Encryption Using a Variant of the Turning-Grille Method

Article *in* Mathematics Magazine · December 2002

DOI: 10.1080/0025570X.2002.11953934

1 author:

Stephen Fratini

17 PUBLICATIONS **33** CITATIONS

SEE PROFILE

Encryption Using a Variant of the Turning-Grille Method

STEPHEN FRATINI
50 Malibu Drive
Eatontown, NJ 07724

In his *De Subtilitate* [**1**], the sixteenth-century mathematician Girolamo Cardano introduced a new method of encryption/decryption. To read an encrypted text, a square grid (or grille) with the correct openings was placed over the text to reveal the decrypted message. Variants of this technique were used as late as the mid-twentieth century. For example, German intelligence used a grid-based encryption in South America during World War II. In this note, we present other variants of this method and describe some modern encryption techniques in conjunction with grids.

By a *grid* we will mean a rectangular array that covers a rectangular text (of the same size) with small openings or cutouts in various places. The dimensions of a grid determine the number of characters that it can cover. For example, a 16×16 grid can cover 16 lines of text, each line having 16 characters. In Cardano's method, we place a grid over a blank piece of paper and write the characters of our message in the grid openings. The grid is lifted off the paper and random characters are added around the message, thereby creating a rectangular block of text of the same dimension as the grid. The intended recipient of our message also has an exact copy of the grid used to encrypt the message. The message recipient places his or her grid over the block of text and the message appears through the grid openings.

A variant of Cardano's approach uses rotations of the grid to introduce encoded text more compactly than in the original approach. For example, in FIGURE 1, we show an 8×8 grid with 16 openings. When placed over an 8×8 text, this grid exposes 16 characters. If we rotate the grid 90° in the clockwise direction and again place the grid over the selection of text, a different set of 16 characters is shown. Two more rotations of 90° will expose two more sets of 16 previously covered characters. The 8×8 squares of FIGURE 1 are in four groups of 16 squares, and the numbering in each quadrant is rotated 90° clockwise each time. Using this numbering, we see that each of the 16 openings (under each rotation) has a distinct number (from 1 to 16). This ensures that all of the 64 squares of the text will be uncovered in a unique rotation of the grid. Text can be put in encrypted form after each rotation, so that the entire 8×8 array may contain up to 64 characters of (encrypted) text, without needing to insert any

1	2	3	4	13	9	5	1
5	6	7	8	14	10	6	2
9	10	11	12	15	11	7	3
13	14	15	16	16	12	8	4
4	8	12	16	16	15	14	13
3	7	11	15	12	11	10	9
2	6	10	14	8	7	6	5
1	5	9	13	4	3	2	1

Figure 1 An 8×8 grid

random characters. This encoding technique is referred to as the *turning-grille* method. For further reading, we recommend the excellent introductions to the turning-grille method and its associated history in the books by Gardner [**3**] and Kippenhahn [**5**].

Example: the Turning-Grille Method Consider the following phrase:

THE CONCEPT OF USING GRIDS TO ENCODE MESSAGES WAS FIRST
DISCOVERED BY THE MATHEMATICIAN GIROLAMO CARDANO

We arrange the characters of the message in 8 rows of 8 characters (blank spaces between words are removed). If there are more than 64 characters, we use additional 8×8 arrays until all the text has been encoded.

The plaintext (that is, unencrypted text) is inserted into the arrays as suggested by FIGURE 2 from left to right, and from top to bottom, starting with the array on the left and then placing the remaining characters into the array on the right. Next, we place the grid shown in FIGURE 1 over the left array and read off the characters appearing through the openings (again, we go from left to right, and from top to bottom). We repeat the procedure after rotating the grid 90°, 180°, and finally 270°, and then go through the entire process again for the array on the right.

