to improving security is to make sure the eavesdropper doesn't receive enough information to make a reliable histogram. Unfortunately, this means that the key has to be changed more often, which brings about its own security implications. Nevertheless, this is the principle behind the one-time pad, which we shall come back to in a moment.

Polyalphabetic cipher

A slight variation on the above, which overcomes the key-changing problem, is to use a *polyalphabetic cipher*, in which several substitution ciphers are used in a predetermined, repeating sequence (which actually becomes part of the key). This has the advantage that there is no longer a one-to-one correspondence between plaintext and ciphertext letters. But with a long enough piece of ciphertext, it is still possible using statistical tests to determine the repeat period of a polyalphabetic cipher. To give a flavour of how such tests work, we will now take a quick look at an example, the *Friedman Test*, which uses the Index of Coincidence.

Index of Coincidence

The *Index of Coincidence (IC)* is defined as the probability that two letters randomly selected from the text are equal. This is something that we can measure on plaintext, and for normal English text the answer is about 0.065. If our language was such that all the letters had the same probability, the Index of Coincidence would be about $1/26 = 0.038$ (this can be calculated using the basic probability arithmetic introduced above). Now let us see what happens if we measure the IC of some ciphertext. With a simple (monoalphabetic) substitution cipher, we expect the IC to be close to 0.065, since all we have done is relabelled the letters. The effect of a polyalphabetic cipher is to even out the probabilities of the letters, so the IC would get closer to 0.038. It is even possible to estimate the repeat period of the polyalphabetic cipher from the IC; the longer the period, the closer the IC will approach 0.038.

So we can see how probabilistic techniques can help the cryptanalyst to break a cipher.

Polygraphic ciphers

One early approach to overcoming the letter distribution problem was to group letters together to make a *polygraphic substitution cipher*. For example, the letters could be grouped in pairs to make an "alphabet" with $26 \times 26 = 676$ symbols.

Clearly, handling a histogram with more symbols is more difficult, but the fundamental problem that the symbols have an uneven distribution does not go away. For example, the pair “th” is very common and the pair “zy” very rare. Nevertheless, the bigger the group of letters chosen as the basis for the algorithm, the more difficult the cryptanalyst’s task will be.

Transposition ciphers

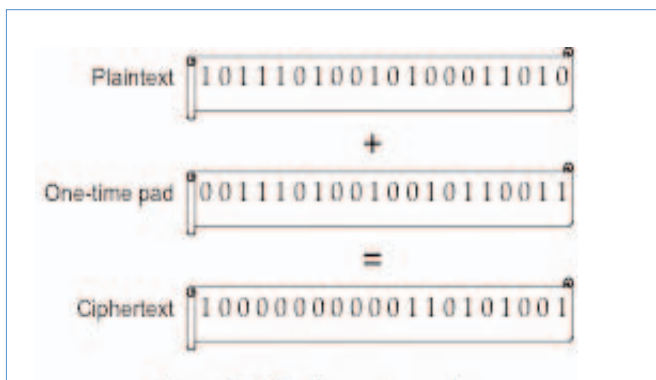
Another approach is to combine substitution with *transposition* - changing the position of the ciphertext as well as its representation. Here, attacks based simply on the distribution of letters are of little help because they say nothing about where the letters are. But more sophisticated use of probability theory can still help the cryptanalyst. For example, we know that adjacent letters in English text are not independent. The probability of *qu* is not the probability of *q* multiplied by that of *u*, but is much closer to the probability of *q* alone, because *q* is almost always followed by *u* (apologies to any Iraqis here!) This fact can be used when measuring probabilities of pairs of letters in the ciphertext spaced by different intervals, to get some idea of the pattern that has been used to rearrange the letters.

The well-known *Data Encryption Standard (DES)* and the newer *Advanced Encryption Standard (AES)*, but nothing to do with audio!) are essentially combinations of multiple polygraphic ciphers (with a block size of 64 bits or 8 letters) with transposition ciphers. It is a testimony to the security of the DES that up to now the only way to break that cipher has been an exhaustive search over all possible keys. Techniques based on probability theory have not worked.

