Vernam Two Cryptographic Engine

Author: Dan P. Milleville, dmilleville@comcast.net 978-772-2928

This cipher methodology utilizes 4 pointers pseudo-randomly set initially and pseudo-randomly advanced between blocks. These pointers, pointing to unique streams within an 8 Mbyte key, are then Xor'ed together producing an **E**ffective **K**ey **S**tream (**EKS**) and then Xor'ed with the plaintext. The pointer pseudo-random advancement algorithm does not allow any of the first 3 pointers to be a duplicate of any of each pointer's past values within 1,000,000 blocks. As long as the plaintext is less than 128 Mbytes, none of the 3 pointers will ever have the same value as when any other prior block was encrypted. The fourth pointer is a mathematical construction of 3 byte accesses from each of 3 table accesses Xor'ed with the ciphertext block number. Both the sender and receiver can now possess a fixed key and still encrypt and decrypt text using the created-on-the-fly non-repeating pseudo-random **EKS**'s formed, just as the Vernam cipher engine does now.

This design is at least 4 times faster, and also experiences no loss of security, as compared to the AES. The same mathematical operator as the Vernam cipher (Xor) is used between the **EKS** and the plaintext. The **EKS** has multiple key solutions that do result in almost countless key sets that all translate correctly to the same **EKS** with no methodology that will ever be available to determine which one of these reconstructed key sets is correct. This is illustrated in Appendix A showing 5 of the almost countless reconstructed incorrect keys. When encrypting 128 characters, the 31,360 (AES Visual Basic version) programming steps executed by the AES are decreased to 1,280 for this Vernam Two. This results in at least a four-fold increase in speed that is needed in today's increasingly data-intensive and security-dependent world.

The 8 Megabyte (8,389,631 numbers) key is hereafter referred to as the **Key.** Four pointers are pseudo-randomly determined and reference this **Key,** each using an exclusive 2,097,497 byte segment of this key. This yields a total of 1.93 x 10^{25} possible **EKS**'s. The four streams of numbers extracted are equal in length to the block size, 128 key numbers in this design, 29 displayed below. Examples:

Pointer 1 = **1,710,426** Pointer 2 = **1,576,906** Pointer 3 = **189,945**

Xor'ing the vertical numbers from these four streams forms an **EKS** that will eventually be Xor'ed with the plaintext, fully detailed in Appendix A, black text key streams. The example in red is '7F' Xor '20' Xor '58' Xor 'D8' = 'DF'. With the attacker not knowing the 4 key numbers, it is Algebraic law that this single equation with 4 unknowns will remain verifiably unsolvable. Note that each stream accesses a different 2Mbyte section of the 8Mbyte key. The particular key segment accessedHere are 4 example key streams using the above pointer numbers and the resulting **EKS**:

```
Key Stream @8,002,650   - 51 F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1
Key Stream @5,771,722   - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945     - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @2,720,380   - DE 83 1A 0F D8 82 15 C9 D8 05 3C 51 43 F3 39 61 BE 14 86 C4 A1 EA 78 C5 23 D1 A0 E1 5C
Effective Key Stream 1 - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68
```

If the first pointer is advanced by 1 to 8,002,651, the following result is obtained. Note that pointer 4 is significantly changed because it depends on the table values that each of the other 3 pointers are pointing. So since pointer 1 would be referencing 3 different values, pointer 4 reflects this. Effective Key Stream from above is copied for you to compare:

```
Key Stream @8,002,651   - F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E
Key Stream @5,771,722   - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945     - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,405,013   - 72 63 46 38 F7 BE 65 F7 F0 E1 7A 93 D8 01 5F 62 F6 3F AA 44 78 77 7B 37 E1 B1 41 4A 74
Effective Key Stream 2 - 9B 5C 54 75 BF AB 36 9B F9 EC 00 6D 84 54 18 EB E6 AC 02 5A CC EE BD 5B 1D E3 6F F1 AF
Effective Key Stream 1 - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68
```

Two screen shots showing the app displaying the data above from the first example is provided in Appendix B. If pointer 1 was advanced to 8,002,650 through 8,002,659 the key stream shift, as well as the pointer 4 changes, would cause the first example above to show the **EKS**'s listed in Appendix C.

For the app to duplicate the first scenario, copy/paste this string into the app: 22,1710426,1576906,189945,209,215

Code/speed comparisons, AES vs Vernam Two

The following AES encrypt engine code was extracted from the Visual Basic version of the AES obtained from the internet:

```
For i = 1 To m_Nr - 1        ' 1 to 13
    For j = 0 To m_Nb - 1    ' 0 to 7
        m = j * 3
        Y(j) = m_ekey(k) Xor m_etable(X(j) And &HFF&) Xor _
            RotateLeft(m_etable(RShift(X(m_fi(m)), 8) And &HFF&), 8) Xor _
            RotateLeft(m_etable(RShift(X(m_fi(m + 1)), 16) And &HFF&), 16) Xor _
            RotateLeft(m_etable(RShift(X(m_fi(m + 2)), 24) And &HFF&), 24)
        k = k + 1
    Next
    t = X
    X = Y
    Y = t
Next
For j = 0 To m_Nb - 1    ' 0 to 7
    m = j * 3
    Y(j) = m_ekey(k) Xor m_etable(X(j) And &HFF&) Xor _
        RotateLeft(m_etable(RShift(X(m_fi(m)), 8) And &HFF&), 8) Xor _
        RotateLeft(m_etable(RShift(X(m_fi(m + 1)), 16) And &HFF&), 16) Xor _
        RotateLeft(m_etable(RShift(X(m_fi(m + 2)), 24) And &HFF&), 24)
    k = k + 1
Next
```

Executing a debug single-step of the 'y(i)…..' instruction caused it to go through 70 steps to obtain the data to Xor together to obtain the value of y(i). Considering that the 'j' loop has 8 passes to execute, 70 times 8 = 560. The 'i' loop has 13 loops plus the second loop, 14 times 560 = 7,840. This is excluding the 't = X, X = y and y = t' steps that would require more work than just a single instruction because these transfer entire structures. Considering that only 32 characters are encrypted by this function, and Vernam Two encrypts 128 characters per block, 4 times 7,840 = 31,360 steps would be required to encrypt the same 128 plaintext characters with the AES engine.

The following code executes the Vernam Two encryption loops, both Vernam and Transposition:

- 'n' is a temporary data holder
- 'ptxBuffer' is the string holder containing 128 characters of plaintext
- 'e1Key' is the Vernam Key Table
- 'p1' through 'p4' are the randomly set pointers that address each of 4 separate 2 Mbyte segments of the 8 Mbyte Vernam Key
- 'iSt' structure is the 'Midstream data holder'
- 'p' is the pointer for the midstream structure
- 'uNib' structure returns the upper nibble of the number
- 'lNib' structure returns the lower nibble of the number
- 'x16' structure multiplies the reference by 16
- 'tk' structure is the Transposition Key
- 'e2Off' is the randomly set offset value, from 1 to 255
- 'e2Ptr' is the randomly set Transposition Key Number holder
- 'ctxOut' is the string holder that is to contain the ciphertext output block

```
For i = 1 To 128
    n = Asc(Mid$(ptxBuffer, i, 1)) Xor e1Key(p1) Xor e1Key(p2) Xor e1Key(p3) Xor e1Key(p4)
    iSt(p) = uNib(n): iSt(p + 1) = lNib(n)
    p = p + 2: p1 = p1 + 1: p2 = p2 + 1: p3 = p3 + 1: p4 = p4 + 1
Next i
For i = e2Off To e2Off + 255 Step 2
    ctxOut = ctxOut + chr$(x16(iSt(tk(e2Ptr, i))) + iSt(tk(e2Ptr, i + 1)))
Next i
```

When executed, it does not jump to any functions to prepare data to Xor together – the data is right there to use and calculate. The first loop executes 8 instructions per loop, 128 loops, 8 times 128 = 1,024 instructions. The second loop doesn't call any functions either, executing another 128 calculations. 1,024 + 256 = 1,280. The 'ctxOut' location contains the ciphertext for output at the end.

Compare the AES's 31,360 steps to encrypt 128 characters to the Vernam Two's 1,280 steps – a 24 to 1 improvement.

The 2,048 random number system key

This design requires only a 2,048 byte key, but it is way too small for use during the execution of this algorithm. The key is made this size so that there is at least one representation of each number between 0 and 255 inclusive and not have to force the random number generator to include any missing numbers. Experimentation has determined that most of the time, 2,048 random numbers accessed will usually result in every number from 0 to 255 inclusive appearing in the list. If not, the regeneration of all 2,048 numbers is executed until this is satisfied. Once satisfied, the key is established as the key for this system. As you will see in Appendix E, the key is generated completely at random, no interference in the process.

Creating the 8 Megabyte Key Stream

Suppose you put a group of randomly selected numbers, plus 1 (255 becomes 0), on blocks, put the blocks in a drum and turn the drum to mix the blocks. You then pour out the blocks and align them. Do the numbers in that order, obviously different from the first arrangement, change the numbers into a block of non-random numbers? Rearranging the numbers, and adding 1 to each, using whatever is needed in different ways in a process that can be duplicated (pseudo) would produce a stream of 8,389,631 random numbers (only 256 numbers from the last set is used) from the original. The process consists of two pseudo functions:

1. Acting on every set of numbers inputed except block 4,194,304 (h40000), the creation of the key block is redone by copying every 4th number to the end of the new buffer and loading in backwards order in this order:
   - Numbers + 1 in positions from 0 to 2,044 (0 to 4,095, 4 to 4,094, 8 to 4,093, … 2,044 to 3,584), then,
   - Numbers + 1 in positions from 1 to 2,045 (1 to 3,583, 5 to 3,582, 9 to 3,581, … 2,045 to 3,072), then,
   - Numbers + 1in positions from 2 to 2,046 (2 to 3,071, 6 to 3,070, 10 to 3,069, … 2,046 to 2,560), then,
   - Numbers + 1in positions from 3 to 2,047 (3 to 2,559, 7 to 2,558, 10 to 2,557, … 2,047 to 2,048).
2. Working on block 4,194,304 (h40000), extract 256 unique numbers from the new block of 2,048 numbers, then use that set as a transposition key. Take that transposition key and use the key to transpose its own key producing key #2. Using the alternating algorithm as specified in Appendix E, do this 8,194 more times for a total of 8,196 transposition keys produced. Combine 8 of these keys (also as detailed in Appendix E) into one master transposition key of 2,048 positions. Use this master transposition key to transpose the 2,048 numbers + 1 produced by step 1 to produce the next block of 2,048 numbers.
3. Go back to step 1 and execute step 1 until a set of 8,389,628 random numbers have been initialized, a total of 4,027 additional executions.

Note: After the 8,195 - 256-position transposition keys are constructed, the transposition key numbers are copied to an area 256 locations above the current one. This use is described later in this document. Using the key in Appendix E, the code specified will input those 2,048 numbers and expand them using the above alternating algorithm.

In Appendix E, the data just from one execution of steps 1 and 2 above is provided. Testing has shown the resulting number stream created using the number set, also displayed in Appendix E, provided in this fashion produces the duplicate stream results between all combinations of the 4 sets in Appendix F. This should verify that the expansion of the original 2,048 number input key is successful in not duplicating that initial key anywhere within the key so produced, or any significant (6 or more numbers in sequence) stream anywhere within.

The Vernam 2 Algorithm – Transposition Engine 2

In addition to the Vernam Engine described on page 1, the Transposition Keys, extracted during the key expansion function, are then used as Engine 2 after the Vernam Engine to transpose the **nibbles** of the Vernam output. This quite literally shreds the Vernam Two output so that there will be no hope of any attack being able to be even started against this system. There is an average of about 16 of each digit in the midstream or Vernam Engine output and the attacker is going to have an impossible time trying to figure out which '8' (for example) in the input would be the first '8' in the line.

Just because the Transposition Algorithm has been broken now for decades does not mean it is useless. The breaking of the algorithm was made possible by plaintext as the input. It is used in this cipher design to 1) expand the key and 2) during encryption/decryption. It uses pseudo-random data as the input and transposes Vernam output **nibbles** instead of the Vernam output bytes. Due to the nature of the input to this engine and transposing only nibbles instead of bytes, the key cannot be determined by any analysis of the input and output.

Take this **midstream** from the demo system and look at the blue highlighted '8' digits, both in this and the ciphertext output below:

```
    A383238AC2D5FD77EB5A62FD228EE1009B95540AC43649574DE7E387167DEB6648182B758B5062
55CACBDF74800FD964A93416A94890ABB7BE95BFF593F8F39B6D533251091752537C5FD80F606E2DB
22A05CDDADBE80BF9180D582169DF77072AEDDF921430F2492E5FDBC0F20CCA5038C226330A615428
D9F56B8AB352BC18
```

Here's the ciphertext **output** stream with the '8' digits also highlighted so you can see the transpositions:

```
    803A98B5D22547ABDDC5FC1A55202577A23423F4785CB6D2839F32BDD23F913530CF2F9F037AB94
A779887846DA118949E5AB22B093C4B55BE6F978E5888E80ECCA019DB9B1A5E80D23100D35050B12
A8883B5564B25D006B2B6F8AD5050624AC2FD4FDC94EBDF07391E2D63F64CDE6BEF02695723FCF9ED
F16506267757AD16
```

Take the first digit '8' in the ciphertext line highlighted in **red**. Which '8' in the midstream would be that first '8' in the ciphertext block? Without the '8' in the midstream being **red**, you would not be able to apply any methodology to determine which '8' was transposed to that first position due to the fact that it is 1) pseudo-random data and 2) transposing nibbles not whole numbers. The use of the Transposition Engine here is safe since no methodology could ever be developed that would be capable of determining (for example), which '8' in the midstream is that first '8' in the ciphertext block.

An offset is also determined into that particular transposition key. Remember in the key expansion algorithm it was noted that the transposition key was also copied to the upper 255 locations? This is why. An offset into this table is pseudo-randomly determined at the start where the start of access to the table is made, from 0 to 255. Even if the attacker were somehow able to reconstruct the correct key, there can never be any indication by any numeric processing of the exposed key as to where location 1 was located in that key. In the above example, the offset was 209. Locations 209 to 465 would contain the transposition key used to transpose the midstream to form the output ciphertext. There could never be any method of determining that position 48 is actually position 1 of the key used with an offset of 209.

For the next block of ciphertext, the pointers are all advanced using separate pseudo-random technology for each pointer that combines both the values of potentially all the pointers and Vernam key values in determining what is used for pointers for the next block of text. In this way, no additional modes of operation are needed for this design. It also checks to make sure no pointer uses a pointer value used by itself within 1,500,000 blocks.

Extensive testing of the advancement algorithm has shown that there is no duplication of all 4 pointers used for any ciphertext block even through 100 million simulated blocks of plaintext. Since the 8 Megabyte key has been shown to be random in nature, Xor'ing 4 different pseudo-randomly selected streams from this key from varying 2 megabyte sections of the main key and Xor'ing them together to form an Effective Key Stream (the **EKS** described on the first page), they can be described as non-repeating, the same as what is required by the original Vernam system. Because the key is fixed, formed by a pseudo-random process that has been proven to repeat its process by both sender and receiver, the transmission of the key with the ciphertext is no longer required. With 2,097,152 possible values for each pointer, the number of **EKS**'s possible is $2{,}097{,}152^4 = 1.9342 \times 10^{25}$.

The communication protocol of pointers to the decrypt engine

Since 5 of the 6 pointers are randomly selected for each ciphertext processed, these numbers need to be communicated with the decrypt engine using an algorithm to hide them from any attacker.

Starting with the Encrypt engine, the 3 pointers need to be inserted. For the first block only, a simple sum of the ASCII characters in the 120 byte plaintext block is obtained, converted to hexadecimal and inserted as the first 4 characters of the plaintext. After the Vernam Engine has executed, producing a stream of pseudo-random midstream numbers and split into 256 nibbles, the 3 pointers are split as follows:

1. The first 5 nibbles are mathematically combined with the top bit cleared to form a single pointer from 0 to 1,048,575 (1FFFFF hex) inclusive.
2. If the value of the location in the Vernam Key is less than 6 or greater than 235, the pointer is incremented and the next value is obtained, repeating until a number within the acceptable range is attained. Example: The first 5 nibbles: 0, 9, 4, 0, 6, forming 9406 Hex or 37,894 decimal. That location contains 160, an acceptable number.
3. Each of the first 3 pointer's first 16 bits are split into two nibbles each and the 8 nibbles are inserted in the midstream starting at the pointer obtained, in this case, locations 160 through 167.
4. The top 5 bits of all 3 pointers are combined into one number (forming bits 0 to 14 of 15 bits) and then that number is split into 2 nibbles and those are inserted into the midstream as well. In this case, locations 168 and 169.

Example using the sample supplied on the first page – the pointers 1 - 1,725,418, 2 - 1,260,439, 3 - 774,161 were changed to 21,482, 1,551, 44,714, then combines the top 5 bits from each pointer (#1-) 26,624 (6800h), (#2-) 992 (3E0h) and (#3-) 9 (9h) for a value of 27,625 (6BE9h). These 4 numbers are broken into the 8 nibbles and inserted in the midstream for the transposition engine. The Transposition engine executes the repositioning of all the nibbles and pairs of nibbles are combined to form the characters of the ciphertext output stream. The transposition numbers themselves, the key number and offset, are communicated by two extra characters inserted in only the first block of ciphertext, making it 130 characters in length instead of 128. The process involves the following steps:

1. The first three characters of the ciphertext line are mathematically combined to form a pointer into the Vernam key table.
2. A loop executes if the location pointed to in the Engine 1 key contains less than 5 or greater than 125 incrementing the pointer if not, exiting if it is within range. Example: the first 3 characters are 69 (45H), 102 (66H) and 85 (55H) forming 456655H. The upper 4 of the hex is stripped, making the pointer 353,877. That location in the Vernam key contains 193, not within range, it advances the pointer to 353,878 that contains 15, that is in range. The function passes that back to the encrypt engine.
3. It then inserts ASCII characters equal to the 1) Engine 2 pointer and 2) zero-based engine 2 offset in positions 15 and 16. Since these numbers are created randomly and inserted in what is defined as pseudo-random data, there is no way any attacker could be able to consistently determine which two characters identified these values.

Then the ciphertext output line is written to the ciphertext output file. Since the pointer handlings for subsequent ciphertext blocks is handled in a pseudo-random fashion, both encrypt and decrypt engines do stay in lock-step with each other in processing the remainder of the plaintext, so no further insertion of register data is needed (steps 1-4 and 1-3 above).

In the decrypt mode, for the first block only, inputting the first 130 ciphertext characters, the second 3-step algorithm above is executed, extracting the two characters, and bringing the block to a 128-character length. Using the pointer and offset numbers extracted, the transposition operation is reversed, constructing the midstream for the Vernam Engine.

For the first block only, the 4-step algorithm is used to extract the nibbles used to identify the 3 Vernam pointers with the 4th pointer then reconstructed by Xor'ing these pointers plus the Transposition pointer and offset. The Vernam engine is then executed, forming the plaintext string.

