MAT199: Math Alive
Cryptography – Part 1

1 Introduction

A cryptographer *encodes* messages (typically texts in some standard language; we will keep with English here) before they are transmitted (by courier, over radio, over the internet, etc.) so that even if the encrypted message is intercepted by a hostile party, its meaning will (hopefully) still remain secret. In principle, only "friends" of the original cryptographer, who know the secret recipe for *decoding* or *decrypting*, can decode the encrypted message and recover the original plain text.

Cryptography schemes have existed for thousands of years, and attempts to "break" coding schemes are almost as old. A "code breaker" seeks to detect patterns in the encrypted messages that will lead to sufficient understanding of the encryption scheme to enable the discovery of a decryption method.

In this module of the Math Alive course we shall discuss several cryptography schemes, starting with a 2000 year old scheme, and ending with a discussion of the public-key cryptography used in current day technology for transmitting credit card information over the internet.

2 Old standards: transliteration schemes or substitution ciphers

The oldest schemes replace the letters in the message one by one, following a fixed recipe. The "key" to such a transliteration scheme just lists all the letters in the alphabet and gives for each letter the corresponding crypto-letter. For instance:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
G	O	K	E	A	N	Q	U	Y	C	P	T	L	F	H	W	B	M	V	X	Z	R	I	D	S	J

indicates that every A will be replaced by a G, every B by an O, etc.

To decrypt schemes of this nature, one just needs to reverse the transliteration; in the example above one would replace every G in the encrypted message by A, every O by B, every K by

C, etc. The decoding is thus again a transliteration. For the example above, the decoding transliteration is

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
E Q J X D N A O W Z C M R F B K G V Y L H S P T I U

One of the very oldest documented transliteration schemes was used by the Roman general Julius Caesar (1st century BC). His transliterations were particularly simple because they involved only a *shift* of the alphabet, such as in the following example:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T

This example was obtained by writing a second copy of the ordered alphabet for the transliteration, starting 6 steps to the right, with A underneath the letter G in the first line, and continuing until T is written underneath Z. At that point, the writing gets "wrapped around" and the alphabet is completed, from U to Z, by writing underneath the letters A through F from the first line. Codes of this simple shift type are called *Caesar codes*.

Note that the corresponding decoding transliteration

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
G H I J K L M N O P Q R S T U V W X Y Z A B C D E F

which consists of moving 6 steps to the *left* (or equivalently, 20 steps to the right: after all, moving 26 steps is the same as not moving at all). This is again a Caesar code.

A variant on Caesar codes is obtained by not only shifting the starting point of the alphabet, but writing the alphabet in the opposite order, as in the following example

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
F E D C B A Z Y X W V U T S R Q P O N M L K J I H G

In this case the decoding scheme is the same as the encoding scheme: the transliteration replaces M by T and T by M; the same symmetry holds for all other letters. These "mirrored Caesar codes" are used as one element in more complex schemes to which we shall return later.

If the language in which the plain text is written is known to the code breaker, and if the messages contain a few sentences of text, then it is very easy to break the transliteration codes by the following trick. In every language some letters are used more often than others. In table 1 we show the typical frequencies for the 26 letters of the alphabet in English: We see that the most frequently used letter is E: on average, 12.62% of the letters in an English text are Es. The least frequent letter is Z, occurring, on average, only 0.09% of the time. If a text is transliterated from English, we can check which letters occur most frequently. Suppose that a typical text is encoded with the first transliteration example above. It is likely that the most frequent letter in the original text is E, which would be encoded as A in our first transliteration

LETTER	STANDARD FREQUENCY
a	.0761
b	.0154
c	.0311
d	.0395
e	.1262
f	.0234
g	.0195
h	.0551
i	.0734
j	.0015
k	.0065
l	.0411
m	.0254
n	.0711
o	.0765
p	.0203
q	.0010
r	.0615
s	.0650
t	.0933
u	.0272
v	.0099
w	.0189
x	.0019
y	.0172
z	.0009

Table 1: Frequencies of occurrence of individual letters in English text.

scheme. Thus, if we are trying to break the code, we would soon observe that the most frequent letter in the encoded text is A, and be led to the reasonable guess that A stands for the original letter E. Then we would check for the next most frequent letter and would surmise that the crypto-letter X stands for the original letter T, and so forth. Once sufficiently many letters are decrypted, some of the text becomes legible, and guesses about its content can be used to help with the further decryption. If one observes that the transliteration is a simple alphabet shift, then it is even easier to complete the transliteration table after the first few letters.