The one-time pad

An important class of encryption algorithm is the *one-time pad*, where the key is as long as the message and any part of the key is only used once. A simple example of a one-time pad, but one that is as secure as any other, is shown in figure 10.

Figure 10: Simple one-time pad



The message is expressed as a sequence of bits and the key is a random sequence of the same number of bits. The encryption algorithm is simply the modulo-2 addition (exclusive-OR) of the message with the key. It is possible to prove that the one-time pad is completely secure; without knowledge of the key, the ciphertext gives no information whatsoever about the plaintext. Any probabilistic attack on the ciphertext will show that every bit will be independent of every other (provided the key is indeed random.)

There are two problems with the one-time pad. The obvious one is that huge amounts of key information have to be exchanged before communication can take place, and this key information must be random. This problem can be overstated. For example, it is possible to generate a matched pair of DVDs containing random data and give one to the sender and one to the receiver. If the messages are limited to text only, it will be possible for that pair of people to encrypt and decrypt around a million pages of information before exhausting the one-time pad.

The other problem with the one-time pad, which should be noted in passing, is that it is not secure against the kind of attack where the eavesdropper knows some plaintext, intercepts the corresponding ciphertext, recovers that part of the key and uses it to generate alternative “fake” ciphertext.

Changing the message

The other set of approaches to improving security is to change the message so that there is less useful information about the probability distribution of letters or groups of letters in the plaintext. This can happen inadvertently, for example if the text is actually in Polish while the cryptanalyst thinks it is in English, or if we are transmitting the famous novel “Gadsby” by Ernest Vincent Wright, which does not contain the letter *e*. A more generally applicable technique is to try to *hide* the letter distribution.

One way to do this with English language plaintext is to give the more frequent letters a series of symbols, one of which is chosen at random each time the letter is used. For example, we could assign three symbols to *e*, two to *a*, two to *o*, and so on. This would have the effect of flattening out the histograms shown in Figures 8 and 9, making an attack based on letter distributions much more difficult. However, this approach would still be susceptible to attacks based on the probabilities of groupings of letters, and it also has the disadvantage that it increases the size of the message because there are more symbols in the “alphabet”.

Compression

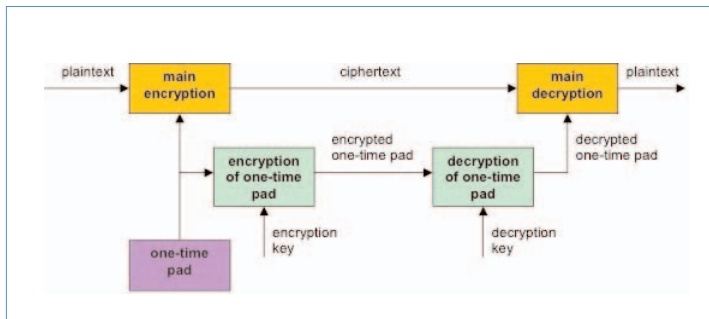
Here is a case where the cryptographer actually seems to get something for nothing! If the message is compressed, the resulting bitstream will appear more random than the original message. This is a fundamental property of compression, because the aim of compression is to remove what is predictable about a message and to leave only what is unpredictable, which has the statistics of random noise. It follows that a compressed message is less susceptible to attacks based on probability distributions of letters or groups of letters, because the variations in these distributions have been taken out by the compression process.

But some care is still needed when encrypting compressed signals. For example, the MPEG standards contain known sequences of bits called "start codes" which are used by MPEG decoders to synchronize the start of a picture or of a stripe of blocks across the picture. The fact that these sequences must exist in the plaintext could help the cryptanalyst.

Encrypted one-time pad

If quantity of data were not too much of an issue, so that we could afford to double the amount of data we send, the following approach could be used.

Figure 11: Encrypted one-time pad



We generate a random one-time pad and use it to encrypt the message. We also encrypt the one-time pad using a smaller key and transmit the encrypted one-time pad. The first part of the encryption is totally secure because it uses a one-time pad as its key. The second part is only as secure as the chosen encryption algorithm, but it is better than directly encrypting a message because its "message" is completely random.