T	H	E	C	O	N	C	E
P	T	O	F	U	S	I	N
G	G	R	I	D	S	T	O
E	N	C	O	D	E	M	E
S	S	A	G	E	S	W	A
S	F	I	R	S	T	D	I
S	C	O	V	E	R	E	D
B	Y	T	H	E	M	A	T

H	E	M	A	T	I	C	I
A	N	G	I	R	O	L	A
M	O	C	A	R	D	A	N
O							

Figure 2 Encoding with an 8×8 grid

There are several approaches that we can use when the last block of plaintext does not fill the grid: fill the grid with dummy text, partially fill the grid with dummy text, or do not add any fill. We'll use the no-fill method in this example. For readability, we present the encoded text in groups of four. The actual coded message would be transmitted without spaces. The following encoding is obtained:

TTOS NSOE GRIV EDHE EITE NCOS SSTD YTMA COPF UGDD ASIS ORET
HENC GRIM EESW AFCB
HNGO ADNI LAOA TAIR MREM ICOC A

Of course, both the encoding party and the decoding party need to know how the characters are placed into the arrays and how the characters are read off when the grid is in place.

To decode the text above, we place the grid in FIGURE 1 over a blank array. Starting from the left of the top line of text, the characters are written into the openings of the grid (from left to right, and from top to bottom). The process is repeated for grid rotations of 90°, 180° and 270°. For the second line of encoded text, the procedure is slightly different. We first note that the text will fill only the first three rows and the first entry in the fourth row (this will be referred to as the *covered region*). No text

should be inserted into a grid opening outside of the covered region. Other than this stipulation, the procedure is the same as the case where the text completely fills the array.

We shall call a *complete grid* one that exposes all of the characters of a selection of text exactly once when successively rotated over the text. The grid shown in FIGURE 1 is an example of a complete grid. Note that this condition implies that in order for an $n \times n$ grid to be complete, n must be an even integer.

The turning-grille method does have weaknesses. First, when the complete-fill method is used, an interloper may be able to detect the grid size of the code from the sizes of the intercepted messages. This weakness can be overcome by using no-fill or partially-filled grids in order to disguise the size of the grid. A second weakness arises when several adjacent openings appear in a row or column. Adjacent openings can expose entire words or parts of words, and a code breaker could try to determine the arrangement of the openings from this information. Third, for smaller grids, a would-be code breaker could easily try all possibilities. For example, we will show there are only 65,536 different 6×6 grids. Of course, this problem can be remedied by using a larger grid. In the next section, we provide a formula for the number of grids of a given dimension.

The difficulty of breaking a code In order to estimate the difficulty in breaking these codes we will count the number of possible grids for a given dimension. The more possible grids there are, the longer it would take to break the code by exhaustively checking each possible grid.

Grids of the same dimension differ in the location of their openings. If one grid can be rotated clockwise so that it has openings in the same position as another, we say the two grids are *equivalent*. Since we use the turning-grille method, we'll only be interested in counting the number of equivalence classes of complete grids.

Let's consider the 8×8 example again. From our earlier discussion, we know that in order to determine a complete grid, it is sufficient to assign to each integer from 1 to 16 the unique quadrant in which it appears in FIGURE 1. Each number from 1 to 16 appears exactly once in each quadrant, so choosing one quadrant for each integer will guarantee that every square of the array is uncovered in exactly one of the rotations of the grid. Since each integer has four possible quadrants to which it can be assigned, and the assignments for each integer can be made independently, it follows that there are 4^{16} such choices. However, any one such choice will be equivalent to three other choices, one for each rotation. So, the total number of equivalence classes of complete grids is $4^{16}/4 = 4^{15}$.

In general, we see that each quadrant of an $n \times n$ array contains $(n/2)^2$ squares (where n is even). So, we need to assign quadrants to the integers 1 to $(n/2)^2$, and divide by 4 to account for equivalent grids, resulting in $4^{(n/2)^2-1}$ equivalence classes of complete grids.