2.1 Making transliteration more complex

Transliteration codes are easy to break, because known and recognizable patterns in the original text (frequency of letters) persist in the encrypted text, and can be exploited to construct a decoding scheme. Codebreaking attacks always try to detect patterns and use these to get a handle on the coding key, which then suggests a decoding key.

To make a coding scheme hard to break, it is thus important to disguise patterns as much as possible. One way to disguise the relative frequency of letters which persists in transliteration is to alternate between different transliteration schemes, e.g., using one scheme for the 1st, 3rd, 5th, ... letters, and another for the 2nd, 4th, 6th, ... letters. Given a sufficiently long text, the slightly more complex patterns that persist through this scheme are still easy to detect, and every code-breaking expert would make very short work of such an alternating scheme.

The more transliterations used, and the more randomly the different transliterations succeed each other, the harder it becomes to break the code. In fact, suppose you write a *completely random* sequence of the numbers 0 to 25, e.g., 23 5 9 0 2 17 21 13 14. . . , for as many letters as you have in the text you wish to encode, and you use these numbers, one by one, to determine the Caesar code shifts for the letters in your text. (In this example, we shift the first letters 23 letters to the right in the alphabet, the second 5 steps, the third 9 steps, and so on.) Then your encoded message could *not* be decrypted, provided you *never* used the same encryption sequence again. What was just described is the *one-time pad method*, which is known to be unbreakable. (Short of human blunders: if your enemies physically capture your pad, *i.e.*, a record of your sequence of shifts, then they can of course break the code.) If you use the same sequence of numbers over and over again, then the scheme starts leaving detectable patterns, and it can be broken; to be secure, you must use a new pad every time you want to encrypt anything. This makes it very cumbersome for your intended recipient to decrypt messages: somehow, the two of you must have agreed on all those encryption pads beforehand, so that your recipient has them all (and if you intend to have regular transmissions, there will be a lot of them, since each can be used only once).