Quantum cryptography

We end with a very brief mention of the latest exciting developments in cryptography. *Quantum cryptography* makes use of Heisenberg's Uncertainty Principle to ensure the integrity of information transmitted across a channel. Information is sent, for example as polarized light, in such a way that any attempt to read it by an eavesdropper will change the information received. By using appropriate error detection techniques, two people can therefore exchange information secure in the knowledge that nobody else has seen it. This technique is still in its infancy, but commercial systems have become available. The cost and difficulty of these techniques mean that for the time being they are most likely to be used not for encrypted messages themselves but for less data-intensive activities such as Diffie-Hellman key exchange.

Conclusions

Three key areas of mathematics that have been applied to cryptography have been described in a tutorial manner. Modular arithmetic is the basis of many older encryption algorithms as well as modern public-key systems, and the special case of modulo-2 arithmetic is important in pseudo-random number generation and in authentication. Previously esoteric theorems about prime numbers are a further essential foundation for public-key systems. Finally, probability theory comes into play, especially in cryptanalysis and consequently in improving the security of cryptographic systems.

Bibliography

As this is a tutorial paper, explicit references to books and papers are not made within the paper. However, the following short list of useful books and Internet resources is offered:

Piper, F and Murphy, S.
Cryptography - a very short introduction.
Oxford University Press, 2002.

Lewand, R.E.
Cryptological mathematics.
Mathematical Association of America, 2000.

Mel, H.X and Baker, D.
Cryptography decrypted.
Addison-Wesley, 2001.

<http://mathworld.wolfram.com>

Acknowledgements

The author would like to thank the Directors of Snell & Wilcox Ltd. for their permission to publish this paper.

For further information

please contact one of our international sales offices:

Americas

New York
Snell & Wilcox Inc.
274 Madison Avenue
Suite 1704
New York City
NY 10016
Tel: +1 212 481 1830
Fax: +1 212 481 2642
americas@snellwilcox.com

Burbank

Snell & Wilcox Inc.
3519 Pacific Ave.
Burbank, CA 91505
Tel: +1 818 556 2616
Fax: +1 818 556 2626
americas@snellwilcox.com

Asia Pacific

Hong Kong
Snell & Wilcox (Hong Kong) Ltd.
Room 603, Tai Tung Building
No.8 Fleming Road
Wanchai
Hong Kong
Tel: +852 2356 1660
Fax: +852 2575 1690
swhk@snellwilcox.com.hk

China

Snell & Wilcox China
D2002D
Cyber Tower Building B
2 Zhongguancun South Road
Haidan District
Beijing PRC, 100086
China
Tel: +86 10 5172 7909
Fax: +86 10 5172 7914
swchina@snellwilcox.com

India

Snell & Wilcox India
NewBridge Business Centre
Technopolis
DLF Golf Course Rd
Sector 54
Gurgaon-122002
India
Tel: +91 124 462 6000
swindia@snellwilcox.com

South East Asia

Snell & Wilcox
South East Asia Sdn. Bhd.
Suite 9-09
Plaza 138
Jalan Ampang
50450 Kuala Lumpur
Malaysia
Tel: +60 (0) 3 2732 5557
Fax: +60 (0) 3 2732 8669
swmalaysia@snellwilcox.com

Europe, Middle East & Africa

UK
Snell & Wilcox Ltd.
Southleigh Park House
Eastleigh Road
Havant
Hampshire
PO9 2PE
United Kingdom
Tel: +44 (0)23 9248 9000
Fax: +44 (0)23 9245 1411
info@snellwilcox.com

France

Snell & Wilcox France
3 rue de Rome
93110 Rosny Sous Bois
France
Tel: +33 (0) 1 45 28 1000
Fax: +33 (0) 1 45 28 6452
swfrance@snellwilcox.com

Germany

Snell & Wilcox GmbH
Senefelderstrasse 3a
65205 Wiesbaden
Germany
Tel: +49 (0) 6122 98430
Fax: +49 (0) 6122 9843 55
swgermany@snellwilcox.de

Russia

Snell & Wilcox AO
Room 214
35 Arbat str
Moscow 119002
Russia
Tel: +7 495 248 3443
Fax: +7 495 248 1104
swrussia@snellwilcox.com