To help aid in the decrypt engine determining that it has been successful in decrypting the ciphertext, the simple sum of the ASCII of the characters from character 5 to the end are summed, converted to a hex number and compared to the first 4 characters of the first decrypted plaintext block. If the characters are hexadecimal and they equal the sum the decrypt engine reformed, the process continues. Otherwise the error is discovered and the decrypt process aborts and this finding is reported to the user.

Appendix A

The black numbers on this page are the real key numbers; the red numbers (on the next 5 pages) are randomly generated key numbers, except the last key stream, calculated to duplicate the Internal Data Stream of the original sample here. Notice that all respective ‘Plaintext ASCII Hex’, ‘Effective Key Stream’ (both black and red), ’Internal Data Stream’ are the same on all 6 pages. Definition: The ‘Effective Key Stream’ is the Xor of all 4 Key Streams.

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @8,002,650- 51 F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E
Key Stream @5,771,722- 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12 96
Key Stream @189,945  - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97 7B
Key Stream @2,720,380- DE 83 1A 0F D8 82 15 C9 D8 05 3C 51 43 F3 39 61 BE 14 86 C4 A1 EA 78 C5 23 D1 A0 E1 5C FB
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A4 4A 86 A6 AA 8B 52 DE 84 6B 3F 74 75 F2 06 0E 28 4E 27 33 A6 3D A2 20 B7 08 85 96 5A A0 AD F2 8B FA 0A 5A E4
75 2B BD EE 68 58 EF 21 B8 23 8B 54 B9 2A 6B 88 C0 28 FA E9 27 B4 77 FD 83 D7 B2 1D E6 B0 01 BB 08 10 CB 55 A5
2F 3B AF 14 1E 3E 89 7B 1F 1A 09 CC 1C 8F 16 0F 84 31 6E 47 27 DE 39 2B F4 0F BD AB 52 2D CB 24 8D AE AF 76 69
E4 70 B3 29 17 58 C3 7F FD 5D FB 1C CE 11 56 2A C2 BA 1E 59 87 71 B4 6C 61 F3 B6 92 5A 51 47 E8 05 86 20 82 1D
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
77 60 D4 DC 6E A7 82 F5 1C 60 23 29 67 5E AC 67 38 3E BC 5A 92 C5 BB 70 70 73 C7 20 24 A1 34 6C C3 9F A5 BB A7
8F 0D F0 24 3E D4 E1 7D 7B D8 88 4A 7A 25 72 B4 8D CF F6 E5 00 82 D7 51 17 F6 CF 17 77 BA 27 A7 EE 49 F5 13 4C
EE D9 9B 3F EC 71 B3 13 C4 CE 47 B6 A8 98 8E 10 65 74 F5 46 68 B2 CC 43 FA B3 86 49 24 B3 CC 66 7F 69 80 99 00
CD F8 FE 1B FC 8B 87 F8 A4 60 15 B9 75 0E 93 87 DE CD 91 7E 32 9E A5 A0 09 B1 78 F9 3A 04 D9 3A FF 5B 34 CC E9
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
CA D8 97 3C 84 0F 25 01 0E CB E6 9A 88 80 CE 82 2C BD 71 0D C5 97 D4 03
AE 85 AA DD 2C 99 77 E0 1F E7 A3 22 C3 03 B4 BD 0C 53 61 39 77 34 AA 06
5C 8F F5 B7 05 1C 3B 63 CF 09 59 59 BC B6 99 B1 27 2E EA 76 91 BC 48 B3
C6 1E 53 9A 77 C1 93 1C 76 CD 3D 06 B6 8C AD 38 13 DF 54 12 0B 62 14 51
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E

----------------------------------------------------------------------------------------------------------------
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

For the app to duplicate this scenario, copy/paste this string: 22,1710426,1576906,189945,209,215

UNKNOWN Vernam key – assignment is to reproduce the EXACT key – solution is at the end of this document

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
Key Stream @?,???,???- ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
Key Stream @?,???,???- ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
Key Stream @?,???,???- ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C7 66 7F 7F 19 F4 AC 41 C4 F0 8B A0 E6 DC 79 31 12 FD 5D 60 95 35 6A 2F D9 8A 42 F1 DE FF

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
D1 0F 40 90 8D A8 A4 AD 54 78 A7 B4 FB 72 3D 70 E0 0E 2E E8 C4 7E D1 10 C8 07 38 20 3F 95 16 A4 78 CA C9 4A E5

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
53 EB BF 4C D4 E0 49 0B A4 EF 7E 1C 4D 1F 69 6E 80 56 EE D2 D4 15 01 82 E5 D7 64 0E 02 9B DF 6F D1 F0 8A CD 2E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || ||
BB E6 DA DF 43 79 9E 6F AB 22 F6 87 8C 1E 87 94 77 CD 06 37 5B F5 EA B3
```

Look at the data on page 6, how the bottom shows the format of the original Vernam system, then the top of the page, how it is mathematically identical to the bottom. This design uses 4 randomly selected streams from a permanent key and Xor's them together along with the plaintext. By utilizing the **key** **table** **directed** pseudo-random pointer advancement function, this methodology has been proven to produce a non-repeating Effective Key Stream. The number of streams is limited to $2{,}097{,}152^4 = 1.93428 \times 10^{25}$. As illustrated above, even if it was found that two streams had all table pointers identical, it would not give the attacker any clue as to how to obtain the key. Algebraic law would give them no recourse except to produce more keys that do work than there are AES keys that do not work.

Question: How does executing 31,360 instructions, as illustrated on page 2 with **no** proof of security, compare to 1,280 instructions executed **with** proof of security? Maybe the author's estimation of '4 times faster than the AES' is conservative if this system were coded in another language other than Visual Basic. Just the increase in speed alone should be reason enough for this Vernam Two cipher to be used by the security community considering the significant increase in our society's use of digital information. Considering the ciber threat in this world and ever-increasing technical abilities in attack software and hardware, the security world should not wait for the AES to be broken to start looking for and adopting a replacement system. Replacing it with a more complex system than the AES instead of one that has mathematical proof of protection would just keep the game of leap frog going for an extended period of time and further lengthen the time to encrypt a particular block of plaintext. It is time to end this once and for all and approve a permanent solution to this security issue.

#1 (randomly created incorrect key that does work)

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- 60 2A DE 2D BD 48 10 40 DE 5F 25 7E 51 21 79 68 FF BA 69 79 89 1E 50 90 E1 94 BE A0 26 83
Key Stream @?,???,???- 0F B5 AD 0C 9E D8 3D F8 9D 8B F8 58 D0 EC B5 59 E8 A4 4E 3A 77 08 A5 0B 16 73 F6 A9 D3 E1
Key Stream @?,???,???- E4 EF 1C AE FE D3 B6 3C 9B 8D 5E 8D 24 39 BB 0F 31 48 73 F1 C0 2B 0C F4 B3 43 05 70 8F 09
Key Stream @?,???,???- 1D 3E 9C 62 CE 3F DE 63 07 CC 4F 49 79 27 7E 47 5F 41 B0 5C C9 14 70 12 B7 0B 27 EF 12 23
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FA 97 C4 7B BA 8B CC 86 21 2A 90 B0 F9 63 83 D4 31 51 4F A8 99 67 36 0B DB E1 AD 8D BC 69 35 3F 94 09 7D BC 0B
5F F9 E0 CF AD 63 79 52 68 91 16 76 20 69 D1 D9 F8 48 19 87 73 7F 97 17 7F 12 60 06 BF AF 5D 52 1C EC F6 5A 8D
35 A9 73 79 62 53 56 C3 25 62 84 34 FC 16 5A 61 B0 46 40 96 E9 EC E7 7D 0E F1 C4 12 62 3A CC C6 5C F3 71 F1 46
8A ED 70 B8 BE 0E 14 EC B2 D6 44 02 3B 5A 25 CF D7 B2 BB 7D 22 D2 1E FB 0B 21 35 2B D5 90 84 2E DF D4 B4 EC F5
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
50 B5 77 15 C7 34 47 30 96 4D 2F DB 72 6E 72 59 C9 79 D7 A7 30 67 38 BA 0C 19 50 FD 7F 81 C5 09 95 F1 73 A6 AF
C5 05 F4 25 54 F9 D0 5E D2 49 0C A5 CD 1B 17 25 AB 34 CA D7 5F 23 C3 CA 69 A8 04 AE 12 6A CD A7 E9 D9 B6 4F 78
19 DC 7D 18 4D 6B D0 96 D7 6D F9 A7 F8 1B 1E F8 75 DA 25 D1 60 AF A1 36 28 02 FE 41 E3 97 F0 BF DA C9 18 08 55
57 20 BF F4 9E 2F 10 9B 94 7F 23 B5 87 83 B8 C0 19 DF 16 26 C7 80 5F 84 D9 34 5C 95 C3 D0 FE 86 0B 05 39 1C 80
54 CF B6 01 A8 A3 40 A2 81 AA E7 47 BB 41 60 1F 4A D7 4A 6D 45 49 78 03
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
C2 FB 02 2E 29 8C 30 63 54 F0 E3 B6 CC D9 6A 16 06 6C EE A0 4A 0E EB 92
60 0F 0C 4B CF 7C 93 22 06 CB BC 75 E6 18 88 28 74 23 70 8F 4E F0 EF A4
08 F7 23 A8 94 18 19 7D 7B 79 99 63 D0 39 CC 97 2C 87 7A 12 69 CA 5E D2
54 CF B6 01 A8 A3 40 A2 81 AA E7 47 BB 41 60 1F 4A D7 4A 6D 45 49 78 03
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

The above example with the 4 bogus key streams Xor'ed together into what is displayed below as the 'Effective Key Stream' is Xor'ed with the plaintext to produce the 'Internal Data Stream'. Notice the Plaintext ASCII hex and Internal Data Streams are identical in both the set above and this one and, most importantly, the Xor of the 4 streams above is identical to that on page 5:

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

#2 (randomly created incorrect key that does work)

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- 34 4B E8 51 7C 29 0D B4 D5 B2 55 6E E4 7B 11 43 64 55 53 B4 8B 98 E7 63 9F AF 94 C6 D3 E4
Key Stream @?,???,???- BC C9 04 7F DD 3B 8B 8B 82 2A 7B 87 32 04 CF D4 23 89 44 A7 2B E0 62 C2 E6 F9 C6 77 28 24
Key Stream @?,???,???- 5A 51 BF B6 CF 94 03 B6 8B ED 9C 04 AD 7B C5 70 3E 74 78 4E 73 CE 4F 41 21 2C 9C 49 80 0A
Key Stream @?,???,???- 44 9D A0 75 7D FA C0 6E 03 E0 7E 0F A7 D7 12 9E 00 BF 8B B3 24 9F 43 9D AB D5 A4 6E 13 82
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
94 0F 66 6D 50 5A FE B6 60 18 41 EF C1 BA B5 D8 58 A2 D9 CC A7 46 84 0B CD A5 7D 4C C3 EE 5A 82 D5 6E 9C EE 5E
B5 DF A5 F2 DE 56 DE 4B 6E 39 4E D0 59 E2 D4 73 E6 E4 63 53 7B 83 E9 DD AB 76 7C 1F 7D E0 CB 6F FE 8B C2 3C 08
CB 0A 23 2C F6 5F 37 43 C9 1F FB E1 98 2D 7F 9F BE C5 A0 8B 38 05 43 D7 81 B9 D0 BF AF 6B F1 BB CE AA 8C E5 7B
F0 F0 C7 C6 B3 E6 E0 45 19 31 B2 2E 1E 33 33 97 AE 6E B7 D0 C5 E6 76 9B 46 49 ED 5E A5 09 40 D3 EE 8D 9C CC 18
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
BA 0B 65 C7 1E 4F E2 C1 44 D0 4E C1 28 CE 8E BC 3B C3 CD 24 D5 F8 0B 21 E0 65 E6 14 C2 CE A7 48 8A 1B 8F FA 5E
66 7F 33 54 40 74 95 BE C6 DD D7 D7 BA 65 7B B0 75 13 2D E4 9F 79 21 E6 2C 1F AB 8C 9A 2D 1F DC 68 58 EA 16 D9
9A FD AC C4 BC 98 BA 3F FE 5E 21 69 05 FE CD 40 B9 0E 25 23 12 8A 03 E9 21 8C D6 3F 61 4D 15 46 3E 5B E6 6C E6
9D C5 BB 8B A2 2A 9A 23 7B 45 41 13 57 B8 FB 08 F9 96 EB 64 90 60 2C EC 79 71 6D 20 74 02 AB 45 71 FC 67 7D 63
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
F6 62 8C 24 AA 35 04 6F 3B 74 CA AB C1 36 A2 8C 09 C3 E7 B3 6D 66 09 A2
0E F1 01 E9 E4 29 79 90 3F 77 39 35 41 C9 E5 75 F6 74 5A 15 BF D0 39 44
1C 0E 8A 4F 82 1C 3E 88 10 D0 8B 44 0C D1 AE A3 05 8F 42 43 88 75 97 86
1A 51 9C 4E 16 4B B9 E9 BC 3B 59 3D CD 97 A7 EC EE 27 51 B5 72 BE 85 87
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

The above example with the 4 bogus key streams Xor'ed together into what is displayed below as the 'Effective Key Stream' is Xor'ed with the plaintext to produce the 'Internal Data Stream'. Notice the Plaintext ASCII hex and Internal Data Streams are identical in both the set above and this one and, most importantly, the Xor of the 4 streams above is identical to that on page 6:

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

#3 (randomly created incorrect key that does work)

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- 79 FD DB 8E 46 29 7B 87 9E B1 F8 46 5C 54 68 E9 1D A6 80 F8 48 CA BE C3 5D 73 D5 FB 7D 18
Key Stream @?,???,???- 67 66 9E 83 5A 82 E9 3D 3C AD 24 9E A7 99 B9 BC F6 79 3B 6D 23 D0 E0 34 E1 7E 40 6D FA 07
Key Stream @?,???,???- 3F F5 65 86 17 7A C0 2C 78 6F BD F0 06 F0 0D 43 22 92 E8 52 4E F9 99 45 DF BC 4A B4 E9 78
Key Stream @?,???,???- B7 20 D3 66 18 AD 17 71 05 E6 AD CA 21 EE D5 6F B0 5A B7 29 D2 CA 4E CF 90 1E B5 B4 06 2F
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
59 8F 3A 76 AB C9 A0 4B 25 A9 83 50 89 7F 0A 76 A7 CE 7B 65 4D 60 48 31 73 A5 5A 5E A7 51 9D 17 B5 35 CB A9 9A
4F BB EA 26 98 B2 BB 91 BB C0 22 0E 29 61 DB B5 1E 82 41 87 DE 35 30 2F 33 96 1F 17 25 FC 78 AE FB 64 3A F8 72
10 69 67 32 39 5C AA F2 D4 EB 03 9C 81 59 B3 10 7B 8A 12 85 E2 FC FE 22 4D 07 87 DE 8F 58 D7 99 C8 AA 67 7A AF
1C 77 90 17 C1 92 46 D3 94 8D E4 32 3F 01 4F 70 6C 2B 85 A3 50 8F DE A6 AC 17 FE 25 B9 99 12 A5 8D 39 D8 D0 72
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
5C 7D 45 AB 68 6E F6 01 2F 11 E0 27 91 56 BA 06 9B B8 8D 10 E1 8A AE AB 18 83 F9 DF F2 86 4E 43 3E B6 51 24 3F
19 D1 55 3C A8 35 96 BB 03 41 3D FE F3 54 13 FC C2 2E EF 0F 29 03 C4 B3 77 E4 EC 7F 02 4F B3 63 D3 14 27 23 8C
C7 5F 8A 1E BE 4C 91 DE 65 09 30 B4 12 E7 3F AB 95 1E 69 A3 78 77 04 19 7A 3A 35 38 48 55 44 20 7D CE 3C 76 DC
59 BF DB 55 3E 9E A6 07 4E 4F 14 01 B0 08 55 15 C2 C0 25 3B 78 95 6B C3 81 DA D6 1F F5 30 BF 97 3D 88 AE 8C 6D
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
B9 5D F7 4B 4B C8 01 A8 06 5D 43 A2 70 16 AB F0 45 19 79 48 06 F6 EE 40
6A 1D 13 9C 51 87 13 A6 2A F4 CD 2D 94 67 2E 84 88 5D 22 F5 B7 18 93 2F
6B C7 2C CB 9E 47 91 13 B6 38 9A D6 43 C4 72 D0 C9 A7 40 D2 A9 45 38 7A
46 4B 53 D0 5E 43 79 83 32 79 35 BE E6 0C B9 12 10 FC B5 3F 30 D6 67 F2
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

The above example with the 4 bogus key streams Xor'ed together into what is displayed below as the 'Effective Key Stream' is Xor'ed with the plaintext to produce the 'Internal Data Stream'. Notice the Plaintext ASCII hex and Internal Data Streams are identical in both the set above and this one and, most importantly, the Xor of the 4 streams above is identical to that on page 6:

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

#4 (randomly created incorrect key that does work)

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- E7 5A D2 6F B4 BD 40 6F 5A 70 D8 3C 35 A6 2C AC 3D E5 75 42 25 09 48 5A B9 9A 9A 97 74 4F
Key Stream @?,???,???- AF 31 46 D6 4F 80 AA BC 2C AC F1 02 A7 49 53 D5 42 FC 1B 5B 52 59 72 2D EB 00 70 15 08 F5
Key Stream @?,???,???- A1 98 36 E0 6C 04 C0 35 22 73 7E 1D 56 9A F9 00 2E F7 7F 5B 5A 3D C2 52 52 00 B1 76 D3 A1
Key Stream @?,???,???- 7F BD 51 B4 84 45 6F 01 8B 3A 9B C1 18 A6 8F 00 28 F9 F5 AC DA 44 71 58 F3 35 31 62 C7 53
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
F6 8D 60 E2 FC AA 33 15 1D 6A A6 04 94 BA 17 AC A8 AC 61 BC 1F 5C FB 9B 05 21 4F B9 9E F8 5C 19 A8 2C 8F 1E D5
5F 0B FA 40 52 6D 12 C5 E6 7E 91 65 56 21 CF 54 C5 D6 AB BC C6 F5 32 93 30 28 92 B9 20 2E 40 55 4E CD 67 70 94
96 D3 9B 7A 70 3D 62 4D 92 DB 1C EA 7C F4 6E 1F F8 F8 6D E4 78 D1 9D C7 AF 6A 28 94 6E 80 5E 49 C0 A8 6D 38 48
25 7F 26 AD 15 4F B4 66 B7 C0 6D 7B A0 29 9B 44 3B 6F 0A 20 80 5E 0C 55 3B 40 C9 26 64 3A 62 80 2D 8B CB AD 3C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
4A DD 95 87 44 24 03 CD E3 C3 30 9E 71 77 F4 54 A7 78 40 6B D4 D3 6E 7A 4E 0A CD 71 1E E9 67 13 E6 AB 15 B6 15
18 93 D2 BB E2 9F 1C 87 82 DC 27 06 C8 82 9B 14 3E 5A D2 32 A2 AC EE 88 A3 66 38 37 DC 8E BB E3 0D EA 20 80 A0
A1 61 7D D9 DB 5E A2 56 47 1B 90 CC 60 A1 B9 CD 79 45 83 0E 48 E2 F6 43 B3 5B A9 3A D9 50 E9 36 64 1F 2B 8E 0D
28 63 7B 39 3D 6C EA 7F 21 12 7E 38 19 B9 15 C9 EE 2F 3F D0 F6 F6 73 73 CA B0 AA FB 56 9B 33 51 22 BA FA 45 BA
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
4D C5 3A 4E 30 0A 47 14 F5 E4 CB BA 94 D0 B2 8A BE 13 E6 33 75 42 88 78
EC CC 5C 5B E3 37 6D 61 5F 8F DC 6E E4 8D D1 52 D3 96 80 AE 94 15 73 EE
1E B4 85 6F 16 85 6B 10 02 8C F4 AC 08 09 D7 C4 B7 3D AA 66 83 59 FC 3E
41 71 78 B6 1F F3 BB FB 00 0F C2 9F 39 ED FA AA CE A7 62 AB 4A 73 25 4F
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

The above example with the 4 bogus key streams Xor'ed together into what is displayed below as the 'Effective Key Stream' is Xor'ed with the plaintext to produce the 'Internal Data Stream'. Notice the Plaintext ASCII hex and Internal Data Streams are identical in both the set above and this one and, most importantly, the Xor of the 4 streams above is identical to that on page 6:

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

#5 (randomly created incorrect key that does work)

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @?,???,???- 88 28 8B A7 D2 B3 30 97 62 07 EA CF 37 F8 75 11 A4 AD C2 2D 29 6C 76 F1 91 A7 A9 55 4C 47
Key Stream @?,???,???- 47 79 2C E5 74 FE 79 54 68 06 32 3D 94 43 1D 11 73 71 53 0B F2 ED D2 71 23 93 48 0C 22 D7
Key Stream @?,???,???- A5 ED 88 CF C1 A9 B9 86 BA 4A 8E CF EF 21 EE 4A A3 6F B9 32 B9 8D 24 A7 B1 57 57 40 E2 DD
Key Stream @?,???,???- FC F2 DC 60 74 98 B5 A2 6F DE 9A DF 90 49 8F 33 0D A4 CC FA 95 25 09 5A F0 CC DC 8F E4 05
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DC E7 0D 38 3A 73 A7 35 13 F7 24 D4 95 C5 27 D4 CC F9 83 A5 DB 1D 1B 37 7C 1A FC 07 B5 62 B8 67 DB 5B 04 0E CC
1E E5 46 26 C3 41 8C 7E 3B DE 40 4C E7 6E EC 85 95 53 85 07 CA 9C D3 31 4E 40 20 C0 F8 99 25 EC BB 23 31 C3 F1
48 D0 F9 4C D9 77 B2 19 30 EC D4 6F 57 09 72 DA 14 97 5F 35 09 5B F1 D1 0E 60 FA E6 CD 86 04 3D 9F 4A CB 1E A6
90 F8 95 27 EB F0 6E A9 C6 CA F6 07 3B E4 94 28 E3 D0 F4 53 39 FC 61 4D 9D 19 1A 93 34 11 B9 33 F4 F0 B0 28 AE
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
0D 95 DF 68 0F 37 FD 50 AE 2E 0E E2 5F D0 CE 74 67 62 D7 CF C8 33 AF EE 6D F7 12 F7 99 65 81 79 8C 13 0B E7 77
74 9A C4 E6 7E 69 BF A8 04 40 EB 0D 5A 8E 48 04 61 F6 36 1F C1 FB 56 61 67 F0 A1 44 41 1F C6 0D 38 37 C7 54 6E
D7 FF 77 2B 4D 3A 4E 39 51 3F 69 79 D5 75 3B 50 28 A3 10 37 40 EC 7B 59 45 0C FF 72 61 D3 FD E6 C7 E2 DD 20 01
75 BC 2D 79 7C ED 5B A2 FC 47 75 FA 10 C6 7E 64 20 7F DF 60 81 4F 87 14 DB 8C BA 46 F4 05 BC 05 DE 22 F5 6E 1A
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
90 5A 55 A2 C3 00 71 DB 82 02 C3 22 76 A4 28 91 2D 77 8C 90 80 A0 E7 39
FB D6 CE B9 28 6C 5C 5F A6 AB 59 12 F3 F5 1F 2D CB CD DA 30 1B ED 91 79
A5 80 2D 7E 90 E4 98 5B 9D 18 87 70 05 63 16 9E 3D AF F3 57 33 3B 3A 6D
30 C0 2D A9 A1 C3 4F 41 11 59 3C A7 C1 8B 6F 94 CF 0A 0B A7 80 0B 6E CA
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

The above example with the 4 bogus key streams Xor'ed together into what is displayed below as the 'Effective Key Stream' is Xor'ed with the plaintext to produce the 'Internal Data Stream'. Notice the Plaintext ASCII hex and Internal Data Streams are identical in both the set above and this one and, most importantly, the Xor of the 4 streams above is identical to that on page 6:

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1A 2A 27 75 CB B5 F7 FB DE 0F 46 F0 1E 46 2D A3 AE ED AD C4 21 26 58 9A A1 23 3C B2 B4 6C 20 85 0B C2 4E FB 35
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
68 4B 53 10 EB C1 9F 9E AD 6A 66 83 6A 23 5D D0 8E 82 CB E4 55 4E 3D BA 86 71 5D DC D0 03 4D A5 48 AB 3E 93 50

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
DB 4C 41 DC 40 89 57 63 07 16 F9 6C C0 ED C3 44 0E 48 2E 87 C8 6B 05 C2 94 87 F6 87 4D AC 06 97 AD E4 E4 FD 02
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
A9 6C 0E A9 34 F9 22 17 74 31 D9 01 AF 89 A6 64 7B 3B 47 E9 AF 4B 50 8C D5 D3 A2 C6 0E E7 47 D5 E1 A1 C4 BC 6E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
FE CC 9B CC DA 4B FA 9E A8 E8 21 E7 41 B9 4E B6 14 1F AE 50 28 7D 22 E7
|| || || || || || || || || || || || || || || || || || || || || || || ||
99 A9 F9 BE BB 22 99 BE C4 89 56 C7 27 D6 3C 96 67 7A CD 25 5A 14 56 9E
```