It's worth noting that complete grids for $n \times n$ arrays where n is an *odd* integer are not entirely impossible. In using such an array, there will be one square in the center of the array that will be fixed by every rotation. Simply excluding its use in the encoding and decoding process, we can again determine four distinct (though non-square) quadrants to which we can apply the turning-grill method. As an exercise the reader should show that the number of equivalence classes of complete $n \times n$ grids, for n odd (with the center excluded) is 4^{k-1}, where $k = (\frac{n+1}{2})(\frac{n-1}{2})$.

Linear grids We now consider a variant of the square grid called a *linear grid*. A linear grid is a linear array of squares that is divided into pieces of equal lengths called *sectors*, with openings or cutouts appearing at different locations along the grid. These

sectors take on the same role as the quadrants in square grids, so that a linear grid will never contain more openings than the common length of its sectors. The top row in FIGURE 3 depicts a linear grid with sectors of size 5 (double lines are used to show the boundary between sectors). We will always assume the length of a sector is at least 2. The role of a rotation in square grids is replaced by a *shift*. A shift moves the openings of a sector over to the sector to its immediate right. The openings of the last sector of the grid are then shifted to the front on the first sector (the second row in FIGURE 3 is the result of applying one shift to the first row).

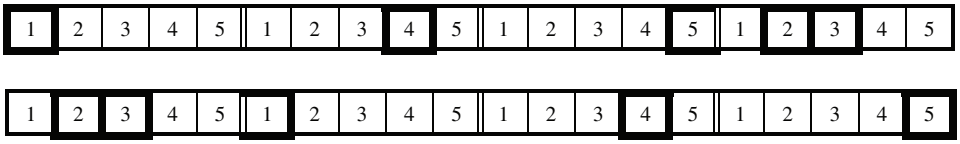


Figure 3 Sector shifting in a linear grid

We can create sectors of any size. FIGURE 4 shows a linear grid with 6 sectors, where each sector has 5 elements. A linear grid consisting of m sectors with each sector having n elements is said to have dimension $m \times n$.

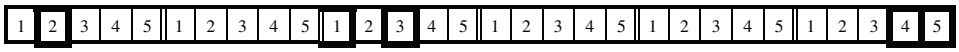


Figure 4 Linear grid with 6 sectors each of size 5

Two linear grids of the same size are said to be *equivalent* if one grid can be transformed into the other by shifting the grid one or more times. In general, there are m^{n-1} equivalence classes of linear grids of dimension $m \times n$, where $m \geq 2$ and $n \geq 2$. This result is easy to prove using the ideas developed in the square grid case. In an $m \times n$ linear grid, n denotes the length of the sector (and consequently the number of openings in the grid), so that in order to determine such a grid, we need to assign one of the m possible sectors to each integer from 1 to n. Like the square grid case, these can be assigned independently, so there are m^n such choices, but each of the m shifts of any single grid determines an equivalent grid. Thus, there are only m^{n-1} distinct equivalence classes of linear grids.

Permutations The encoding process determined by a linear grid also determines a permutation on the set of characters of the text. In this way we can regard the set of $m \times n$ linear grids as a subset of the set of permutations of degree mn. As an example, consider the permutation generated by the 4×4 linear grid shown in FIGURE 5.

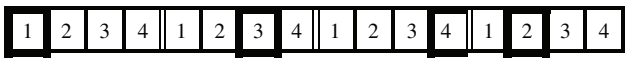
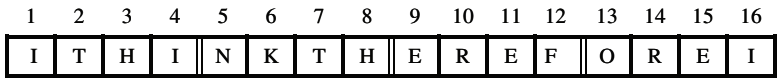


Figure 5 Example of a 4×4 linear grid

The sequence of text



is mapped to

1	7	12	14	2	5	11	16	4	6	9	15	3	8	10	13
I	T	F	R	T	N	E	I	I	K	E	E	H	H	R	O
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

In permutation notation, this mapping can be represented as

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 5 & 13 & 9 & 6 & 10 & 2 & 14 & 11 & 15 & 7 & 3 & 16 & 4 & 12 & 8 \end{pmatrix}.$$

We can apply the encoding process several times (say p times) to a text. To decode the message the receiving party must apply the decoding technique p times. Algebraically, this means that if a linear grid determines a permuation σ, and is applied p times, this is equivalent to applying the permuation σ^p, and the decoder must apply σ^{-p}. Clearly p must not be a multiple of the order of the permutation, otherwise the encoding would map the selection of text to itself!