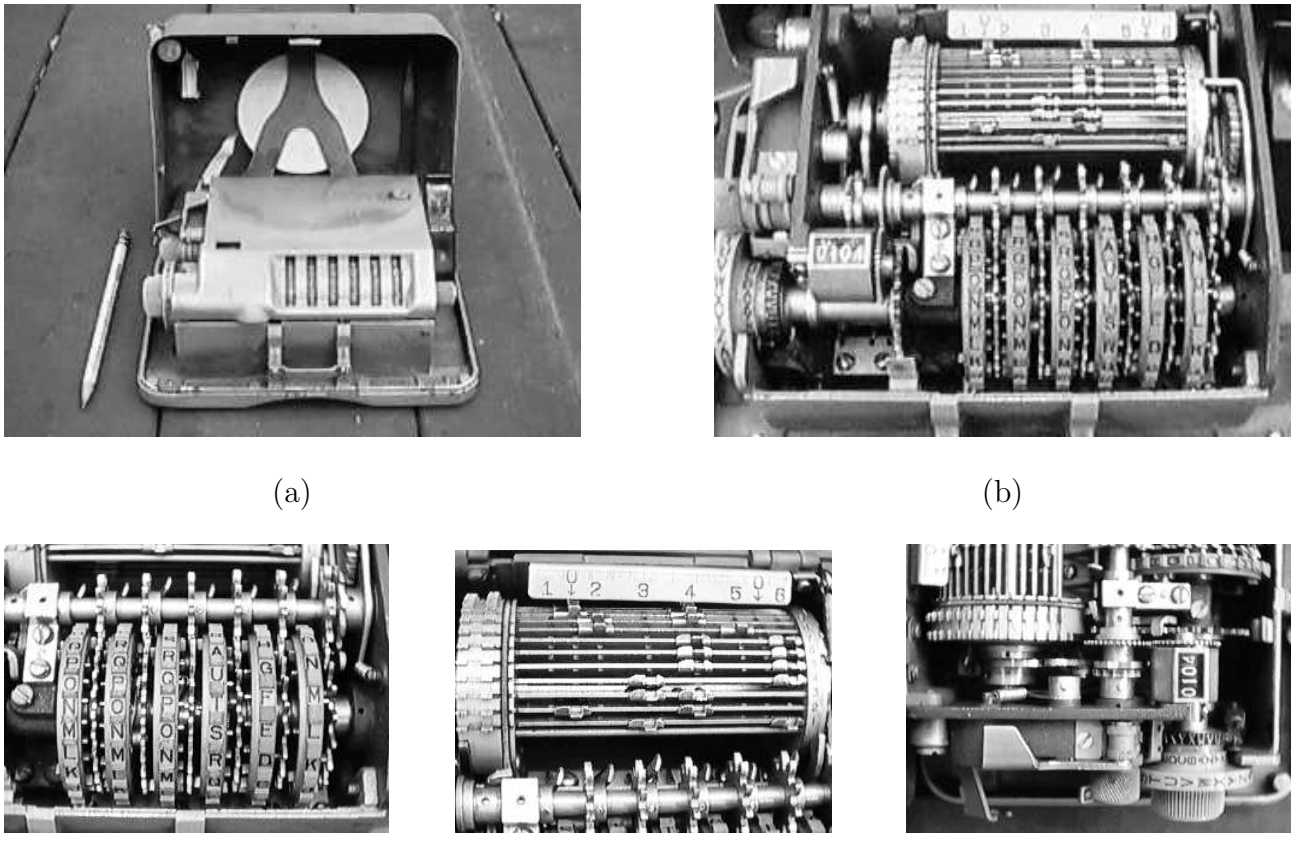
Around Worl War II, several inventors developed coding machines that used constantly changing transliterations, with complex and (they hoped) hard-to-detect patterns in the ordering of the shifts of the consecutive transliterations. There were several major inventors, e.g., Edward Hebern (the ECM machine), Arthur Scherbius (the celebrated Enigma machine) and Boris Hagelin (the Hagelin machine). Although the Enigma machine is the most famous, Hagelin was the only one of the three to make serious money from his invention. Versions of his machine

were in use in many military and diplomatic units, until well into the 50s; the company he started (the Crypto AG company, based in Switzerland), is still a large corporation today.

The Hagelin Cryptograph: One of the most popular cryptographs in the 1940's and 1950's

This machine uses ever-changing "mirrored Caesar codes": for each letter, in the message to be encoded, the machine first determines a shift of the alphabet, and then copies the alphabet (with shift) in reverse order.

The Hagelin machine changes the shift for every letter to be encoded via an ingenious system that tries to make the successive shifts very "random". Several views of a Hagelin cryptograph and its machinery are shown in figure1 below.



(a) (b)

(c) (d) (e)

Figure 1: The Hagelin cryptograph machine.

2.1.1 Description of the mechanism

The machine contains six large rotors (gears). To the left of each rotor is a big wheel of about the same diameter labeled with letters, with one letter for each tooth of the rotor gear. Each

lettered wheel spins along with its rotor and thus tells us the position of that rotor. It is easy to see the six big lettered wheels in figure 1(b) and (c), and then one can notice the gears with pointed teeth in between the lettered wheels.

At each step of encoding or decoding, each rotor is rotated by one tooth of the gear, and its lettered wheel advances one letter along with it. (For example, look at figure 1(c). After one step, the letter N on the leftmost wheel would be where M is in the picture now.) Each of the lettered wheels has pins sticking out of it. In figure 2 below, one such wheel is sketched, with only one pin shown, but in fact there is one pin per letter on the lettered wheel. They are not so easy to see in the pictures of the actual machine, but look at the leftmost lettered wheel in figure 1(c). Here one can see a little shiny tab to the left of the letter K. That is one of the pins, which has been pushed so that it sticks out to the left of the wheel. On the other hand, next to the letters L, M, N, O, and P, one can see the pins sticking out to the right of the wheel – they are the shiny little tabs that one can see between the lettered wheel and the rotor. Every lettered wheel has these pins. (For example, the rightmost wheel has the pin for K to the right, for L to the left, for M to the right, and for N and O to the left.)