UNKNOWN Transposition key – assignment is to reproduce the EXACT key – solution is at the end of this document.

For example, notice the red ‘B’ that starts the ciphertext line. Knowing that the ‘Internal Data Stream’ is defined as pseudo-random, which ‘B’ character in that line, also highlighted in red, is the first ‘B’ in the ciphertext line? This question would have to be answered for every single nibble in the ciphertext line. With the Internal Data Stream being defined as pseudo-random, there is no methodology that would be able to correctly and consistently arrive at the correct answer.

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C7 66 7F 7F 19 F4 AC 41 C4 F0 8B A0 E6 DC 79 31 12 FD 5D 60 95 35 6A 2F D9 8A 42 F1 DE FF
                       -------------------------------^

  Transposition      - ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
 Key Number ?????    - ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
 Offset Number ???   - ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
 Ciphertext Output   - BD EB D9 62 CD 09 00 4E 8D DC 83 77 3D 0C 8C 56 4E 0B 8A 34 77 A0 22 F7 34 A7 A0 46 3A 85

                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
D1 0F 40 90 8D A8 A4 AD 54 78 A7 B4 FB 72 3D 70 E0 0E 2E E8 C4 7E D1 10 C8 07 38 20 3F 95 16 A4 78 CA C9 4A E5
---------------------------------^---^

?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
2D EC 90 24 8F 09 E6 DD BE AC E6 D5 14 41 9F 7C 9F BD F9 27 8F D7 13 D1 4F F8 7A AD 41 60 5B 44 9B BA D0 87 FE

11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
53 EB BF 4C D4 E0 49 0B A4 EF 7E 1C 4D 1F 69 6E 80 56 EE D2 D4 15 01 82 E5 D7 64 0E 02 9B DF 6F D1 F0 8A CD 2E
----^-^---------------^-----------------------------------------------------------------^

?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FE 16 6A E7 C5 CE 35 14 EE AD 9F DA 55 F8 CE 1F 7C 0E 4E 1E 4A 52 17 05 B1 90 43 90 FD 76 9F D7 70 02 46 B7 67

22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
BB E6 DA DF 43 79 9E 6F AB 22 F6 87 8C 1E 87 94 77 CD 06 37 5B F5 EA B3
^^-----------------------^-----------------------------------^

?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
|| || || || || || || || || || || || || || || || || || || || || || || ||
6B FF 94 55 EB F8 18 F4 6D 4F 81 C2 A2 0F F6 AE 81 00 EA C1 3E 86 2E 2D
```

This shows the Transposition Engine #2 operating on the Internal Data Stream from page 6. The numbers above the Internal Stream are the position number in vertical format. The numbers above the Ciphertext Output are the position numbers in vertical format in the Midstream where that hex digit was obtained. The ciphertext pairs of hex digits are combined to form ASCII characters, forming the ciphertext output. So each ciphertext character contains nibbles from two separate characters from the output of the Vernam Engine.

On the next 5 pages are random reconstructions of Transposition Keys that will reconstruct this midstream into the ciphertext in this example. The Internal Data Streams and Ciphertext Outputs are all identical in this and all 5 bogus key examples, clearly illustrating that, like the Vernam Engine, multiple key solutions with no methodology to eliminate the incorrect keys are also possible, drastically compounding any attack attempt. This is excluding the lack of any ability to determine the 'Offset Number' into the key, vital if any key reconstruction was ever attempted.

Just to show you what the Transposition engine does to the Vernam output, look at characters 29 and 30 in red below (squared). Notice that digit 29 is moved near the end of the line at the bottom (arrowed), and 30 is over to the right (also arrowed). Notice too that there are many 'B' and '8' characters (all in red) in the ciphertext line and without the key it is anyone's guess as to which ones were the subject digits 29 and 30. The algorithm that broke the Transposition algorithm cannot be used here since the input is defined as pseudo-random, and because nibbles are transposed, not bytes.

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

  Transposition      - 22 11 1      2 2   2 21    1  21 22 12 22 2   2 21 21 1   2 1     21    2  1  11 1    1 12
 Key Number 209      - 53 49 04  2 40 01 80 43 78 53 25 21 63 03 46 12 48 14 24 35 52 85 43 9  21 77 69 17  5 81
 Offset Number 215   - 57 83 13 54 18 28 51 43 24 58 81 07 58 64 82 51 06 65 85 00 91 68 91 92 20 34 99 38 32 99
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
 Ciphertext Output   - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB


                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

11    1   1 21  2  2 22 1   1 2  11 11 21 2  1  2     1  21  1 11 1  11  1 21  2 12  1    11    11 21 1  12 11
48 63 43 91 44 20 33 14 35 41 42 92 60 42 53 23 55 17 61 25 62 39 74 00 56 02 71 63 74 1  90 64 09 32 24 90 17
43 91 79 67 23 80 59 16 29 49 36 51 65 59 26 22 66 49 82 37 55 62 96 70 53 73 52 23 36 71 09 49 67 67 40 84 58
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

 2 21  1 11 1   1 11    2  1  11 12     1  1  2 21 1   1    1  1  11    1   1 11 21 11 11  1 11 21     2 11
91 08 85 48 49 23 98 25 08 87 17 71  7 90 53 42 51 18 23 13 57 62 98 96 68 71 81 04 26 35 10 95 37  4 92 71 56
03 54 84 20 94 05 62 21 33 77 42 74 80 88 30 76 32 60 74 93 61 05 15 53 19 61 18 91 04 70 33 48 04 78 17 60 28
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

1   1 2     22  2 1   2    11  1 1   1  2  2 12        2  1 12 22  2 1
59 52 54 8  54 61 45 61 68 38 97 78 10 22 23 03 53 6  92  7 32 43 31 61
37 06 12 76 41 05 07 68 11 98 25 12 12 34 95 42 47 74 39 90 85 71 40 76
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

#1 (randomly created incorrect Transposition key that does work)

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

   Transposition     - 1  12     1  1  2 1  21 12    11 2  12 21    11  1  2 1  11 11 1  21 2  21 1  11 1  21 1
    Key Number ???   - 18 64 22 33 40 94 49 13 84  9 15 27 03 28 89 73 49 13 39 22 67 92 42 53 22 29 18 67 25 44
  Offset Number ???  - 15 51 91 80 11 46 88 33 82 99 81 08 08 63 91 27 39 94 53 78 29 27 35 16 31 35 36 93 72 29
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
  Ciphertext Output  - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB
```

```
                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

21 11 11  1 21  2 2     1   2  1 11 11 11  2 12  2 11    21 1   1    2  12  2 2  11 21  2 21  1  1 21    1  22
35 47 49 71 36 64 51 52 05 21 79 71 90 92 54 01 81 37 31 45 68  8  7 27 60 53 25 21 34 64 12 61 17 30 82 98 21
59 05 73 40 38 75 03 38 69 09 01 04 85 09 70 94 12 87 92 95 62 62 25 91 02 27 16 42 95 58 02 49 18 67 35 44 58
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40
```

```
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

11 12  1    21     1     1 1     2   2  1  2  2 21 1  11  2              1  2  1     1  1 11       21 1   1 1
73 73 52 33 53 98  8 52 34 13 18 58  1 65 60  0 45 06 45 70 7  51 25 62 39 75 79 64 16 46 60 11 14 58 44  1 03
41 32 80 21 39 00 19 12 06 67 08 66 85 08 29 46 77 48 16 23 97 56 64 13 36 62 77 90 84 61 73 75 44 54 48 55 85
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58
```

```
22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

1  11 2  21 11  2  2  2 21 1  11 12 11 22 2  21 11     1 2   1 21  2 22
16 58 03 17 04 92 40 62 03 69 78 34 38 12 08 59 58 52 45 0  92 34 43 10
73 45 04 11 23 64 75 68 14 37 61 64 27 62 87 45 30 04 20 43 26 19 50 77
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

#2 (randomly created incorrect Transposition key that does work)

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

   Transposition     -  1 2  11        1    2   1 22 1  21 12 2     11  1  2 1   1 1  21  2 1  11 1  11 2   2 1
    Key Number ???   -  6 07 97 13 42 95 76 47 75 50 48 26 04 01 86 33 98 13 22 42 74 32 21 7  26 19 68 47 92 05
  Offset Number ???  - 75 57 29 40 19 44 10 49 57 30 58 17 01 13 92 61 36 94 80 77 88 69 66 62 28 15 00 63 04 19
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
  Ciphertext Output  - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB


                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

 1  1 12 22 21 12 11 2  21 2  2  21  2 2     12 12 1  11 12        2  1  1 11  2 22 1  12 12 2  11 1  11 1  12
89 39 73 02 02 21 01 15 18 28 46 38 85 31 83 01 31 65 13 94 67 54 21 62 14 70 45 34 66 54 91 21 88 16 44 3  51
77 39 38 65 41 00 78 10 42 76 94 21 12 06 56 99 32 46 42 12 52 89 13 83 84 25 66 97 26 65 65 21 93 99 61 49 37
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

   2   1    11 1   1  1 21    1   1  2 12       21  2 11 21 1   1  1 11 11 12 1  1      1 1  1     1  1   1 12
42 54 46 11 16 72 30 22 05 73 95 14 64 00 98  9 35 90 06 26 95 25 78 08 39 43 78 51  9 94 25 9   3 4  34  1 52
43 52 09 20 23 04 86 26 99 41 32 72 73 47 10 46 15 83 86 01 47 88 05 28 00 97 53 05 82 70 41 86 54 73 93 17 19
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

11 11    2     12 21 22 21 2  21 11  1  1 2  21 1     11 21 2     21
12 10 94 32 63 12 54 12 57 06 44 73 38 85 05 27 13 53 83 39 55 87 43 72
05 33 95 57 19 63 08 88 47 23 83 17 27 22 83 64 57 55 45 35 14 46 08 85
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

#3 (randomly created incorrect Transposition key that does work)

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

   Transposition     - 1  11 2   1 2     2   1 22 21     1 21 21     2 21 21 1     1   1  1 11  1  1 1  1   1 21
    Key Number ???   - 7  07 3  85 35 8  06 26 15 39 72 82 58 04 39 14 48 19 24 43 64 24 63 39 74 80 63 26 32 19
  Offset Number ???  - 37 55 62 90 29 78 50 36 33 16 35 24 26 15 81 54 01 67 78 73 71 91 47 85 03 52 09 06 62 42
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
  Ciphertext Output  - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB


                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

2   2  1 21 1   2  1 11 11  2 22 11  2    2  1   1 22 11 12 1   1 1  11 1  21 11 21 12 12  1 2  2  1  2  1  21
18 83 51 20 53 14  1 65 01 91 42 72 73 12 54 93 57 55 98 03 54  4 76 31 65 04 89 31 51 33 95 28 01 83 11 3  27
13 08 74 63 57 85 48 34 19 09 92 41 97 47 54 12 67 16 99 93 66 62 93 51 93 77 88 42 97 30 92 46 83 20 86 49 58
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

1  1     12 22  1    11  2    11     2  1 2  1   1 1  12 21  1    21 11 2     11  1 12 12  1 11 12 11 1  1  11
87 62 56 34 43 92  1 81 23 91 82 64 52 95 03 67 84 16 31 23 82 25 15 83 5  77 94 67 11 20 71 95 72 47 44 95 74
45 50 87 21 75 38 11 55 49 50 09 59 03 81 25 14 49 69 62 10 13 82 53 71 05 61 36 82 30 53 87 48 67 80 45 01 10
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

2  2  2     2     21 11 1  1  1  1   2 22 1     11  1  2 2  21 21    2
29 45 04 98 07 92 20 68 01 31 63 07 15 24 06 64 26 50 20 4  41 05 39 24
27 64 03 48 67 66 90 23 47 99 41 82 24 03 62 12 68 57 19 23 80 47 42 80
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

#4 (randomly created incorrect Transposition key that does work)

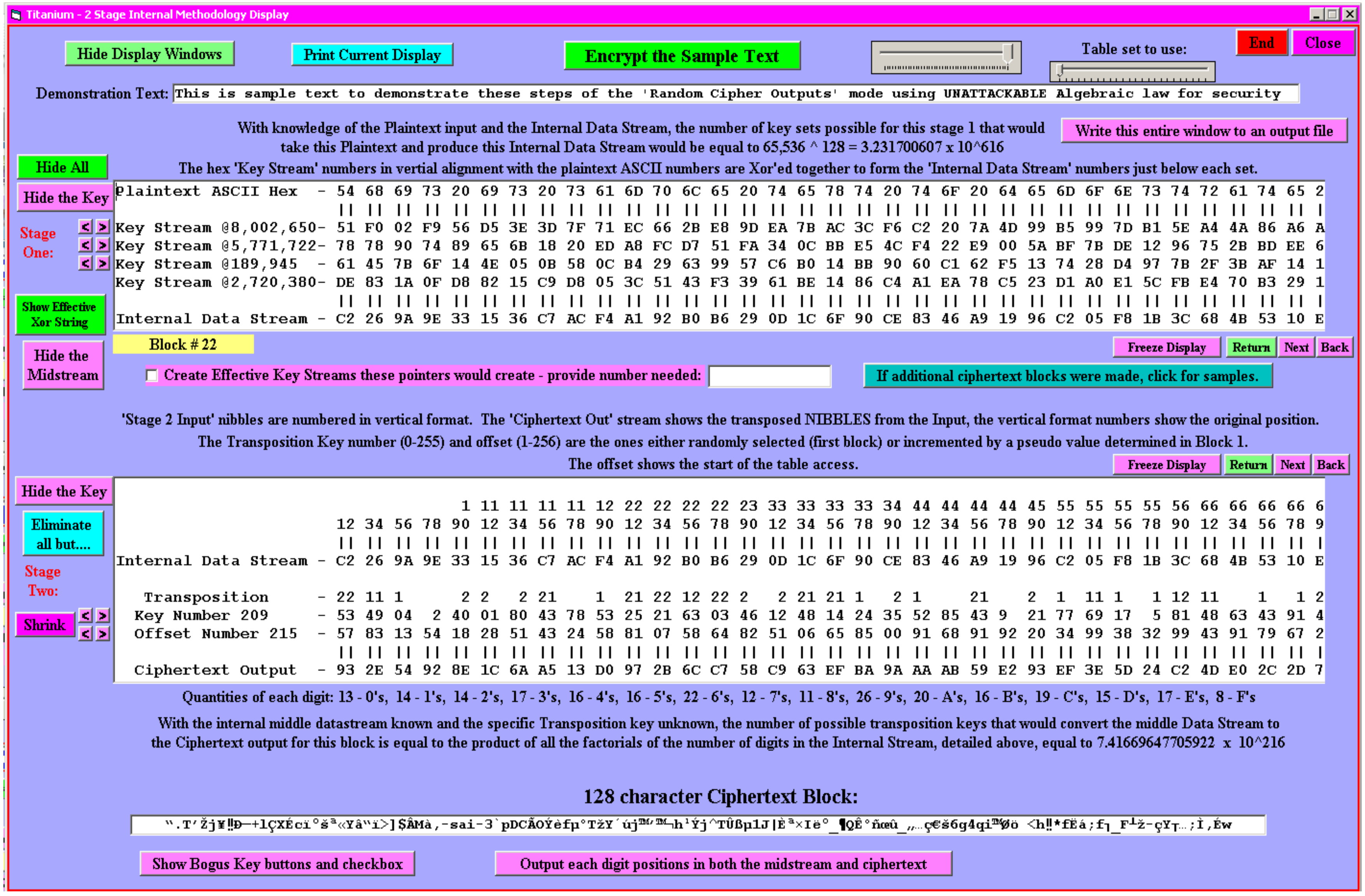
```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