Finally, we note that a partially filled grid gives rise to a different permutation than the same grid when completely filled, in terms of the size of the permutation and the actual mapping. If, for the grid shown in FIGURE 5, only the first 12 places are filled, we generate the following permutation (written in cycle notation): $(1)(2, 4, 7)(3, 10, 12)(5)(6, 8, 11)(9)$, which is different from σ.

Code parameters In order for two parties to have secure communications they need to agree on their encryption technique (e.g., linear grids), the initial parameters associated with the code (e.g., the dimension of the linear grid and the location of the openings), how often these code parameters are to be changed (e.g., every 3 days), and how to indicate a change to the code parameters (sometimes referred to as *key distribution*).

Clearly one might never change the code parameters, or change them according to a fixed, pre-set schedule. Another approach is to embed the code parameters in the message itself. This way one can change the encoding technique at any time.

We will assume some initial choice of code parameters has been made between the sender and receiver. In order to specify within the message a change in the code parameters, we need to indicate within this message the dimension of the grid and the location of the openings. One method would be to insert the grid description into a selection of dummy text (or into the actual message). For example, consider the 4×6 grid shown in FIGURE 6.

We will use the dummy text: "THE YANKEES WON THE WORLD SERIES." Two consecutive letters of the dummy passage will indicate the end of a sector, and one letter will separate the numbers indicating the location of an opening within a sector. Since there may be multiple sectors without openings at the end of the grid, it is necessary to indicate the end of the grid description. This is done by placing any single-digit number after three characters of the dummy passage. Using this technique, the code parameters for the grid shown in FIGURE 6 can be represented as

T4HE 2Y3A N6KE 1E5S WO7NT HEWO RLDS ERIES.

A similar embedding technique can be defined for square grids.

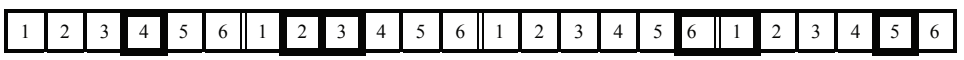


Figure 6 A 4×6 linear grid

There are still two problems with this procedure. If the current key is discovered, then all subsequent keys can be deciphered (assuming the code breaker can determine how the keys are embedded in the message). Secondly, selection of the linear grid's dimension and the configuration of openings is not automated. In the next section, we will see how modern encryption techniques can be used to solve these two problems.

Applying modern encryption techniques Modern encryption makes extensive use of public keys and binary representation of encoded messages. We shall demonstrate how both of these concepts can be used to strengthen the security of linear grids.

Key distribution was a major problem for all encryption techniques until the latter part of the 20th century. This changed in 1976 with the discovery of a key exchange method that allowed two parties to establish a secret key via a public discussion [**2**]. The key distribution procedure is known as the *Diffie-Hellman key exchange scheme*. This scheme depends on a modular arithmetic function of the form $A^B \pmod C$ where A, B and C are positive integers, C is a large prime (a 1024-bit number or larger), and $A \leq C$. Further, A should be selected so that the powers of A generate all elements from 1 to $C - 1$, modulo C.

We now give a brief example of how the Diffie-Hellman key exchange works. For simplicity, we use a small value for C.