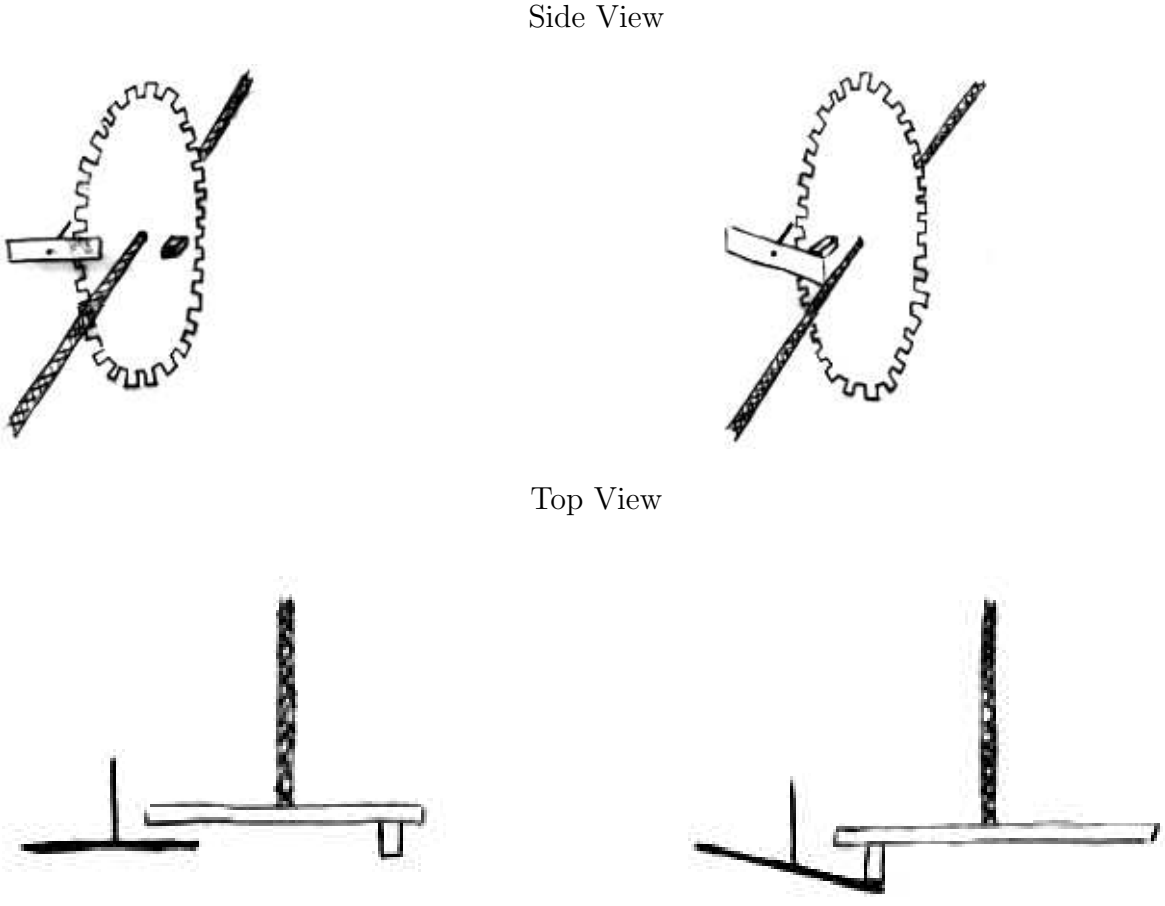


Figure 2: Schematic diagram of the Hagelin cryptograph.

In one position, these pins push against a little lever, while in the other they don't. These levers can be seen in figure 1(c) and (d). Above the rotors and lettered wheels there is an axle with

smaller gears. Just above this axle, one can see six little tabs, which are the tops of the levers. Note the two of the six levers are a little bit higher than the other four (the third and sixth lever, if we count from the left). As we rotate a lettered wheel, the lever above it will move up and down depending on whether the pin near the lever is to the left (which causes the lever to be down) or to the right (which causes the lever to be up). For example, the leftmost lever is controlled by the pin positions on the leftmost lettered wheel. When the portion of the leftmost wheel visible in figure 1(c) (labeled with letters K–P) is rotated around to the back where it interacts with the leftmost lever, the pin near K (which is to the left) will cause the lever to be down. If we rotate the wheel a little further, the pin near L (which is to the right) will change the lever to the up position. Then for four more steps the lever will stay up (because the pins near M, N, O, and P are also to the right). The lever will move down in the next step because the pin near Q is to the left (this is barely visible in Figure 3).