   Transposition     - 21 11 1   1 21  1 1  11 22  1    21    11    1      2 22  1 12 11 2  11    12  1 1   2
    Key Number ???   - 50 11 14  4 30 95 69 33 24 77 15 27  3 11 88 27 31 93 22 36 64 47 18 98 23 72 59 66 22 41
  Offset Number ???  - 20 14 94 10 21 44 55 73 17 66 38 08 79 68 97 92 60 79 97 51 20 22 52 64 61 36 09 06 03 92
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
  Ciphertext Output  - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB


                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

2  21 11 11 11       21  1    2     11 1  1  21  2 22 12 2      2  2 2   1 12 2  1  11 1   1  2  2 2  11 12  2
07 07 06 02 54  1 37 11 88 91 47  7 92 62 44 23 55 51 83 28 61 51 41 07 56 55 11 2  89 9  95 61 51 06 63 70 53
23 35 58 26 73 84 38 13 62 01 90 27 43 47 81 22 66 12 06 44 59 29 86 91 53 15 37 49 37 85 92 44 98 89 76 74 44
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

12     1  1 21  1 21 11  2  2 1   1  2  2  1 11 11    11     2    11 2   2 12 21  1 1  11 1  1  2  11     1 11
94 59 22 28 34 30 19 81 61 74 84 70 24 60 52 20 54 94 03 22 80 6  21 56 84 13 34 35 69 23 55 58 3  47 64 30 73
54 73 50 96 14 47 02 50 87 41 10 96 83 07 38 74 59 87 84 34 15 76 25 43 05 27 86 08 92 50 91 68 03 70 22 83 15
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

   21 21 21 21  1 1  12 11 1  2   1 1  11    21 21     2 2  11 22  1
27 41 07 34 09 69 58 42 86 34 43 13 89 80 39 07 22 1  25 44 35 53 49 81
25 67 04 51 63 11 05 58 76 96 87 51 96 89 21 19 51 84 10 25 83 33 30 36
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

#5 (randomly created incorrect Transposition key that does work)

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - A1 18 A5 63 79 BB 0A 55 D3 E1 14 E8 52 53 B8 9B 81 81 9A 95 11 11 1E 81 B3 45 C4 3D 65 BF

   Transposition     - 21 11 1     2  11        2 12 1  21 11 11 1   2     1     1 21 11 12 1  21    11  1  1 22
    Key Number ???   - 54 09 1   3 41 36 76 17 74 54 72 21 16 16 78 11 33 74 63 32 19 97 02 34 21 59 58 16 25 30
  Offset Number ???  - 58 03 92 15 02 99 10 99 52 78 85 18 18 62 67 66 67 26 94 07 75 21 90 81 34 78 41 87 12 68
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
  Ciphertext Output  - 66 69 B1 A8 1B C3 6F ED E7 7A 05 E0 69 F0 AC 5E 19 E0 81 88 01 B5 2E A1 29 6F 39 30 12 BB


                                                          1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FC E2 D0 38 82 6E 0F E7 90 D8 DE 07 6B C5 A1 CA 1C FF EF A6 BF 4F 6B 85 24 67 39 4F 40 B3 92 60 E4 88 58 EB DD

    2  1  1 1   1 12    12 2   2  1 12    11    11 2   1 22 2   1  1 2   2  1 2  11 21 12  2 1  21  1       1
58 43  9 91 47 29 63 9  01 38 22 98 93 98 07 74 35 06 30 43 58 58 48 08 50 27 18 85 25 64 82 95 19 13 6  58 81
33 78 79 55 97 80 19 18 69 26 62 00 47 98 50 09 46 05 91 31 62 82 88 95 52 73 31 35 89 65 94 19 47 15 66 64 53
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
C0 89 69 F4 79 3A 80 C3 BB 1B 22 19 D6 A5 61 2B DD AD 9B 27 DE 5B 1E 86 3C 56 ED 07 00 DA A2 2B B0 B8 05 D7 40


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
85 EA C8 5B 9C 00 66 78 52 43 7D 75 03 80 CA 6D 09 31 55 61 9A D0 19 9B 01 49 FE BA 2B 9D 1A 0D 9A FC 87 6F 5B

1  21 2  11 2  1  11  2 11 11 11 1  11  1     1  1 22 11      1 11  1 2  12 22    2   2 2   1 2     12 1   1 11
93 07 05 37 39 83 98 22 22 82 47 92 62 90 66 80  1 20 57 92 12 12 61 52 03 55 37 21 61 4  71 15 94 42 64 35 34
37 54 70 25 34 43 69 25 84 71 32 89 02 68 28 02 92 63 17 74 73 39 47 43 70 32 13 95 70 44 80 22 22 77 35 83 60
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
1E 61 53 B9 7C 18 AB 44 80 F9 95 DB 32 F5 C8 8F 77 F8 5D E8 D6 35 24 FE 8A 76 90 85 3A E8 04 D5 A1 61 C1 A4 58


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
8A CD EB 2E 00 BE E2 22 4F 10 8A 71 70 CB 69 01 97 2E A3 7A 28 A6 7F 6D

11 21    21 1  22 11 21  1 2  11 11     2 11 21  2 11  2       22  2 2
03 42 12 14 01 01 36 14 63 34 68 52 37 64 44 07 54 25 45 74  5 04  5 34
31 66 40 11 40 65 05 85 13 56 46 85 24 39 24 19 41 00 30 64 51 47 31 40
|| || || || || || || || || || || || || || || || || || || || || || || ||
4E 34 A1 C5 F9 F2 86 00 FD CE A9 5E BF E2 BC F1 49 38 18 71 A4 77 1A 05
```

This design seems to be the first one to allow more solutions that do decrypt the text than there are keys for the AES, with no mathematical methodology available to help isolate the correct set of keys. With the capability of expanding a relatively small key (2,048 bytes) into an 8 Mbyte Vernam and over 8,000 Transposition keys, this cipher design ought to be seriously considered by the security community.

Appendix B


Titanium - 2 Stage Internal Methodology Display
Hide Display Windows
Print Current Display
Encrypt the Sample Text
Table set to use:
End
Close
Write this entire window to an output file
Hide All
Hide the Key
Stage One:
Show Effective Xor String
Hide the Midstream
Freeze Display
Return
Next
Back
If additional ciphertext blocks were made, click for samples.
Hide the Key
Eliminate all but....
Stage Two:
Shrink
Show Bogus Key buttons and checkbox
Output each digit positions in both the midstream and ciphertext

Demonstration Text: `This is sample text to demonstrate these steps of the 'Random Cipher Outputs' mode using UNATTACKABLE Algebraic law for security`

With knowledge of the Plaintext input and the Internal Data Stream, the number of key sets possible for this stage 1 that would take this Plaintext and produce this Internal Data Stream would be equal to 65,536 ^ 128 = 3.231700607 x 10^616

The hex 'Key Stream' numbers in vertial alignment with the plaintext ASCII numbers are Xor'ed together to form the 'Internal Data Stream' numbers just below each set.

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74 72 61 74 65 2
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Key Stream @8,002,650- 51 F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6 A
Key Stream @5,771,722- 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12 96 75 2B BD EE 6
Key Stream @189,945  - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97 7B 2F 3B AF 14 1
Key Stream @2,720,380- DE 83 1A 0F D8 82 15 C9 D8 05 3C 51 43 F3 39 61 BE 14 86 C4 A1 EA 78 C5 23 D1 A0 E1 5C FB E4 70 B3 29 1
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C 68 4B 53 10 E
```

Block # 22

☐ Create Effective Key Streams these pointers would create - provide number needed:

'Stage 2 Input' nibbles are numbered in vertical format. The 'Ciphertext Out' stream shows the transposed NIBBLES from the Input, the vertical format numbers show the original position.
The Transposition Key number (0-255) and offset (1-256) are the ones either randomly selected (first block) or incremented by a pseudo value determined in Block 1.
The offset shows the start of the table access.

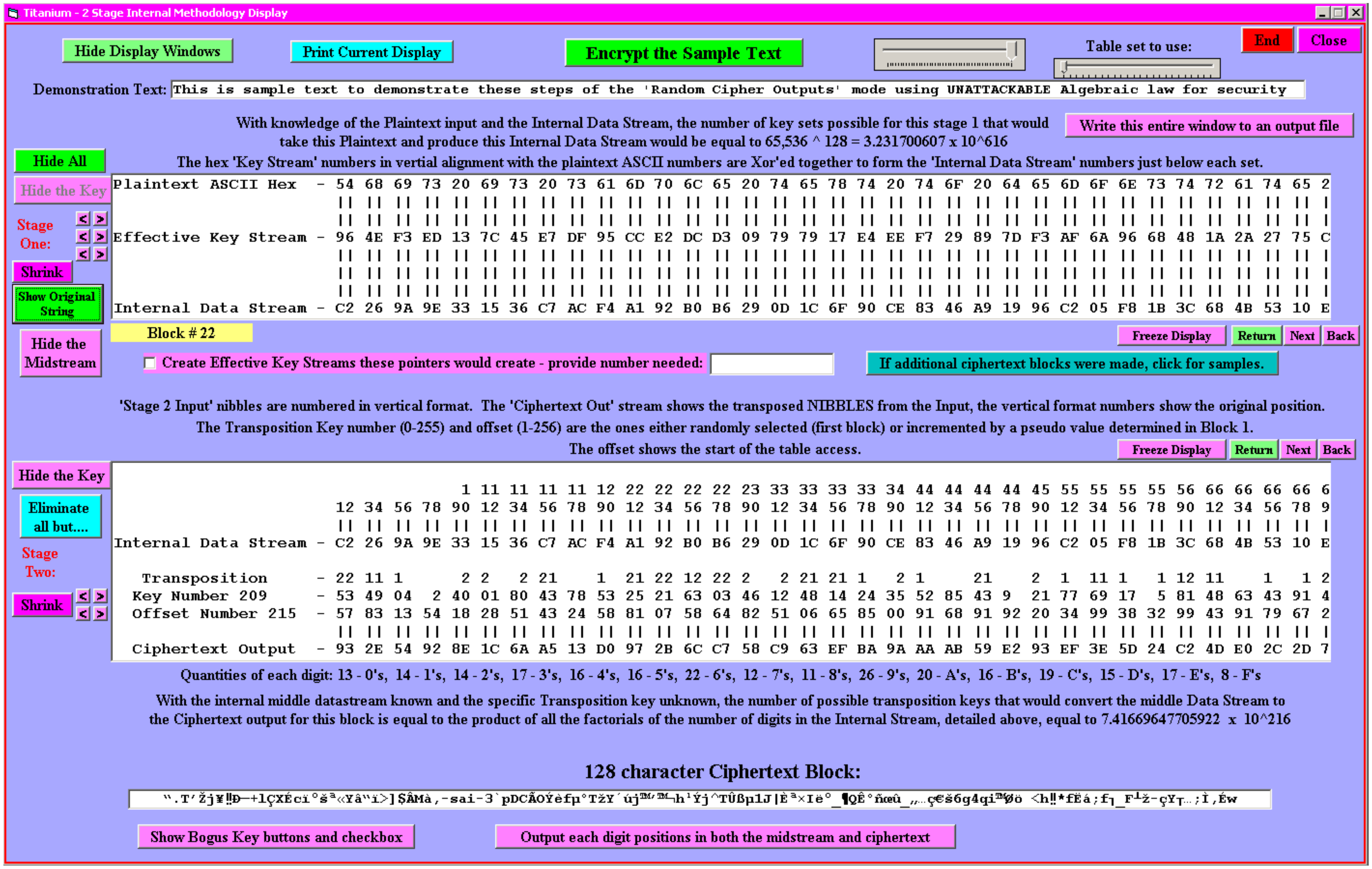
```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 6
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 9
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C 68 4B 53 10 E

  Transposition      - 22 11 1      2 2   2 21    1  21 22 12 22 2   2 21 21 1   2 1     21    2  1  11 1   1 12 11    1   1 2
 Key Number 209      - 53 49 04  2 40 01 80 43 78 53 25 21 63 03 46 12 48 14 24 35 52 85 43 9  21 77 69 17  5 81 48 63 43 91 4
 Offset Number 215   - 57 83 13 54 18 28 51 43 24 58 81 07 58 64 82 51 06 65 85 00 91 68 91 92 20 34 99 38 32 99 43 91 79 67 2
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
 Ciphertext Output   - 93 2E 54 92 8E 1C 6A A5 13 D0 97 2B 6C C7 58 C9 63 EF BA 9A AA AB 59 E2 93 EF 3E 5D 24 C2 4D E0 2C 2D 7
```

Quantities of each digit: 13 - 0's, 14 - 1's, 14 - 2's, 17 - 3's, 16 - 4's, 16 - 5's, 22 - 6's, 12 - 7's, 11 - 8's, 26 - 9's, 20 - A's, 16 - B's, 19 - C's, 15 - D's, 17 - E's, 8 - F's

With the internal middle datastream known and the specific Transposition key unknown, the number of possible transposition keys that would convert the middle Data Stream to the Ciphertext output for this block is equal to the product of all the factorials of the number of digits in the Internal Stream, detailed above, equal to 7.41669647705922 x 10^216

128 character Ciphertext Block:

```
".T'žj¥‼Ð—+lÇXÉci°š³«Yâ"i>]$ÂMà,-sai-3`pDCÃOÝèfµ°TžY´új™⁄™ℎh¹Ýj^TÛßµ1J|È³×Ié°_¶QÊ°ñœû_„…ç€š6g4qi™Øö <h‼*fËá;f┐_F┴ž-çY┬…;Ì,Èw
```



Demonstration Text: This is sample text to demonstrate these steps of the 'Random Cipher Outputs' mode using UNATTACKABLE Algebraic law for security

With knowledge of the Plaintext input and the Internal Data Stream, the number of key sets possible for this stage 1 that would take this Plaintext and produce this Internal Data Stream would be equal to 65,536 ^ 128 = 3.231700607 x 10^616

The hex 'Key Stream' numbers in vertial alignment with the plaintext ASCII numbers are Xor'ed together to form the 'Internal Data Stream' numbers just below each set.