1. Two parties wishing to communicate securely (say Abe and Bea) agree on values for A and C (say $A = 51$ and $C = 53$). A and C are referred to as *public keys*. The value of A and C need not be kept secret.
2. Abe and Bea each select a *private key*. No one except Abe knows Abe's private key, and no one except Bea knows Bea's private key. Let Abe's private key be $x = 7$ and Bea's private key be $y = 5$.
3. Abe computes $L \equiv A^x \pmod C \equiv 51^7 \pmod{53} \equiv 31$. Bea computes $M \equiv A^y \pmod C \equiv 51^5 \pmod{53} \equiv 21$. Abe and Bea exchange their results. The values of L and M need not be kept secret, since they are not the key to the code.
4. Now Abe computes $21^x \pmod C \equiv 21^7 \pmod{53} \equiv 35$, and Bea computes $31^y \pmod C \equiv 31^5 \pmod{53} \equiv 35$. At this point, Abe and Bea have determined a common key.

Any two parties using the above procedure always arrive at the same key since

$$L^y \equiv (A^x)^y \equiv (A^y)^x \equiv M^x \pmod C.$$

The function used in the Diffie-Hellman key exchange, that is, $f(x) \equiv A^x \pmod C$, is called a *trapdoor one-way function*. A function is called *one-way* if it is significantly easier to evaluate than is its inverse. A trapdoor one-way function is a one-way function in which the inverse is easy to compute if certain secret information (referred to as the trapdoor) is available. The private keys in the Diffie-Hellman key exchange are the trapdoor.

For large values of C (that is, at least 1024 bits), solving for the private keys is considered computationally difficult (that is, practically impossible). At present, no one has discovered an algorithm that solves the general discrete logarithm (the function needed to invert the one-way function in the Diffie-Hellman exchange) in polynomial time. In fact, the U.S. government's Digital Signature Algorithm (DSA) is based on a variant of the Diffie-Hellman key exchange [**6**].

We can use the Diffie-Hellman procedure to exchange keys instead of the key embedding technique described in the previous section. There is one complication, however. The Diffie-Hellman exchange produces an integer and not a description of a

linear grid. We solve this problem by defining an ordering on the set of all linear grids, and will use the grid and code parameters associated with the location in the ordering determined by the integer produced by the Diffie-Hellman exchange.

The ordering is defined in two steps. First, we order the dimensions of the set of linear grids and then we order the (finite set of) linear grids within a given dimension. FIGURE 7 illustrates how the various dimensions are to be ordered. The first grid is (2, 2), followed by (3, 2) and (2, 3), and then (4, 2), (3, 3), and (2, 4), etc.

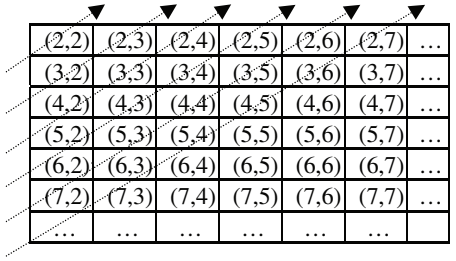


Figure 7 Ordering of linear grid dimensions

Next, we define an ordering of the set of linear grids of the same dimension. Given two sectors within a particular grid, we'll say the sector with more openings is *greater* than the other. If two sectors have the same number of openings, then the sector with the opening in the highest number (furthest to the right in the sector) is greater than the other sector. There cannot be a tie since each slot opening appears in exactly one sector of a given linear grid. In this way, we can create an ordering of the sectors within a grid. Now take each linear grid and shift the sectors such that the greatest sector appears farthest to the left. Effectively, we are selecting one representative from each equivalence class. We write each linear grid in the form $a = (a_1, a_2, \ldots, a_n)$, where a_1 is the greatest sector. Given two linear grids of the same dimension, that is, $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$, we say that $a > b$ if $a_i > b_i$ for the smallest i such that $a_i \neq b_i$. This completes the ordering of all linear grids. For a given integer $N \geq 1$, we can map N to the Nth linear grid in the ordering.