When the lever is in this "up" position, it moves some of the little riders, called lugs, on bars of metal on a rotating drum; in the "down" position it does nothing. The drum and its lugs can be seen clearly in figure 1(b) and (d). The drum makes a full 360-degree rotation each time the machine is cranked for the encoding of the next letter.

A lever in the "up" position pushes to the left any of the lugs on the drum that are near it. When any little rider (or lug) gets pushed out of place, it drags along with it to the left the entire horizontal bar of metal to which it is attached. This causes the bar to stick out of the leftmost edge of the drum. One can also see that the leftmost portion of each bar has two little ridges which stick out from the left rim of the drum. When the bar is not being deflected by a lever pushing one of its lugs, these ridges don't touch anything. But when a bar is pushed to the left, these ridges are also pushed to the left, and one of them will touch a gear wheel (too deep in the machine to see from the photos) and advance it one step. Since many lugs might be deflected during a single 360-degree rotation of the drum, the gear wheel might be advanced many steps during a single step of encoding.

Finally, this gear wheel is connected with the spinning alphabet wheel (this is not one of the six lettered wheels mentioned before, but rather the letter wheel with a handle which one can see at the bottom of figure 1(e) and the far left of figure 1(b)). So the shift in a given step of encryption is determined by how many times this last gear is advanced; the gear is advanced one step for each lug that is moved leftward; each such lug is only moved if the lever near it is in the "up" position; each such lever is only up if the pin near it is positioned to the right. The six different rotors (along with the lettered wheels that bear the pins) are turned by one gear step each after every letter encoded. This changes which pins are pushing on the levers, which changes which levers are up and down, so that the number of lugs moved changes as well; therefore the alphabet shift will change from one step to the next.

Complexity of the Hagelin encryption scheme

The six rotor gears from left to right have respectively 26, 25, 23, 21, 19, and 17 teeth. The leftmost rotor will therefore be in its original position after every 26 cranks, the second after every 25 cranks, and so on. One can then ask after how many cranks (N) the rotors will all be

back in their original position; N determines the period of the whole encryption scheme, *i.e.*, the number of encryptions after which the pattern starts repeating.

N has to be a multiple of 17, 19, 21, 23, 25, and 26. Since these numbers are relatively prime (no pair has a common divisor), N must be their product:

$$N = 17 \times 19 \times 21 \times 23 \times 25 \times 26 = 101,405,850 \ .$$

This is a very large number, much larger than the number of letters in any text one would like to encrypt. However, even though it takes this very large number of cranks before the whole system repeats exactly, because it is made up of six rotors, each with its own system of pins that cycle back to their original position much faster, the whole mechanism leads to more patterns than a single rotating wheel with N gears would have (which of course cannot be constructed!). Thus there are more patterns than just this repetition every $101,405,850$ steps. In addition, if the machine, with the same settings, is used by many senders (e.g. in a military setting at war), then the code-breaker can use patterns present in many of their messages. As a result, this code can be (and was) broken, especially if it is used unwisely, for instance, if the internal settings, governed by the pins and lugs, are not changed sufficiently frequently. Nevertheless, the code breaking takes significantly more work than if the alphabet shift did not change with every letter.

3 A newer scheme, used by HBO in the 1980s

(Some explanation of this can also be found in For All Practical Purposes.)

HBO transmissions to their subscribers (some of whom are licensed local cable stations who then transmit the program, over cable, to *their* subscribers; others are individuals who pay their subscription directly to HBO and who capture the signal with their own antenna dish) have been encrypted since the 1980s. The encryption of the movies and other programs distributed by HBO is the same for everyone, so that HBO has to broadcast only one encrypted version. The encryption is governed by a password that consists of a sequence of 56 digits, each 1 or 0, e.g.,

$$11010100011010110100110111101100011 \ldots .$$

Because the subscribers of HBO pay their fee monthly, the password used by HBO changes monthly as well: subscribers who don't pay their fee for a given month won't receive the password for that month, and won't (in principle) be able to decode the HBO programs that month.

The next question HBO faces is how to transmit, each month, the password to only those customers who paid their subscription for that month. Rather than send out thousands of letters, HBO broadcasts the password as well, with an additional layer of encryption, different for each user. Each customer has a key, which is also a binary number of 56 digits. Each successive digit in this key tells us how HBO "shifts" each successive "letter" of the password in order to encrypt it. This is exactly like in our alphabet examples, except that in our alphabet

now there are only two digits, 0 and 1. A 0 in the key means no shift: 0 remains 0, 1 remains 1, or, in the double-line format we saw for letter encoding:

$$\begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix}$$

A 1 in the key means a circular shift: 0 becomes 1, 1 becomes 0, or:

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$$

The password encrypted with this key is thus obtained as follows:

$$\begin{array}{rl} \text{password} & 110101000110101101\ldots \\ \text{key} & \underline{010110100100111101\ldots} \\ \text{encrypted password} & 100011100010010000\ldots \end{array}$$

In order to recover the password, you just use your key to undo all the shifts one by one, which in this case is actually the same as just repeating the shifts, *i.e.*, encryption = decryption, just as for the mirrored Caesar codes.

> "Shifting" by 0 or 1 can also be viewed as a special kind of addition of binary numbers in which you "forget" the carry:

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \underline{@\ 0} & \underline{@\ 1} & \underline{@\ 0} & \underline{@\ 1} \\ 0 & 1 & 1 & 0 \end{array}$$

↑ forget the carry.

> You can check that the "encrypted password" above is obtained by such carry-less addition, which is often called *parity addition* or *XOR addition*. In this week's homework, you can get some practice on working with binary numbers, with binary addition and with parity addition.

HBO sends out a whole sequence of [password @ key], with a different key for each paying customer. The customer's unscrambling box just tries its key on each of these 56-bit sequences, and tries to decrypt with the computed password candidate. When the right one is reached, the picture on the screen looks suddenly miraculously perfect; this can easily be detected automatically, and the unscrambler now knows the password for the next month.

The situation is made slightly more complicated in that the key itself of every customer changes every month. That change is done via a pseudo-random generator, of which customer and HBO have identical copies. Because the "seed" from which the pseudo random number generator

starts is different for all customers, their keywords every month will be different as well. But the pseudo random number generator is chosen so that working "backwards" is not known to be possible. Otherwise, you could pay one month, get the password that month, use it to compute the keys for that month of all the other customers and from there work backwards to find their "seeds". By installing such a pirated seed in your unscrambling box you would be able to unscramble the programs without paying your fee, since your box would now generate, every month, the key of another subscriber. Of course, if that subscriber discontinues service, you would not be able to unscramble the programs anymore. But if you keep a large enough collection of pirated seeds, it is unlikely that all the subscribers whose seeds you pirated would discontinue service at the same time.

Note: it didn't take long before pirates found ways around the HBO scheme, despite the added security layer; it has been replaced since then by other schemes.

> How can it be easy to go forward from a seed, but hard to go backwards?
>
> Let us look at an example in decimal numbers, instead of binary, and with passwords of four digits only. I have to tell you in a very simple way how to compute key_1, key_2, ... (a sequence of successive keys) in such a way that a snooper would not be able to guess key_{n+1} even if he had intercepted key_1, key_2, ..., key_n. Imagine that I tell you to compute the decimals of π, and to take successive blocks of four digits, starting with the fifth:
>
> $$3.1415\ \underbrace{9265}_{\text{key}_1}\ \underbrace{3589}_{\text{key}_2}\ \underbrace{79\ldots}_{\text{key}_3}\ \ldots$$
>
> Since it is possible to compute millions of digits of π quite accurately, you can go on, happily computing key_n every month, for a long time. But the different key_n's do not appear to have any relationship to each other, and if you didn't know the number was π to start with, you'd be in trouble to guess the next one.
>
> In practice, you wouldn't use π, because that is so obvious an example, but some other number that is not so universal and can also be computed with great accuracy.

a second) to test whether a password guess is correct,