Modern computers use binary format, and encryption is performed on the bits, not the characters. For linear grids, we can represent the text in binary format and place bits, as opposed to characters, into the grid. This approach has the advantage of limiting the appearance of word fragments in the encrypted text. For example, when characters are placed into a grid, two adjacent openings expose two-letter word fragments. However, when we place bits into a grid, it takes 14 successive openings to expose two adjacent characters, using the standard 7-bit ASCII format.

For further reading, the book by Singh [**8**] provides an excellent historical overview of cryptography. A detailed description and analysis of modern encryption techniques can be found in *Applied Cryptography* [**7**] and the textbook by Kaufman, Perlman, and Speciner [**4**].

Coding-breaking challenge We leave the reader with a passage of text that has been encrypted with a 5×13 linear grid.

WCOD LCKF HUCV TWOD OHUH KCUK YHHC ODCU IOOU ODFM UHUC KAWC OCHW OAWO CDCL DORG B

The decrypted passage can be found on page 398.

REFERENCES

1. Girolamo Cardano, *De Subtilitate libri XXI*, 1550.
2. Whitfield Diffie and Martin E. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, Vol. IT-22 (November 1976), pp. 29–40.
3. Martin Gardner, *Codes, Ciphers and Secret Writing*, Dover Publications, Inc., New York, 1972.
4. Charlie Kaufman, Radia Perlman, and Mike Speciner, *Network Security: PRIVATE Communication in a PUBLIC World*, Prentice Hall, 1995.
5. Rudolf Kippenhahn, *Code Breaking: A History and Exploration* (English version translated from German), Overlook Press, Woodstock, N.Y., 1999.
6. National Institute of Standards and Technology (NIST), *Digital Signal Standard*, Federal Information Processing Standards (FIPS) Publication 186-2, 2000.
7. Bruce Schneier, *Applied Cryptography* (2nd Edition), John Wiley & Sons, Inc., 1996.
8. Simon Singh, *The Code Book*, Doubleday, 1999.

Math Bite: Axial View of Trigonometric Functions

M. VALI SIADAT
Richard J. Daley College
Chicago, IL 60652

A typical way to picture the sine and cosine functions is shown in FIGURE 1, where a given central angle θ appears in its standard position in the unit circle of the Cartesian plane. Since the horizontal distance OM is $\cos\theta$, we may loosely call the horizontal axis the *cosine axis*. Similarly, the vertical distance OL is $\sin\theta$ and so the the sine function is associated with the vertical axis. An advantage of this approach is that students can reliably determine the correct signs for these ratios when θ is outside the first quadrant. You may be familiar with similar axes associated with the tangent and cotangent functions, but have you ever thought of a secant axis?

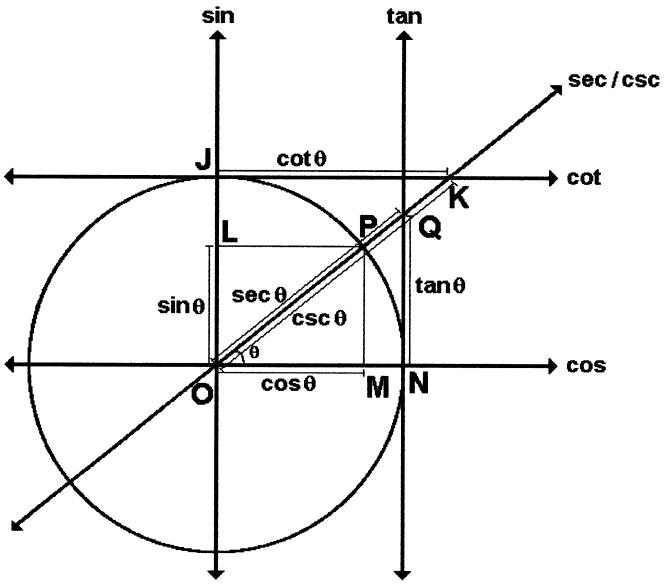


Figure 1 An angle in standard position in the first quadrant