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74 72 61 74 65 2
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Effective Key Stream - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68 48 1A 2A 27 75 C
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C 68 4B 53 10 E
```

Block # 22

☐ Create Effective Key Streams these pointers would create - provide number needed:

'Stage 2 Input' nibbles are numbered in vertical format. The 'Ciphertext Out' stream shows the transposed NIBBLES from the Input, the vertical format numbers show the original position.
The Transposition Key number (0-255) and offset (1-256) are the ones either randomly selected (first block) or incremented by a pseudo value determined in Block 1.
The offset shows the start of the table access.

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 6
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 9
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
Internal Data Stream - C2 26 9A 9E 33 15 36 C7 AC F4 A1 92 B0 B6 29 0D 1C 6F 90 CE 83 46 A9 19 96 C2 05 F8 1B 3C 68 4B 53 10 E

  Transposition      - 22 11 1      2 2   2 21     1 21 22 12 22 2    2 21 21 1    2 1     21    2  1  11 1    1 12 11    1    1 2
 Key Number 209      - 53 49 04  2 40 01 80 43 78 53 25 21 63 03 46 12 48 14 24 35 52 85 43 9  21 77 69 17  5 81 48 63 43 91 4
 Offset Number 215   - 57 83 13 54 18 28 51 43 24 58 81 07 58 64 82 51 06 65 85 00 91 68 91 92 20 34 99 38 32 99 43 91 79 67 2
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || |
 Ciphertext Output   - 93 2E 54 92 8E 1C 6A A5 13 D0 97 2B 6C C7 58 C9 63 EF BA 9A AA AB 59 E2 93 EF 3E 5D 24 C2 4D E0 2C 2D 7
```

Quantities of each digit: 13 - 0's, 14 - 1's, 14 - 2's, 17 - 3's, 16 - 4's, 16 - 5's, 22 - 6's, 12 - 7's, 11 - 8's, 26 - 9's, 20 - A's, 16 - B's, 19 - C's, 15 - D's, 17 - E's, 8 - F's

With the internal middle datastream known and the specific Transposition key unknown, the number of possible transposition keys that would convert the middle Data Stream to the Ciphertext output for this block is equal to the product of all the factorials of the number of digits in the Internal Stream, detailed above, equal to 7.41669647705922 x 10^216

128 character Ciphertext Block:

```
".T'Žj¥‼Ð—+1ÇXÉci°š ª«Yâ"i>]$ÂMà,-sai-3`pDCÃOÝèfµ°TžY´új™⁄™ℎh¹Ýj^TÛßµ1J|È ª×Ië°_¶QÊ°ñœû_„…ç€š6g4qi™Øö <h‼*fËá;f┐_F┴ž-çY┬…;Ì,Éw
```

Appendix C

All the Effective Key Streams are copied to the bottom of this page for your comparison to see the changes resulting from very minor changes to just one of the first 3 pointers. The 4th pointer is calculated from the first three pointers, so since the first pointer changed, pointer 4 will change significantly.

```
Key Stream @8,002,650  - 51 F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @2,720,380  - DE 83 1A 0F D8 82 15 C9 D8 05 3C 51 43 F3 39 61 BE 14 86 C4 A1 EA 78 C5 23 D1 A0 E1 5C
Effective Key Stream 1 - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68

Key Stream @8,002,651  - F0 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,405,013  - 72 63 46 38 F7 BE 65 F7 F0 E1 7A 93 D8 01 5F 62 F6 3F AA 44 78 77 7B 37 E1 B1 41 4A 74
Effective Key Stream 2 - 9B 5C 54 75 BF AB 36 9B F9 EC 00 6D 84 54 18 EB E6 AC 02 5A CC EE BD 5B 1D E3 6F F1 AF

Key Stream @8,002,652  - 02 F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @2,165,114  - E9 69 BE BE BD 99 09 E7 F3 EC 17 CF 08 8C 78 46 E4 2C 4C 78 74 74 26 43 81 79 5E 84 2C
Effective Key Stream 3 - F2 AD 03 70 1E 8F 18 85 67 6B 20 F2 21 AE AE 18 64 75 D0 84 9A DA 34 03 51 CF BC D0 0D

Key Stream @8,002,653  - F9 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,844,345  - C0 E3 51 4D 75 01 B0 9E 15 DB 13 28 20 41 86 5B 09 C0 1F 10 8C E5 31 F5 D3 61 54 0C 82
Effective Key Stream 4 - 20 88 6F 68 D5 55 AF 61 0B 11 E7 60 7E F2 87 95 43 AD 61 B6 55 9F 0F 99 E7 1B 59 A2 4D

Key Stream @8,002,654  - 56 D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,482,898  - 9C BD A2 05 A0 20 65 97 5E C6 B3 DE C4 62 F4 CB D4 A8 66 F2 29 06 45 FA 54 77 BE 7B 80
Effective Key Stream 5 - D3 55 77 23 42 7A E7 E2 0D CF 32 E1 0B 06 65 CF AA 27 42 63 24 50 57 72 AC E2 49 3B 83

Key Stream @8,002,655  - D5 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,592,209  - AE 49 55 82 6D A2 05 26 F7 30 13 36 2E 86 90 BD ED 55 F5 FD 0C 5D C6 39 03 E5 06 8E 52
Effective Key Stream 6 - 62 4A 83 E6 81 65 0D 1E 67 4C E5 98 36 72 CB 8D 71 80 E6 B8 2D 27 30 7D 14 8A 1F 02 71

Key Stream @8,002,656  - 3E 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6 AA
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @4,050,259  - A7 43 4B B3 DF 9A 57 A5 84 E6 86 08 4C E1 A0 B2 3C F9 6D 84 E5 6A 29 5A AC 4E 01 73 2A
Effective Key Stream 7 - 80 43 DF D9 AE D7 12 5E 61 ED E1 71 C4 DF CF 60 FA 1B AA ED E8 F4 13 F1 41 CF D4 DF 05

Key Stream @8,002,657  - 3D 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6 AA 8B
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @4,099,933  - 51 4F EB B6 EE EA CD BC 12 65 9A D1 6D C9 C8 C5 AD 7A 45 1B 12 4E 0D 76 1A 74 C8 86 64
Effective Key Stream 8 - 75 0D 71 41 15 EA 4B 32 80 FF 2A 38 2F C3 45 4D 5C 4C AE 5E FB 1C D8 27 19 39 3D 26 6A

Key Stream @8,002,658  - 7F 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6 AA 8B 52
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,965,376  - 8C E4 47 E3 3D A2 F1 E3 A5 CC E6 E7 90 03 5B BB 7A 46 85 9B FA 95 A7 53 E2 55 7A B5 E4
Effective Key Stream 9 - EA A8 40 9E 8B 61 02 1A A6 81 C6 C4 E6 EB 8C 04 5F 5C 42 3A DF 28 88 EC 2D 38 83 34 33

Key Stream @8,002,659  - 71 EC 66 2B E8 9D EA 7B AC 3C F6 C2 20 7A 4D 99 B5 99 7D B1 5E A4 4A 86 A6 AA 8B 52 DE
Key Stream @5,771,722  - 78 78 90 74 89 65 6B 18 20 ED A8 FC D7 51 FA 34 0C BB E5 4C F4 22 E9 00 5A BF 7B DE 12
Key Stream @189,945    - 61 45 7B 6F 14 4E 05 0B 58 0C B4 29 63 99 57 C6 B0 14 BB 90 60 C1 62 F5 13 74 28 D4 97
Key Stream @3,284,554  - 95 F7 68 AC 2E F0 7C B2 F5 48 1E 87 2C C5 54 14 9F 0A 5A 43 84 8E 73 39 3D 9D BC A0 E3
Effective Key Stream 10- FD 26 E5 9C 5B 46 F8 DA 21 95 F4 90 B8 77 B4 7F 96 3C 79 2E 4E C9 B2 4A D2 FC 64 F8 B8


Effective Key Stream 1 - 96 4E F3 ED 13 7C 45 E7 DF 95 CC E2 DC D3 09 79 79 17 E4 EE F7 29 89 7D F3 AF 6A 96 68
Effective Key Stream 2 - 9B 5C 54 75 BF AB 36 9B F9 EC 00 6D 84 54 18 EB E6 AC 02 5A CC EE BD 5B 1D E3 6F F1 AF
Effective Key Stream 3 - F2 AD 03 70 1E 8F 18 85 67 6B 20 F2 21 AE AE 18 64 75 D0 84 9A DA 34 03 51 CF BC D0 0D
Effective Key Stream 4 - 20 88 6F 68 D5 55 AF 61 0B 11 E7 60 7E F2 87 95 43 AD 61 B6 55 9F 0F 99 E7 1B 59 A2 4D
Effective Key Stream 5 - D3 55 77 23 42 7A E7 E2 0D CF 32 E1 0B 06 65 CF AA 27 42 63 24 50 57 72 AC E2 49 3B 83
Effective Key Stream 6 - 80 43 DF D9 AE D7 12 5E 61 ED E1 71 C4 DF CF 60 FA 1B AA ED E8 F4 13 F1 41 CF D4 DF 05
Effective Key Stream 7 - 80 43 DF D9 AE D7 12 5E 61 ED E1 71 C4 DF CF 60 FA 1B AA ED E8 F4 13 F1 41 CF D4 DF 05
Effective Key Stream 8 - 75 0D 71 41 15 EA 4B 32 80 FF 2A 38 2F C3 45 4D 5C 4C AE 5E FB 1C D8 27 19 39 3D 26 6A
Effective Key Stream 9 - EA A8 40 9E 8B 61 02 1A A6 81 C6 C4 E6 EB 8C 04 5F 5C 42 3A DF 28 88 EC 2D 38 83 34 33
Effective Key Stream 10- FD 26 E5 9C 5B 46 F8 DA 21 95 F4 90 B8 77 B4 7F 96 3C 79 2E 4E C9 B2 4A D2 FC 64 F8 B8
```

Appendix D

This pseudo-random key table-based function is called to advance all 6 Vernam Two pointers:

```
Sub advancePointers()
    Dim byte0P1 As Long, byte1P1 As Long, byte2P1 As Long, byte0P2 As Long, byte1P2 As Long, byte2P2 As Long, byte0P3 As Long, byte1P3 As Long, _
        byte2P3 As Long, n As Byte, p1 As Long, p2 As Long, p3 As Long

    ' Get the pointer to the base value
    n = ((ctxBlockNum - 1) Mod 24) + 1: p1 = ptrBase(1, n) + e1p1: p2 = ptrBase(2, n) + e1p2: p3 = ptrBase(3, n) + e1p3
    ' Select how the new pointers will be formed
    Select Case (ctxBlockNum Mod 6)
        Case 0: byte0P1 = e1Key(p1): byte1P1 = e1Key(p2): byte2P1 = e1Key(p3)
                byte0P2 = e1Key(p2): byte1P2 = e1Key(p3): byte2P2 = e1Key(p1)
                byte0P3 = e1Key(p3): byte1P3 = e1Key(p1): byte2P3 = e1Key(p2)
        Case 1: byte0P1 = e1Key(p1): byte1P1 = e1Key(p3): byte2P1 = e1Key(p2)
                byte0P2 = e1Key(p3): byte1P2 = e1Key(p2): byte2P2 = e1Key(p1)
                byte0P3 = e1Key(p2): byte1P3 = e1Key(p1): byte2P3 = e1Key(p3)
        Case 2: byte0P1 = e1Key(p2): byte1P1 = e1Key(p1): byte2P1 = e1Key(p3)
                byte0P2 = e1Key(p1): byte1P2 = e1Key(p3): byte2P2 = e1Key(p2)
                byte0P3 = e1Key(p3): byte1P3 = e1Key(p2): byte2P3 = e1Key(p1)
        Case 3: byte0P1 = e1Key(p2): byte1P1 = e1Key(p3): byte2P1 = e1Key(p1)
                byte0P2 = e1Key(p3): byte1P2 = e1Key(p1): byte2P2 = e1Key(p2)
                byte0P3 = e1Key(p1): byte1P3 = e1Key(p2): byte2P3 = e1Key(p3)
        Case 4: byte0P1 = e1Key(p3): byte1P1 = e1Key(p1): byte2P1 = e1Key(p2)
                byte0P2 = e1Key(p1): byte1P2 = e1Key(p2): byte2P2 = e1Key(p3)
                byte0P3 = e1Key(p2): byte1P3 = e1Key(p3): byte2P3 = e1Key(p1)
        Case 5: byte0P1 = e1Key(p3): byte1P1 = e1Key(p2): byte2P1 = e1Key(p1)
                byte0P2 = e1Key(p2): byte1P2 = e1Key(p1): byte2P2 = e1Key(p3)
                byte0P3 = e1Key(p1): byte1P3 = e1Key(p3): byte2P3 = e1Key(p2)
    End Select
    ' Form the pointers
    e1p1 = ((byte2P1 * &H10000) + (byte1P1 * &H100&) + byte0P1) And &H1FFFFF
    e1p2 = ((byte2P2 * &H10000) + (byte1P2 * &H100&) + byte0P2) And &H1FFFFF
    e1p3 = ((byte2P3 * &H10000) + (byte1P3 * &H100&) + byte0P3) And &H1FFFFF
    ' Advance the pointers if it has already been used within 1,000,000 blocks
    If lastPtrUsage(1, e1p1) > 0 Then Do While (ctxBlockNum - lastPtrUsage(1, e1p1)) < 1000000: e1p1 = (e1p1 + 1) And &H1FFFFF: Loop
    If lastPtrUsage(2, e1p2) > 0 Then Do While (ctxBlockNum - lastPtrUsage(2, e1p2)) < 1000000: e1p2 = (e1p2 + 1) And &H1FFFFF: Loop
    If lastPtrUsage(3, e1p3) > 0 Then Do While (ctxBlockNum - lastPtrUsage(3, e1p3)) < 1000000: e1p3 = (e1p3 + 1) And &H1FFFFF: Loop
    lastPtrUsage(1, e1p1) = ctxBlockNum: lastPtrUsage(2, e1p2) = ctxBlockNum: lastPtrUsage(3, e1p3) = ctxBlockNum
    ' Now modify the Engine 2 pointers
    Select Case ((e1p1 Xor e1p2) And &H1F&)
        Case 0:  e2Ptr = e1p1 And &H1FFF&
        Case 1:  e2Ptr = Int(e1p1 / 2) And &H1FFF&
        Case 2:  e2Ptr = Int(e1p1 / 4) And &H1FFF&
        Case 3:  e2Ptr = Int(e1p1 / 8) And &H1FFF&
        Case 4:  e2Ptr = Int(e1p1 / &H10&) And &H1FFF&
        Case 5:  e2Ptr = Int(e1p1 / &H20&) And &H1FFF&
        Case 6:  e2Ptr = Int(e1p1 / &H40&) And &H1FFF&
        Case 7:  e2Ptr = Int(e1p1 / &H80&) And &H1FFF&
        Case 8:  e2Ptr = e1p2 And &H1FFF&
        Case 9:  e2Ptr = Int(e1p2 / 2) And &H1FFF&
        Case 10:  e2Ptr = Int(e1p2 / 4) And &H1FFF&
        Case 11:  e2Ptr = Int(e1p2 / 8) And &H1FFF&
        Case 12:  e2Ptr = Int(e1p2 / &H10&) And &H1FFF&
        Case 13:  e2Ptr = Int(e1p2 / &H20&) And &H1FFF&
        Case 14:  e2Ptr = Int(e1p2 / &H40&) And &H1FFF&
        Case 15:  e2Ptr = Int(e1p2 / &H80&) And &H1FFF&
        Case 16:  e2Ptr = e1p3 And &H1FFF&
        Case 17:  e2Ptr = Int(e1p3 / 2) And &H1FFF&
        Case 18:  e2Ptr = Int(e1p3 / 4) And &H1FFF&
        Case 19:  e2Ptr = Int(e1p3 / 8) And &H1FFF&
        Case 20:  e2Ptr = Int(e1p3 / &H10&) And &H1FFF&
        Case 21:  e2Ptr = Int(e1p3 / &H20&) And &H1FFF&
        Case 22:  e2Ptr = Int(e1p3 / &H40&) And &H1FFF&
        Case 23:  e2Ptr = Int(e1p3 / &H80&) And &H1FFF&
        Case 24:  e2Ptr = e1p4 And &H1FFF&
        Case 25:  e2Ptr = Int(e1p4 / 2) And &H1FFF&
        Case 26:  e2Ptr = Int(e1p4 / 4) And &H1FFF&
        Case 27:  e2Ptr = Int(e1p4 / 8) And &H1FFF&
        Case 28:  e2Ptr = Int(e1p4 / &H10&) And &H1FFF&
        Case 29:  e2Ptr = Int(e1p4 / &H20&) And &H1FFF&
        Case 30:  e2Ptr = Int(e1p4 / &H40&) And &H1FFF&
        Case 31:  e2Ptr = Int(e1p4 / &H80&) And &H1FFF&
    End Select
```

```
Select Case ((e1p2 Xor e1p3) And &H1F&)
        Case 0:  e2Off = e1p1 And &HFF&
        Case 1:  e2Off = Int(e1p1 / 2) And &HFF&
        Case 2:  e2Off = Int(e1p1 / 4) And &HFF&
        Case 3:  e2Off = Int(e1p1 / 8) And &HFF&
        Case 4:  e2Off = Int(e1p1 / &H10&) And &HFF&
        Case 5:  e2Off = Int(e1p1 / &H20&) And &HFF&
        Case 6:  e2Off = Int(e1p1 / &H40&) And &HFF&
        Case 7:  e2Off = Int(e1p1 / &H80&) And &HFF&
        Case 8:  e2Off = e1p2 And &HFF&
        Case 9:  e2Off = Int(e1p2 / 2) And &HFF&
        Case 10:  e2Off = Int(e1p2 / 4) And &HFF&
        Case 11:  e2Off = Int(e1p2 / 8) And &HFF&
        Case 12:  e2Off = Int(e1p2 / &H10&) And &HFF&
        Case 13:  e2Off = Int(e1p2 / &H20&) And &HFF&
        Case 14:  e2Off = Int(e1p2 / &H40&) And &HFF&
        Case 15:  e2Off = Int(e1p2 / &H80&) And &HFF&
        Case 16:  e2Off = e1p3 And &HFF&
        Case 17:  e2Off = Int(e1p3 / 2) And &HFF&
        Case 18:  e2Off = Int(e1p3 / 4) And &HFF&
        Case 19:  e2Off = Int(e1p3 / 8) And &HFF&
        Case 20:  e2Off = Int(e1p3 / &H10&) And &HFF&
        Case 21:  e2Off = Int(e1p3 / &H20&) And &HFF&
        Case 22:  e2Off = Int(e1p3 / &H40&) And &HFF&
        Case 23:  e2Off = Int(e1p3 / &H80&) And &HFF&
        Case 24:  e2Off = e1p4 And &HFF&
        Case 25:  e2Off = Int(e1p4 / 2) And &HFF&
        Case 26:  e2Off = Int(e1p4 / 4) And &HFF&
        Case 27:  e2Off = Int(e1p4 / 8) And &HFF&
        Case 28:  e2Off = Int(e1p4 / &H10&) And &HFF&
        Case 29:  e2Off = Int(e1p4 / &H20&) And &HFF&
        Case 30:  e2Off = Int(e1p4 / &H40&) And &HFF&
        Case 31:  e2Off = Int(e1p4 / &H80&) And &HFF&
    End Select
    ' Set the pointer to its correct 1-based number
    e2Off = e2Off + 1
    ' Form the 4th pointer - the VERY LAST setup to perform
    ' Get the particular base number to use from the ciphertext block number, then form the base into the respective blocks
    n = ((ctxBlockNum - 1) Mod 24) + 1: p1 = ptrBase(1, n) + e1p1: p2 = ptrBase(2, n) + e1p2: p3 = ptrBase(3, n) + e1p3
    ' Form the byte pointers
    Select Case (ctxBlockNum Mod 6)
        Case 0: byte0 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte1 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte2 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
        Case 1: byte0 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte2 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte1 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
        Case 2: byte1 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte0 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte2 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
        Case 3: byte1 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte2 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte0 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
        Case 4: byte2 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte0 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte1 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
        Case 5: byte2 = e1Key(p1) Xor e1Key(p2 + 1) Xor e1Key(p3 + 2)
                byte1 = e1Key(p1 + 1) Xor e1Key(p2 + 2) Xor e1Key(p3)
                byte0 = e1Key(p1 + 2) Xor e1Key(p2) Xor e1Key(p3 + 1)
    End Select
    ' Form the pointer
    e1p4 = (((byte2 * &H10000) + (byte1 * &H100&) + byte0) Xor ctxBlockNum Xor e2Ptr Xor e2Off) And &H1FFFFF
End Sub
```

The effectiveness of the ‘AdvancePointers’ function:

Not only are the pointers changed in a key table-based pseudo-random fashion, changing their position instead of their values will result in different sequences. Notice in these examples that 5 values are used, set to different pointers for the 6 possible arrangements. The Engine 2 pointers are held constant in all 6 scenarios. Pointer 4 and the two Engine 2 pointers rely on the order of the first 3 pointers for their incremental values, as shown here. Notice that block 1 illustrates the same starting values as the first scenario but in a different order. For example, notice that scenarios 1 and 2 hold the same starting values for pointer 1. But the next block’s pointer 1 contains 1,976,443 and 1,124,470 respectively. The only items that changed were the order of pointers 2 and 3, not their value:

```
Initial pointers Block #1: P1 = 1,619,019, P2 = 663,180, P3 = 961,408, P4 = 1,835,494, E2P = 253, E2O = 92

Scenario #1: Block #1: P1 = 1,619,019, P2 = 663,180, P3 = 961,408, P4 = 1,835,494, E2P = 253, E2O = 92
             Block #2: P1 = 1,976,443, P2 = 1,793,576, P3 = 555,870, P4 = 1,390,795, E2P = 3,947, E2O = 294
             Block #3: P1 = 1,196,355, P2 = 225,857, P3 = 82,802, P4 = 1,154,972, E2P = 4,176, E2O = 165
             Block #4: P1 = 1,745,703, P2 = 506,531, P3 = 206,778, P4 = 492,789, E2P = 2,610, E2O = 204
             Block #5: P1 = 1,262,074, P2 = 1,717,057, P3 = 129,587, P4 = 646,007, E2P = 4,254, E2O = 71
             Block #6: P1 = 507,547, P2 = 1,771,454, P3 = 2,005,767, P4 = 1,991,660, E2P = 7,668, E2O = 71
             Block #7: P1 = 835,719, P2 = 486,592, P3 = 34,668, P4 = 517,206, E2P = 6,529, E2O = 199
             Block #8: P1 = 723,795, P2 = 1,248,011, P3 = 742,155, P4 = 1,873,764, E2P = 1,110, E2O = 98

Scenario #2: Block #1: P1 = 1,619,019, P2 = 961,408, P3 = 663,180, P4 = 1,873,764, E2P = 253, E2O = 92
             Block #2: P1 = 1,124,470, P2 = 1,495,336, P3 = 554,705, P4 = 1,010,917, E2P = 4,701, E2O = 246
             Block #3: P1 = 1,215,085, P2 = 856,714, P3 = 683,282, P4 = 1,405,354, E2P = 1,300, E2O = 282
             Block #4: P1 = 393,364, P2 = 1,361,408, P3 = 38,086, P4 = 1,903,192, E2P = 2,380, E2O = 160
             Block #5: P1 = 353,126, P2 = 435,555, P3 = 222,885, P4 = 297,704, E2P = 2,843, E2O = 146
             Block #6: P1 = 411,767, P2 = 1,549,896, P3 = 554,918, P4 = 1,874,194, E2P = 2,325, E2O = 217
             Block #7: P1 = 818,786, P2 = 183,422, P3 = 1,991,372, P4 = 1,552,946, E2P = 2,449, E2O = 164
             Block #8: P1 = 999,268, P2 = 266,047, P3 = 2,057,231, P4 = 457,019, E2P = 5,702, E2O = 209

Scenario #3: Block #1: P1 = 663,180, P2 = 1,619,019, P3 = 961,408, P4 = 457,019, E2P = 253, E2O = 92
             Block #2: P1 = 2,011,798, P2 = 1,466,034, P3 = 1,218,142, P4 = 1,579,229, E2P = 2,857, E2O = 212
             Block #3: P1 = 843,489, P2 = 126,174, P3 = 2,023,916, P4 = 975,197, E2P = 4,145, E2O = 310
             Block #4: P1 = 1,656,580, P2 = 276,807, P3 = 459,833, P4 = 1,382,808, E2P = 2,272, E2O = 176
             Block #5: P1 = 1,921,847, P2 = 1,523,027, P3 = 1,259,325, P4 = 995,023, E2P = 5,427, E2O = 217
             Block #6: P1 = 731,774, P2 = 2,026,282, P3 = 687,851, P4 = 2,015,732, E2P = 2,030, E2O = 231
             Block #7: P1 = 915,264, P2 = 3,575, P3 = 1,523,725, P4 = 926,012, E2P = 3,712, E2O = 314
             Block #8: P1 = 839,472, P2 = 1,101,007, P3 = 995,532, P4 = 26,339, E2P = 7,234, E2O = 268

Scenario #4: Block #1: P1 = 663,180, P2 = 961,408, P3 = 1,619,019, P4 = 26,339, E2P = 253, E2O = 92
             Block #2: P1 = 1,094,262, P2 = 1,495,218, P3 = 1,210,064, P4 = 1,493,389, E2P = 2,855, E2O = 282
             Block #3: P1 = 1,632,359, P2 = 522,472, P3 = 550,904, P4 = 810,636, E2P = 4,081, E2O = 127
             Block #4: P1 = 1,443,201, P2 = 87,557, P3 = 360,790, P4 = 1,233,220, E2P = 88, E2O = 227
             Block #5: P1 = 1,643,598, P2 = 956,692, P3 = 1,330,841, P4 = 1,765,597, E2P = 5,201, E2O = 202
             Block #6: P1 = 1,457,950, P2 = 1,996,351, P3 = 2,039,414, P4 = 745,847, E2P = 8,079, E2O = 263
             Block #7: P1 = 267,969, P2 = 115,734, P3 = 1,491,396, P4 = 1,353,211, E2P = 3,459, E2O = 120
             Block #8: P1 = 710,505, P2 = 600,791, P3 = 1,534,250, P4 = 435,246, E2P = 4,759, E2O = 213

Scenario #5: Block #1: P1 = 961,408, P2 = 1,619,019, P3 = 663,180, P4 = 435,246, E2P = 253, E2O = 92
             Block #2: P1 = 1,116,310, P2 = 1,495,304, P3 = 562,897, P4 = 1,216,357, E2P = 6,800, E2O = 136
             Block #3: P1 = 508,973, P2 = 853,956, P3 = 273,671, P4 = 1,165,000, E2P = 994, E2O = 75
             Block #4: P1 = 1,084,863, P2 = 2,044,045, P3 = 900,912, P4 = 1,661,515, E2P = 4,044, E2O = 264
             Block #5: P1 = 648,807, P2 = 502,246, P3 = 419,753, P4 = 1,430,904, E2P = 4,915, E2O = 217
             Block #6: P1 = 1,504,466, P2 = 1,234,676, P3 = 1,364,694, P4 = 1,756,084, E2P = 7,123, E2O = 155
             Block #7: P1 = 503,505, P2 = 1,140,654, P3 = 971,111, P4 = 1,835,448, E2P = 5,527, E2O = 108
             Block #8: P1 = 1,903,437, P2 = 892,171, P3 = 740,765, P4 = 1,279,295, E2P = 5,165, E2O = 154

Scenario #6: Block #1: P1 = 961,408, P2 = 663,180, P3 = 1,619,019, P4 = 1,279,295, E2P = 253, E2O = 92
             Block #2: P1 = 1,050,747, P2 = 1,822,728, P3 = 555,984, P4 = 1,039,779, E2P = 3,962, E2O = 224
             Block #3: P1 = 1,974,635, P2 = 728,609, P3 = 92,958, P4 = 1,507,839, E2P = 1,928, E2O = 301
             Block #4: P1 = 880,775, P2 = 494,960, P3 = 1,083,277, P4 = 436,727, E2P = 271, E2O = 291
             Block #5: P1 = 81,913, P2 = 1,646,911, P3 = 2,095,393, P4 = 21,057, E2P = 1,279, E2O = 292
             Block #6: P1 = 1,991,920, P2 = 1,064,548, P3 = 323,646, P4 = 1,579,756, E2P = 3,843, E2O = 228
             Block #7: P1 = 344,765, P2 = 1,926,466, P3 = 179,557, P4 = 1,393,223, E2P = 4,149, E2O = 225
             Block #8: P1 = 1,350,545, P2 = 1,168,539, P3 = 1,806,804, P4 = 1,347,639, E2P = 5,414, E2O = 122
```

The effectiveness of the ‘AdvancePointers’ function:

In several tests (153, 1 hr each) simulating the encryption of 2 billion blocks of plaintext (256 Gigabytes) in each test, it was learned that all values between 0 and 2,097,151 inclusive were selected for each of the first 3 pointers in usually less than 20 million blocks. The fourth pointer, due to its different initialization, takes longer to use all of its values:

```
E1P1 = 1,002,614, E1P2 = 1,521,525, E1P3 = 863,223, E2P = 193, E2O = 117:
    Block count where all addresses used for pointer #1 - 15,183,700
    Block count where all addresses used for pointer #2 - 15,406,500
    Block count where all addresses used for pointer #3 - 16,555,000
    Block count where all addresses used for pointer #4 - 35,122,700

E1P1 = 1,004,373, E1P2 = 786,647, E1P3 = 538,067, E2P = 150, E2O = 153:
    Block count where all addresses used for pointer #1 - 17,282,500
    Block count where all addresses used for pointer #2 - 16,396,400
    Block count where all addresses used for pointer #3 - 15,649,600
    Block count where all addresses used for pointer #4 - 32,882,200

E1P1 = 1,014,903, E1P2 = 1,350,259, E1P3 = 469,502, E2P = 125, E2O = 26:
    Block count where all addresses used for pointer #1 - 16,255,800
    Block count where all addresses used for pointer #2 - 17,077,400
    Block count where all addresses used for pointer #3 - 16,002,100
    Block count where all addresses used for pointer #4 - 31,966,500

E1P1 = 1,052,942, E1P2 = 234,539, E1P3 = 398,933, E2P = 171, E2O = 199:
    Block count where all addresses used for pointer #1 - 16,012,200
    Block count where all addresses used for pointer #2 - 16,548,500
    Block count where all addresses used for pointer #3 - 18,478,200
    Block count where all addresses used for pointer #4 - 29,425,100

E1P1 = 1,068,442, E1P2 = 1,448,842, E1P3 = 575,802, E2P = 19, E2O = 50:
    Block count where all addresses used for pointer #1 - 17,401,100
    Block count where all addresses used for pointer #2 - 16,229,800
    Block count where all addresses used for pointer #3 - 17,989,200
    Block count where all addresses used for pointer #4 - 31,781,600

E1P1 = 1,069,639, E1P2 = 874,755, E1P3 = 732,750, E2P = 199, E2O = 63:
    Block count where all addresses used for pointer #1 - 17,187,100
    Block count where all addresses used for pointer #2 - 18,344,900
    Block count where all addresses used for pointer #3 - 16,260,500
    Block count where all addresses used for pointer #4 - 33,311,100

E1P1 = 1,069,869, E1P2 = 1,466,703, E1P3 = 62,571, E2P = 8, E2O = 253:
    Block count where all addresses used for pointer #1 - 16,377,600
    Block count where all addresses used for pointer #2 - 16,461,100
    Block count where all addresses used for pointer #3 - 15,850,200
    Block count where all addresses used for pointer #4 - 35,395,500

E1P1 = 1,092,940, E1P2 = 628,083, E1P3 = 1,347,710, E2P = 4, E2O = 96:
    Block count where all addresses used for pointer #1 - 15,000,100
    Block count where all addresses used for pointer #2 - 15,567,400
    Block count where all addresses used for pointer #3 - 15,562,800
    Block count where all addresses used for pointer #4 - 35,864,000

E1P1 = 1,098,469, E1P2 = 1,745,191, E1P3 = 2,066,400, E2P = 122, E2O = 149:
    Block count where all addresses used for pointer #1 - 16,256,200
    Block count where all addresses used for pointer #2 - 15,000,100
    Block count where all addresses used for pointer #3 - 18,781,300
    Block count where all addresses used for pointer #4 - 28,592,900

E1P1 = 1,105,622, E1P2 = 1,243,604, E1P3 = 1,996,890, E2P = 36, E2O = 245:
    Block count where all addresses used for pointer #1 - 17,551,100
    Block count where all addresses used for pointer #2 - 17,494,300
    Block count where all addresses used for pointer #3 - 15,809,900
    Block count where all addresses used for pointer #4 - 31,304,200

E1P1 = 1,120,928, E1P2 = 1,414,902, E1P3 = 962,422, E2P = 45, E2O = 189:
    Block count where all addresses used for pointer #1 - 18,335,800
    Block count where all addresses used for pointer #2 - 17,641,300
    Block count where all addresses used for pointer #3 - 16,052,400
    Block count where all addresses used for pointer #4 - 30,695,400

E1P1 = 1,144,045, E1P2 = 631,950, E1P3 = 340,268, E2P = 161, E2O = 193:
    Block count where all addresses used for pointer #1 - 16,548,300
    Block count where all addresses used for pointer #2 - 18,098,300
    Block count where all addresses used for pointer #3 - 17,333,100
    Block count where all addresses used for pointer #4 - 31,200,300

E1P1 = 1,151,087, E1P2 = 1,326,731, E1P3 = 1,482,166, E2P = 178, E2O = 137:
    Block count where all addresses used for pointer #1 - 18,091,600
    Block count where all addresses used for pointer #2 - 16,918,500
    Block count where all addresses used for pointer #3 - 18,009,700
    Block count where all addresses used for pointer #4 - 30,588,700

E1P1 = 1,153,597, E1P2 = 160,926, E1P3 = 983,292, E2P = 208, E2O = 232:
    Block count where all addresses used for pointer #1 - 18,094,400
    Block count where all addresses used for pointer #2 - 15,591,200
    Block count where all addresses used for pointer #3 - 18,916,000
    Block count where all addresses used for pointer #4 - 39,986,400

E1P1 = 1,153,970, E1P2 = 561,474, E1P3 = 935,954, E2P = 218, E2O = 252:
    Block count where all addresses used for pointer #1 - 17,645,700
    Block count where all addresses used for pointer #2 - 15,000,100
    Block count where all addresses used for pointer #3 - 15,725,900
    Block count where all addresses used for pointer #4 - 31,077,400

E1P1 = 1,156,289, E1P2 = 203,412, E1P3 = 48,794, E2P = 62, E2O = 136:
    Block count where all addresses used for pointer #1 - 19,871,400
    Block count where all addresses used for pointer #2 - 16,281,000
    Block count where all addresses used for pointer #3 - 17,821,200
    Block count where all addresses used for pointer #4 - 33,951,000

E1P1 = 1,165,673, E1P2 = 523,165, E1P3 = 1,977,984, E2P = 185, E2O = 58:
    Block count where all addresses used for pointer #1 - 15,395,300
    Block count where all addresses used for pointer #2 - 15,000,100
    Block count where all addresses used for pointer #3 - 17,367,500
    Block count where all addresses used for pointer #4 - 35,185,000

E1P1 = 1,178,089, E1P2 = 527,387, E1P3 = 268,036, E2P = 217, E2O = 18:
    Block count where all addresses used for pointer #1 - 15,161,000
    Block count where all addresses used for pointer #2 - 17,985,300
    Block count where all addresses used for pointer #3 - 18,553,300
    Block count where all addresses used for pointer #4 - 32,805,700

E1P1 = 1,183,525, E1P2 = 1,645,161, E1P3 = 1,376,796, E2P = 161, E2O = 96:
    Block count where all addresses used for pointer #1 - 15,515,100
    Block count where all addresses used for pointer #2 - 18,017,900
    Block count where all addresses used for pointer #3 - 18,367,200
    Block count where all addresses used for pointer #4 - 30,396,800

E1P1 = 1,184,888, E1P2 = 593,903, E1P3 = 725,258, E2P = 241, E2O = 123:
    Block count where all addresses used for pointer #1 - 22,501,000
    Block count where all addresses used for pointer #2 - 17,178,700
    Block count where all addresses used for pointer #3 - 16,918,300
    Block count where all addresses used for pointer #4 - 29,613,600

E1P1 = 1,185,530, E1P2 = 888,937, E1P3 = 1,046,428, E2P = 194, E2O = 55:
    Block count where all addresses used for pointer #1 - 16,518,700
    Block count where all addresses used for pointer #2 - 16,377,600
    Block count where all addresses used for pointer #3 - 17,711,200
    Block count where all addresses used for pointer #4 - 30,585,000

E1P1 = 1,186,075, E1P2 = 1,744,645, E1P3 = 1,412,295, E2P = 42, E2O = 213:
    Block count where all addresses used for pointer #1 - 16,475,300
    Block count where all addresses used for pointer #2 - 15,000,100
    Block count where all addresses used for pointer #3 - 16,010,600
    Block count where all addresses used for pointer #4 - 34,434,400

E1P1 = 1,194,496, E1P2 = 550,870, E1P3 = 1,028,051, E2P = 19, E2O = 246:
    Block count where all addresses used for pointer #1 - 15,250,100
    Block count where all addresses used for pointer #2 - 17,344,200
    Block count where all addresses used for pointer #3 - 16,365,800
    Block count where all addresses used for pointer #4 - 30,295,100

E1P1 = 1,220,857, E1P2 = 1,441,643, E1P3 = 820,117, E2P = 40, E2O = 104:
    Block count where all addresses used for pointer #1 - 16,280,700
    Block count where all addresses used for pointer #2 - 19,139,500
    Block count where all addresses used for pointer #3 - 16,215,700
    Block count where all addresses used for pointer #4 - 34,426,100

E1P1 = 1,268,072, E1P2 = 372,768, E1P3 = 1,041,272, E2P = 93, E2O = 123:
    Block count where all addresses used for pointer #1 - 19,116,200
    Block count where all addresses used for pointer #2 - 17,554,100
    Block count where all addresses used for pointer #3 - 17,075,000
    Block count where all addresses used for pointer #4 - 36,694,100

E1P1 = 1,275,554, E1P2 = 1,753,202, E1P3 = 1,079,938, E2P = 84, E2O = 134:
    Block count where all addresses used for pointer #1 - 16,698,400
    Block count where all addresses used for pointer #2 - 17,213,900
    Block count where all addresses used for pointer #3 - 16,623,900
    Block count where all addresses used for pointer #4 - 30,195,800

E1P1 = 1,276,759, E1P2 = 1,797,842, E1P3 = 1,980,896, E2P = 223, E2O = 118:
    Block count where all addresses used for pointer #1 - 15,656,000
    Block count where all addresses used for pointer #2 - 17,860,300
    Block count where all addresses used for pointer #3 - 16,275,700
    Block count where all addresses used for pointer #4 - 31,660,000

E1P1 = 1,290,346, E1P2 = 1,715,352, E1P3 = 1,567,117, E2P = 136, E2O = 226:
    Block count where all addresses used for pointer #1 - 17,619,400
    Block count where all addresses used for pointer #2 - 18,156,400
    Block count where all addresses used for pointer #3 - 17,799,200
    Block count where all addresses used for pointer #4 - 33,109,500

E1P1 = 1,290,574, E1P2 = 680,556, E1P3 = 1,613,458, E2P = 167, E2O = 134:
    Block count where all addresses used for pointer #1 - 18,401,200
    Block count where all addresses used for pointer #2 - 17,252,900
    Block count where all addresses used for pointer #3 - 16,447,700
    Block count where all addresses used for pointer #4 - 29,178,900

E1P1 = 1,297,988, E1P2 = 1,663,882, E1P3 = 1,938,232, E2P = 129, E2O = 6:
    Block count where all addresses used for pointer #1 - 16,269,300
    Block count where all addresses used for pointer #2 - 18,204,500
    Block count where all addresses used for pointer #3 - 16,839,500
    Block count where all addresses used for pointer #4 - 32,801,200

E1P1 = 1,299,900, E1P2 = 930,723, E1P3 = 1,732,461, E2P = 205, E2O = 111:
    Block count where all addresses used for pointer #1 - 15,164,000
    Block count where all addresses used for pointer #2 - 16,647,800
    Block count where all addresses used for pointer #3 - 18,027,900
    Block count where all addresses used for pointer #4 - 38,955,800

E1P1 = 1,310,684, E1P2 = 1,890,114, E1P3 = 597,650, E2P = 47, E2O = 71:
    Block count where all addresses used for pointer #1 - 17,849,600
    Block count where all addresses used for pointer #2 - 16,631,600
    Block count where all addresses used for pointer #3 - 15,145,100
    Block count where all addresses used for pointer #4 - 31,426,500

E1P1 = 1,321,525, E1P2 = 1,113,143, E1P3 = 2,016,179, E2P = 245, E2O = 9:
    Block count where all addresses used for pointer #1 - 15,299,000
    Block count where all addresses used for pointer #2 - 15,000,100
    Block count where all addresses used for pointer #3 - 18,101,100
    Block count where all addresses used for pointer #4 - 34,507,800

E1P1 = 1,343,768, E1P2 = 1,853,967, E1P3 = 123,562, E2P = 65, E2O = 46:
    Block count where all addresses used for pointer #1 - 16,901,100
    Block count where all addresses used for pointer #2 - 17,238,900
    Block count where all addresses used for pointer #3 - 18,358,000
    Block count where all addresses used for pointer #4 - 29,110,200

E1P1 = 1,345,465, E1P2 = 1,878,317, E1P3 = 748,625, E2P = 61, E2O = 251:
    Block count where all addresses used for pointer #1 - 16,306,000
    Block count where all addresses used for pointer #2 - 15,742,500
    Block count where all addresses used for pointer #3 - 15,679,600
    Block count where all addresses used for pointer #4 - 33,455,900

E1P1 = 1,375,031, E1P2 = 923,570, E1P3 = 181,697, E2P = 149, E2O = 121:
    Block count where all addresses used for pointer #1 - 15,834,200
    Block count where all addresses used for pointer #2 - 15,993,800
    Block count where all addresses used for pointer #3 - 17,114,800
    Block count where all addresses used for pointer #4 - 31,348,400
```

```
E1P1 = 1,427,447, E1P2 = 918,129, E1P3 = 4,739, E2P = 189, E2O = 110:
     Block count where all addresses used for pointer #1 - 15,248,600
     Block count where all addresses used for pointer #2 - 16,310,400
     Block count where all addresses used for pointer #3 - 16,537,900
     Block count where all addresses used for pointer #4 - 31,624,700

E1P1 = 1,427,450, E1P2 = 335,719, E1P3 = 615,585, E2P = 157, E2O = 44:
     Block count where all addresses used for pointer #1 - 16,755,600
     Block count where all addresses used for pointer #2 - 16,893,800
     Block count where all addresses used for pointer #3 - 15,026,700
     Block count where all addresses used for pointer #4 - 29,835,300

E1P1 = 1,453,761, E1P2 = 1,887,635, E1P3 = 111,260, E2P = 117, E2O = 20:
     Block count where all addresses used for pointer #1 - 18,531,000
     Block count where all addresses used for pointer #2 - 15,730,600
     Block count where all addresses used for pointer #3 - 15,959,000
     Block count where all addresses used for pointer #4 - 35,746,600

E1P1 = 1,466,692, E1P2 = 2,011,403, E1P3 = 38,709, E2P = 3, E2O = 116:
     Block count where all addresses used for pointer #1 - 15,228,300
     Block count where all addresses used for pointer #2 - 17,562,500
     Block count where all addresses used for pointer #3 - 16,308,000
     Block count where all addresses used for pointer #4 - 30,973,700

E1P1 = 1,473,218, E1P2 = 601,488, E1P3 = 312,997, E2P = 231, E2O = 15:
     Block count where all addresses used for pointer #1 - 16,644,800
     Block count where all addresses used for pointer #2 - 15,397,700
     Block count where all addresses used for pointer #3 - 16,589,600
     Block count where all addresses used for pointer #4 - 38,510,600

E1P1 = 1,497,974, E1P2 = 1,780,852, E1P3 = 1,664,764, E2P = 153, E2O = 52:
     Block count where all addresses used for pointer #1 - 15,308,700
     Block count where all addresses used for pointer #2 - 15,109,800
     Block count where all addresses used for pointer #3 - 16,873,800
     Block count where all addresses used for pointer #4 - 31,063,300

E1P1 = 1,529,197, E1P2 = 1,839,121, E1P3 = 1,427,365, E2P = 119, E2O = 182:
     Block count where all addresses used for pointer #1 - 15,886,200
     Block count where all addresses used for pointer #2 - 16,607,100
     Block count where all addresses used for pointer #3 - 15,523,000
     Block count where all addresses used for pointer #4 - 30,026,900

E1P1 = 1,544,515, E1P2 = 1,957,134, E1P3 = 712,492, E2P = 209, E2O = 59:
     Block count where all addresses used for pointer #1 - 16,226,600
     Block count where all addresses used for pointer #2 - 16,130,100
     Block count where all addresses used for pointer #3 - 17,977,800
     Block count where all addresses used for pointer #4 - 30,052,600

E1P1 = 1,554,631, E1P2 = 1,078,914, E1P3 = 1,696,464, E2P = 197, E2O = 73:
     Block count where all addresses used for pointer #1 - 18,435,600
     Block count where all addresses used for pointer #2 - 16,274,300
     Block count where all addresses used for pointer #3 - 16,484,300
     Block count where all addresses used for pointer #4 - 33,120,000

E1P1 = 1,572,296, E1P2 = 2,067,838, E1P3 = 1,371,102, E2P = 53, E2O = 208:
     Block count where all addresses used for pointer #1 - 15,801,300
     Block count where all addresses used for pointer #2 - 15,640,200
     Block count where all addresses used for pointer #3 - 16,179,400
     Block count where all addresses used for pointer #4 - 28,792,600

E1P1 = 1,576,485, E1P2 = 114,535, E1P3 = 1,256,736, E2P = 202, E2O = 43:
     Block count where all addresses used for pointer #1 - 15,223,600
     Block count where all addresses used for pointer #2 - 18,192,800
     Block count where all addresses used for pointer #3 - 15,699,900
     Block count where all addresses used for pointer #4 - 28,247,800

E1P1 = 1,579,643, E1P2 = 489,703, E1P3 = 681,251, E2P = 100, E2O = 91:
     Block count where all addresses used for pointer #1 - 16,446,300
     Block count where all addresses used for pointer #2 - 21,735,300
     Block count where all addresses used for pointer #3 - 16,033,500
     Block count where all addresses used for pointer #4 - 30,016,100

E1P1 = 1,585,609, E1P2 = 912,125, E1P3 = 190,177, E2P = 251, E2O = 233:
     Block count where all addresses used for pointer #1 - 17,165,600
     Block count where all addresses used for pointer #2 - 15,418,800
     Block count where all addresses used for pointer #3 - 15,739,800
     Block count where all addresses used for pointer #4 - 32,321,300

E1P1 = 1,602,686, E1P2 = 295,260, E1P3 = 1,247,300, E2P = 43, E2O = 147:
     Block count where all addresses used for pointer #1 - 16,241,000
     Block count where all addresses used for pointer #2 - 17,124,100
     Block count where all addresses used for pointer #3 - 19,452,200
     Block count where all addresses used for pointer #4 - 28,292,400

E1P1 = 1,631,933, E1P2 = 39,969, E1P3 = 811,893, E2P = 242, E2O = 117:
     Block count where all addresses used for pointer #1 - 15,000,100
     Block count where all addresses used for pointer #2 - 17,151,300
     Block count where all addresses used for pointer #3 - 17,972,700
     Block count where all addresses used for pointer #4 - 32,104,700

E1P1 = 1,648,236, E1P2 = 1,945,748, E1P3 = 633,755, E2P = 247, E2O = 13:
     Block count where all addresses used for pointer #1 - 16,207,700
     Block count where all addresses used for pointer #2 - 16,945,300
     Block count where all addresses used for pointer #3 - 16,444,600
     Block count where all addresses used for pointer #4 - 33,847,300

E1P1 = 1,652,229, E1P2 = 1,472,200, E1P3 = 555,518, E2P = 88, E2O = 185:
     Block count where all addresses used for pointer #1 - 18,660,500
     Block count where all addresses used for pointer #2 - 15,868,600
     Block count where all addresses used for pointer #3 - 16,345,700
     Block count where all addresses used for pointer #4 - 32,329,400

E1P1 = 1,659,332, E1P2 = 1,474,954, E1P3 = 2,047,930, E2P = 208, E2O = 66:
     Block count where all addresses used for pointer #1 - 15,883,400
     Block count where all addresses used for pointer #2 - 18,482,100
     Block count where all addresses used for pointer #3 - 17,076,200
     Block count where all addresses used for pointer #4 - 30,674,400

E1P1 = 1,661,577, E1P2 = 1,862,203, E1P3 = 490,660, E2P = 124, E2O = 79:
     Block count where all addresses used for pointer #1 - 18,825,700
     Block count where all addresses used for pointer #2 - 17,370,500
     Block count where all addresses used for pointer #3 - 17,062,600
     Block count where all addresses used for pointer #4 - 31,053,300

E1P1 = 1,674,121, E1P2 = 2,064,315, E1P3 = 1,393,317, E2P = 216, E2O = 210:
     Block count where all addresses used for pointer #1 - 15,825,300
     Block count where all addresses used for pointer #2 - 15,784,800
     Block count where all addresses used for pointer #3 - 15,143,500
     Block count where all addresses used for pointer #4 - 31,617,500

E1P1 = 1,677,397, E1P2 = 212,697, E1P3 = 1,332,428, E2P = 231, E2O = 236:
     Block count where all addresses used for pointer #1 - 17,290,400
     Block count where all addresses used for pointer #2 - 18,861,400
     Block count where all addresses used for pointer #3 - 16,650,200
     Block count where all addresses used for pointer #4 - 31,551,200

E1P1 = 1,683,855, E1P2 = 413,225, E1P3 = 857,565, E2P = 71, E2O = 24:
     Block count where all addresses used for pointer #1 - 16,630,700
     Block count where all addresses used for pointer #2 - 16,486,700
     Block count where all addresses used for pointer #3 - 18,763,300
     Block count where all addresses used for pointer #4 - 33,386,800

E1P1 = 1,691,789, E1P2 = 597,681, E1P3 = 1,121,221, E2P = 171, E2O = 205:
     Block count where all addresses used for pointer #1 - 16,828,700
     Block count where all addresses used for pointer #2 - 17,161,400
     Block count where all addresses used for pointer #3 - 16,019,200
     Block count where all addresses used for pointer #4 - 31,829,000

E1P1 = 1,707,063, E1P2 = 189,618, E1P3 = 1,597,121, E2P = 175, E2O = 204:
     Block count where all addresses used for pointer #1 - 18,970,000
     Block count where all addresses used for pointer #2 - 16,409,300
     Block count where all addresses used for pointer #3 - 16,023,100
     Block count where all addresses used for pointer #4 - 30,807,200

E1P1 = 1,709,834, E1P2 = 643,129, E1P3 = 1,996,844, E2P = 24, E2O = 31:
     Block count where all addresses used for pointer #1 - 18,053,100
     Block count where all addresses used for pointer #2 - 19,073,300
     Block count where all addresses used for pointer #3 - 15,209,600
     Block count where all addresses used for pointer #4 - 41,661,200

E1P1 = 1,716,939, E1P2 = 309,877, E1P3 = 1,179,897, E2P = 204, E2O = 150:
     Block count where all addresses used for pointer #1 - 16,400,300
     Block count where all addresses used for pointer #2 - 20,065,900
     Block count where all addresses used for pointer #3 - 15,262,200
     Block count where all addresses used for pointer #4 - 34,829,700

E1P1 = 1,722,884, E1P2 = 323,658, E1P3 = 1,961,208, E2P = 49, E2O = 208:
     Block count where all addresses used for pointer #1 - 18,428,800
     Block count where all addresses used for pointer #2 - 19,750,500
     Block count where all addresses used for pointer #3 - 17,918,100
     Block count where all addresses used for pointer #4 - 32,444,500

E1P1 = 1,780,950, E1P2 = 902,486, E1P3 = 1,504,597, E2P = 243, E2O = 102:
     Block count where all addresses used for pointer #1 - 16,245,800
     Block count where all addresses used for pointer #2 - 16,385,000
     Block count where all addresses used for pointer #3 - 16,698,200
     Block count where all addresses used for pointer #4 - 35,196,700

E1P1 = 1,785,540, E1P2 = 377,099, E1P3 = 814,518, E2P = 158, E2O = 12:
     Block count where all addresses used for pointer #1 - 17,993,900
     Block count where all addresses used for pointer #2 - 16,528,700
     Block count where all addresses used for pointer #3 - 15,622,200
     Block count where all addresses used for pointer #4 - 30,453,700

E1P1 = 1,802,944, E1P2 = 2,023,062, E1P3 = 1,661,078, E2P = 146, E2O = 100:
     Block count where all addresses used for pointer #1 - 15,000,100
     Block count where all addresses used for pointer #2 - 17,142,900
     Block count where all addresses used for pointer #3 - 15,886,800
     Block count where all addresses used for pointer #4 - 27,379,000

E1P1 = 1,821,174, E1P2 = 485,237, E1P3 = 1,942,136, E2P = 237, E2O = 90:
     Block count where all addresses used for pointer #1 - 18,682,700
     Block count where all addresses used for pointer #2 - 17,685,100
     Block count where all addresses used for pointer #3 - 16,046,400
     Block count where all addresses used for pointer #4 - 35,500,400

E1P1 = 1,838,059, E1P2 = 788,117, E1P3 = 1,283,607, E2P = 163, E2O = 140:
     Block count where all addresses used for pointer #1 - 18,921,100
     Block count where all addresses used for pointer #2 - 15,061,100
     Block count where all addresses used for pointer #3 - 18,107,100
     Block count where all addresses used for pointer #4 - 35,870,100

E1P1 = 1,838,819, E1P2 = 863,916, E1P3 = 1,084,114, E2P = 130, E2O = 78:
     Block count where all addresses used for pointer #1 - 16,574,700
     Block count where all addresses used for pointer #2 - 16,691,200
     Block count where all addresses used for pointer #3 - 16,518,700
     Block count where all addresses used for pointer #4 - 29,480,300

E1P1 = 1,851,372, E1P2 = 1,684,755, E1P3 = 633,629, E2P = 182, E2O = 180:
     Block count where all addresses used for pointer #1 - 17,478,600
     Block count where all addresses used for pointer #2 - 18,194,900
     Block count where all addresses used for pointer #3 - 15,954,200
     Block count where all addresses used for pointer #4 - 30,284,800

E1P1 = 1,857,295, E1P2 = 1,469,099, E1P3 = 563,542, E2P = 182, E2O = 95:
     Block count where all addresses used for pointer #1 - 15,478,700
     Block count where all addresses used for pointer #2 - 18,933,300
     Block count where all addresses used for pointer #3 - 16,724,300
     Block count where all addresses used for pointer #4 - 33,101,800

E1P1 = 1,869,143, E1P2 = 455,377, E1P3 = 1,034,213, E2P = 151, E2O = 117:
     Block count where all addresses used for pointer #1 - 17,424,800
     Block count where all addresses used for pointer #2 - 16,938,100
     Block count where all addresses used for pointer #3 - 16,170,500
     Block count where all addresses used for pointer #4 - 35,469,400

E1P1 = 1,895,878, E1P2 = 1,455,492, E1P3 = 35,788, E2P = 168, E2O = 218:
     Block count where all addresses used for pointer #1 - 16,278,500
     Block count where all addresses used for pointer #2 - 17,318,300
     Block count where all addresses used for pointer #3 - 16,247,500
     Block count where all addresses used for pointer #4 - 47,667,400

E1P1 = 1,919,891, E1P2 = 1,721,759, E1P3 = 712,443, E2P = 3, E2O = 124:
     Block count where all addresses used for pointer #1 - 15,991,800
     Block count where all addresses used for pointer #2 - 20,071,400
     Block count where all addresses used for pointer #3 - 17,448,100
     Block count where all addresses used for pointer #4 - 28,114,100

E1P1 = 1,927,293, E1P2 = 526,814, E1P3 = 213,308, E2P = 114, E2O = 106:
     Block count where all addresses used for pointer #1 - 16,716,000
     Block count where all addresses used for pointer #2 - 16,987,200
     Block count where all addresses used for pointer #3 - 17,538,900
     Block count where all addresses used for pointer #4 - 32,585,800

E1P1 = 1,944,592, E1P2 = 129,574, E1P3 = 1,130,340, E2P = 103, E2O = 30:
     Block count where all addresses used for pointer #1 - 18,373,700
     Block count where all addresses used for pointer #2 - 19,269,200
     Block count where all addresses used for pointer #3 - 15,743,300
     Block count where all addresses used for pointer #4 - 29,403,800
```

```
E1P1 = 1,986,282, E1P2 = 1,353,881, E1P3 = 1,535,756, E2P = 93, E2O = 124:
     Block count where all addresses used for pointer #1 - 18,829,000
     Block count where all addresses used for pointer #2 - 20,771,100
     Block count where all addresses used for pointer #3 - 16,506,000
     Block count where all addresses used for pointer #4 - 34,175,100

  E1P1 = 10,625, E1P2 = 1,627,858, E1P3 = 371,296, E2P = 68, E2O = 204:
     Block count where all addresses used for pointer #1 - 17,387,800
     Block count where all addresses used for pointer #2 - 18,164,500
     Block count where all addresses used for pointer #3 - 15,969,900
     Block count where all addresses used for pointer #4 - 30,953,500

   E1P1 = 18,976, E1P2 = 507,384, E1P3 = 989,680, E2P = 238, E2O = 73:
     Block count where all addresses used for pointer #1 - 15,966,000
     Block count where all addresses used for pointer #2 - 17,981,300
     Block count where all addresses used for pointer #3 - 22,480,300
     Block count where all addresses used for pointer #4 - 31,567,200

   E1P1 = 181,525, E1P2 = 635,416, E1P3 = 69,263, E2P = 36, E2O = 155:
     Block count where all addresses used for pointer #1 - 17,295,100
     Block count where all addresses used for pointer #2 - 19,012,000
     Block count where all addresses used for pointer #3 - 15,315,300
     Block count where all addresses used for pointer #4 - 30,042,500

 E1P1 = 194,917, E1P2 = 1,920,679, E1P3 = 1,018,979, E2P = 88, E2O = 253:
     Block count where all addresses used for pointer #1 - 16,125,400
     Block count where all addresses used for pointer #2 - 17,031,900
     Block count where all addresses used for pointer #3 - 19,428,000
     Block count where all addresses used for pointer #4 - 29,996,600

E1P1 = 2,046,458, E1P2 = 1,167,336, E1P3 = 147,357, E2P = 242, E2O = 127:
     Block count where all addresses used for pointer #1 - 17,291,300
     Block count where all addresses used for pointer #2 - 16,840,400
     Block count where all addresses used for pointer #3 - 16,317,400
     Block count where all addresses used for pointer #4 - 30,083,100

 E1P1 = 2,065,805, E1P2 = 741,552, E1P3 = 1,348,295, E2P = 7, E2O = 206:
     Block count where all addresses used for pointer #1 - 17,027,200
     Block count where all addresses used for pointer #2 - 17,372,000
     Block count where all addresses used for pointer #3 - 16,609,100
     Block count where all addresses used for pointer #4 - 30,200,800

 E1P1 = 201,951, E1P2 = 1,772,728, E1P3 = 1,974,446, E2P = 15, E2O = 43:
     Block count where all addresses used for pointer #1 - 18,157,400
     Block count where all addresses used for pointer #2 - 16,477,900
     Block count where all addresses used for pointer #3 - 16,598,600
     Block count where all addresses used for pointer #4 - 31,381,800

  E1P1 = 208,907, E1P2 = 967,349, E1P3 = 321,593, E2P = 110, E2O = 119:
     Block count where all addresses used for pointer #1 - 16,552,000
     Block count where all addresses used for pointer #2 - 15,825,300
     Block count where all addresses used for pointer #3 - 17,270,000
     Block count where all addresses used for pointer #4 - 30,286,500

   E1P1 = 209,400, E1P2 = 723,054, E1P3 = 603,276, E2P = 93, E2O = 157:
     Block count where all addresses used for pointer #1 - 16,547,200
     Block count where all addresses used for pointer #2 - 16,427,400
     Block count where all addresses used for pointer #3 - 19,911,600
     Block count where all addresses used for pointer #4 - 29,929,500

 E1P1 = 21,191, E1P2 = 1,986,818, E1P3 = 1,792,465, E2P = 105, E2O = 151:
     Block count where all addresses used for pointer #1 - 18,162,300
     Block count where all addresses used for pointer #2 - 16,066,200
     Block count where all addresses used for pointer #3 - 18,625,700
     Block count where all addresses used for pointer #4 - 32,580,900

 E1P1 = 221,698, E1P2 = 1,103,314, E1P3 = 1,749,985, E2P = 99, E2O = 187:
     Block count where all addresses used for pointer #1 - 16,339,000
     Block count where all addresses used for pointer #2 - 16,567,600
     Block count where all addresses used for pointer #3 - 15,909,700
     Block count where all addresses used for pointer #4 - 31,536,900

 E1P1 = 224,903, E1P2 = 1,220,544, E1P3 = 886,678, E2P = 138, E2O = 242:
     Block count where all addresses used for pointer #1 - 17,501,900
     Block count where all addresses used for pointer #2 - 16,615,000
     Block count where all addresses used for pointer #3 - 19,040,300
     Block count where all addresses used for pointer #4 - 29,810,900

E1P1 = 234,065, E1P2 = 1,432,674, E1P3 = 1,628,592, E2P = 130, E2O = 246:
     Block count where all addresses used for pointer #1 - 18,178,000
     Block count where all addresses used for pointer #2 - 17,085,900
     Block count where all addresses used for pointer #3 - 16,887,300
     Block count where all addresses used for pointer #4 - 31,617,800

 E1P1 = 234,789, E1P2 = 1,825,128, E1P3 = 709,150, E2P = 100, E2O = 202:
     Block count where all addresses used for pointer #1 - 15,593,400
     Block count where all addresses used for pointer #2 - 17,368,600
     Block count where all addresses used for pointer #3 - 15,480,400
     Block count where all addresses used for pointer #4 - 30,199,400

 E1P1 = 240,385, E1P2 = 1,421,396, E1P3 = 1,115,099, E2P = 198, E2O = 79:
     Block count where all addresses used for pointer #1 - 16,822,100
     Block count where all addresses used for pointer #2 - 16,785,700
     Block count where all addresses used for pointer #3 - 15,990,400
     Block count where all addresses used for pointer #4 - 28,520,000

  E1P1 = 240,541, E1P2 = 773,505, E1P3 = 544,597, E2P = 144, E2O = 232:
     Block count where all addresses used for pointer #1 - 15,622,800
     Block count where all addresses used for pointer #2 - 16,118,600
     Block count where all addresses used for pointer #3 - 17,715,900
     Block count where all addresses used for pointer #4 - 31,293,500

  E1P1 = 244,966, E1P2 = 434,084, E1P3 = 2,017,002, E2P = 143, E2O = 95:
     Block count where all addresses used for pointer #1 - 19,811,500
     Block count where all addresses used for pointer #2 - 20,096,400
     Block count where all addresses used for pointer #3 - 16,359,100
     Block count where all addresses used for pointer #4 - 29,907,600

   E1P1 = 250,202, E1P2 = 1,696,073, E1P3 = 427,260, E2P = 16, E2O = 6:
     Block count where all addresses used for pointer #1 - 15,083,200
     Block count where all addresses used for pointer #2 - 18,763,100
     Block count where all addresses used for pointer #3 - 17,106,700
     Block count where all addresses used for pointer #4 - 32,314,700

  E1P1 = 253,061, E1P2 = 1,104,585, E1P3 = 542,589, E2P = 38, E2O = 104:
     Block count where all addresses used for pointer #1 - 18,356,300
     Block count where all addresses used for pointer #2 - 15,000,100
     Block count where all addresses used for pointer #3 - 19,211,000
     Block count where all addresses used for pointer #4 - 35,771,700
```

```
E1P1 = 256,747, E1P2 = 1,187,607, E1P3 = 1,977,362, E2P = 224, E2O = 84:
     Block count where all addresses used for pointer #1 - 15,201,400
     Block count where all addresses used for pointer #2 - 18,673,100
     Block count where all addresses used for pointer #3 - 16,886,900
     Block count where all addresses used for pointer #4 - 34,833,800

 E1P1 = 261,282, E1P2 = 55,025, E1P3 = 1,654,147, E2P = 201, E2O = 225:
     Block count where all addresses used for pointer #1 - 15,685,200
     Block count where all addresses used for pointer #2 - 16,741,700
     Block count where all addresses used for pointer #3 - 17,394,400
     Block count where all addresses used for pointer #4 - 33,046,600

  E1P1 = 290,694, E1P2 = 1,576,517, E1P3 = 649,095, E2P = 33, E2O = 30:
     Block count where all addresses used for pointer #1 - 20,299,200
     Block count where all addresses used for pointer #2 - 17,521,000
     Block count where all addresses used for pointer #3 - 15,392,600
     Block count where all addresses used for pointer #4 - 29,160,300

E1P1 = 316,522, E1P2 = 1,446,296, E1P3 = 1,722,256, E2P = 222, E2O = 84:
     Block count where all addresses used for pointer #1 - 18,448,500
     Block count where all addresses used for pointer #2 - 18,834,600
     Block count where all addresses used for pointer #3 - 15,874,300
     Block count where all addresses used for pointer #4 - 29,774,600

  E1P1 = 331,421, E1P2 = 1,309,697, E1P3 = 572,756, E2P = 94, E2O = 99:
     Block count where all addresses used for pointer #1 - 15,121,400
     Block count where all addresses used for pointer #2 - 16,203,600
     Block count where all addresses used for pointer #3 - 17,064,500
     Block count where all addresses used for pointer #4 - 32,268,600

  E1P1 = 343,102, E1P2 = 885,148, E1P3 = 524,546, E2P = 228, E2O = 198:
     Block count where all addresses used for pointer #1 - 17,086,700
     Block count where all addresses used for pointer #2 - 16,634,900
     Block count where all addresses used for pointer #3 - 17,868,200
     Block count where all addresses used for pointer #4 - 31,058,700

 E1P1 = 387,970, E1P2 = 930,513, E1P3 = 1,683,556, E2P = 193, E2O = 125:
     Block count where all addresses used for pointer #1 - 16,887,400
     Block count where all addresses used for pointer #2 - 17,243,100
     Block count where all addresses used for pointer #3 - 18,719,400
     Block count where all addresses used for pointer #4 - 32,727,000

  E1P1 = 41,109, E1P2 = 1,906,967, E1P3 = 388,371, E2P = 53, E2O = 132:
     Block count where all addresses used for pointer #1 - 19,571,100
     Block count where all addresses used for pointer #2 - 16,007,800
     Block count where all addresses used for pointer #3 - 16,257,800
     Block count where all addresses used for pointer #4 - 39,329,300

E1P1 = 419,411, E1P2 = 1,678,814, E1P3 = 2,022,588, E2P = 67, E2O = 239:
     Block count where all addresses used for pointer #1 - 15,765,600
     Block count where all addresses used for pointer #2 - 19,244,400
     Block count where all addresses used for pointer #3 - 16,156,400
     Block count where all addresses used for pointer #4 - 32,038,900

   E1P1 = 42,726, E1P2 = 1,587,492, E1P3 = 979,945, E2P = 195, E2O = 8:
     Block count where all addresses used for pointer #1 - 16,130,100
     Block count where all addresses used for pointer #2 - 16,990,700
     Block count where all addresses used for pointer #3 - 15,639,500
     Block count where all addresses used for pointer #4 - 30,587,200

  E1P1 = 45,327, E1P2 = 1,070,634, E1P3 = 780,889, E2P = 61, E2O = 249:
     Block count where all addresses used for pointer #1 - 16,019,400
     Block count where all addresses used for pointer #2 - 17,682,400
     Block count where all addresses used for pointer #3 - 17,062,500
     Block count where all addresses used for pointer #4 - 35,355,200

  E1P1 = 450,771, E1P2 = 120,542, E1P3 = 735,293, E2P = 114, E2O = 236:
     Block count where all addresses used for pointer #1 - 15,460,000
     Block count where all addresses used for pointer #2 - 17,900,900
     Block count where all addresses used for pointer #3 - 17,755,000
     Block count where all addresses used for pointer #4 - 29,861,800

 E1P1 = 46,776, E1P2 = 1,452,335, E1P3 = 1,623,370, E2P = 176, E2O = 38:
     Block count where all addresses used for pointer #1 - 17,000,100
     Block count where all addresses used for pointer #2 - 19,532,100
     Block count where all addresses used for pointer #3 - 15,994,600
     Block count where all addresses used for pointer #4 - 31,992,500

 E1P1 = 473,014, E1P2 = 794,165, E1P3 = 1,736,248, E2P = 49, E2O = 168:
     Block count where all addresses used for pointer #1 - 16,658,800
     Block count where all addresses used for pointer #2 - 18,360,100
     Block count where all addresses used for pointer #3 - 15,669,500
     Block count where all addresses used for pointer #4 - 33,783,000

   E1P1 = 474,879, E1P2 = 94,939, E1P3 = 278,726, E2P = 108, E2O = 83:
     Block count where all addresses used for pointer #1 - 19,804,300
     Block count where all addresses used for pointer #2 - 15,397,100
     Block count where all addresses used for pointer #3 - 16,917,800
     Block count where all addresses used for pointer #4 - 32,744,400

 E1P1 = 488,363, E1P2 = 1,737,943, E1P3 = 785,106, E2P = 236, E2O = 38:
     Block count where all addresses used for pointer #1 - 15,308,100
     Block count where all addresses used for pointer #2 - 15,687,200
     Block count where all addresses used for pointer #3 - 16,197,200
     Block count where all addresses used for pointer #4 - 30,213,300

 E1P1 = 502,297, E1P2 = 244,619, E1P3 = 1,755,316, E2P = 206, E2O = 240:
     Block count where all addresses used for pointer #1 - 16,923,800
     Block count where all addresses used for pointer #2 - 20,301,500
     Block count where all addresses used for pointer #3 - 16,852,500
     Block count where all addresses used for pointer #4 - 35,348,700

  E1P1 = 530,566, E1P2 = 1,029,444, E1P3 = 199,820, E2P = 3, E2O = 255:
     Block count where all addresses used for pointer #1 - 19,564,600
     Block count where all addresses used for pointer #2 - 15,958,000
     Block count where all addresses used for pointer #3 - 17,765,800
     Block count where all addresses used for pointer #4 - 30,046,700

  E1P1 = 551,980, E1P2 = 1,342,161, E1P3 = 359,011, E2P = 7, E2O = 201:
     Block count where all addresses used for pointer #1 - 16,938,100
     Block count where all addresses used for pointer #2 - 17,258,400
     Block count where all addresses used for pointer #3 - 15,386,300
     Block count where all addresses used for pointer #4 - 38,954,400

  E1P1 = 553,294, E1P2 = 884,846, E1P3 = 207,502, E2P = 141, E2O = 239:
     Block count where all addresses used for pointer #1 - 17,143,100
     Block count where all addresses used for pointer #2 - 15,728,900
     Block count where all addresses used for pointer #3 - 16,925,000
     Block count where all addresses used for pointer #4 - 31,869,200
```

```
E1P1 = 556,073, E1P2 = 1,867,482, E1P3 = 315,976, E2P = 102, E2O = 66:
    Block count where all addresses used for pointer #1 - 17,219,600
    Block count where all addresses used for pointer #2 - 16,124,500
    Block count where all addresses used for pointer #3 - 15,805,000
    Block count where all addresses used for pointer #4 - 29,929,300

E1P1 = 557,322, E1P2 = 341,047, E1P3 = 26,161, E2P = 71, E2O = 184:
    Block count where all addresses used for pointer #1 - 18,560,700
    Block count where all addresses used for pointer #2 - 16,412,600
    Block count where all addresses used for pointer #3 - 16,621,000
    Block count where all addresses used for pointer #4 - 30,183,400

E1P1 = 558,100, E1P2 = 1,430,297, E1P3 = 1,872,267, E2P = 82, E2O = 137:
    Block count where all addresses used for pointer #1 - 16,629,900
    Block count where all addresses used for pointer #2 - 15,652,700
    Block count where all addresses used for pointer #3 - 17,657,900
    Block count where all addresses used for pointer #4 - 33,251,500

E1P1 = 606,543, E1P2 = 478,697, E1P3 = 1,509,786, E2P = 129, E2O = 206:
    Block count where all addresses used for pointer #1 - 17,474,600
    Block count where all addresses used for pointer #2 - 15,465,800
    Block count where all addresses used for pointer #3 - 16,144,500
    Block count where all addresses used for pointer #4 - 29,303,300

E1P1 = 629,809, E1P2 = 212,674, E1P3 = 1,192,592, E2P = 254, E2O = 239:
    Block count where all addresses used for pointer #1 - 19,320,300
    Block count where all addresses used for pointer #2 - 16,598,000
    Block count where all addresses used for pointer #3 - 17,236,300
    Block count where all addresses used for pointer #4 - 32,010,200

E1P1 = 64,286, E1P2 = 1,014,398, E1P3 = 1,603,037, E2P = 48, E2O = 140:
    Block count where all addresses used for pointer #1 - 15,849,900
    Block count where all addresses used for pointer #2 - 19,214,900
    Block count where all addresses used for pointer #3 - 16,946,300
    Block count where all addresses used for pointer #4 - 32,861,600

E1P1 = 644,888, E1P2 = 1,261,966, E1P3 = 1,970,094, E2P = 243, E2O = 198:
    Block count where all addresses used for pointer #1 - 17,230,700
    Block count where all addresses used for pointer #2 - 18,044,500
    Block count where all addresses used for pointer #3 - 18,317,100
    Block count where all addresses used for pointer #4 - 36,283,300

E1P1 = 647,983, E1P2 = 276,809, E1P3 = 513,149, E2P = 217, E2O = 38:
    Block count where all addresses used for pointer #1 - 16,600,700
    Block count where all addresses used for pointer #2 - 17,849,500
    Block count where all addresses used for pointer #3 - 18,592,300
    Block count where all addresses used for pointer #4 - 30,185,800

E1P1 = 650,550, E1P2 = 1,680,948, E1P3 = 1,240,249, E2P = 158, E2O = 202:
    Block count where all addresses used for pointer #1 - 20,086,700
    Block count where all addresses used for pointer #2 - 16,113,400
    Block count where all addresses used for pointer #3 - 17,504,000
    Block count where all addresses used for pointer #4 - 35,279,600

E1P1 = 651,622, E1P2 = 505,766, E1P3 = 1,465,957, E2P = 195, E2O = 176:
    Block count where all addresses used for pointer #1 - 19,081,800
    Block count where all addresses used for pointer #2 - 18,651,800
    Block count where all addresses used for pointer #3 - 15,523,500
    Block count where all addresses used for pointer #4 - 32,532,000

E1P1 = 653,016, E1P2 = 190,671, E1P3 = 1,640,042, E2P = 14, E2O = 224:
    Block count where all addresses used for pointer #1 - 16,211,100
    Block count where all addresses used for pointer #2 - 19,152,300
    Block count where all addresses used for pointer #3 - 15,324,600
    Block count where all addresses used for pointer #4 - 38,170,300

E1P1 = 663,097, E1P2 = 1,367,851, E1P3 = 1,957,332, E2P = 104, E2O = 118:
    Block count where all addresses used for pointer #1 - 19,085,600
    Block count where all addresses used for pointer #2 - 16,811,700
    Block count where all addresses used for pointer #3 - 15,962,600
    Block count where all addresses used for pointer #4 - 31,429,400

E1P1 = 696,122, E1P2 = 1,329,320, E1P3 = 973,790, E2P = 179, E2O = 254:
    Block count where all addresses used for pointer #1 - 15,784,500
    Block count where all addresses used for pointer #2 - 16,920,600
    Block count where all addresses used for pointer #3 - 17,857,700
    Block count where all addresses used for pointer #4 - 29,217,600

E1P1 = 703,294, E1P2 = 1,706,781, E1P3 = 348,415, E2P = 110, E2O = 153:
    Block count where all addresses used for pointer #1 - 15,000,100
    Block count where all addresses used for pointer #2 - 19,418,400
    Block count where all addresses used for pointer #3 - 16,065,500
    Block count where all addresses used for pointer #4 - 29,927,200

E1P1 = 741,968, E1P2 = 222,568, E1P3 = 385,952, E2P = 226, E2O = 45:
    Block count where all addresses used for pointer #1 - 15,262,600
    Block count where all addresses used for pointer #2 - 18,264,100
    Block count where all addresses used for pointer #3 - 16,576,800
    Block count where all addresses used for pointer #4 - 33,174,100

E1P1 = 791,952, E1P2 = 1,950,759, E1P3 = 183,777, E2P = 219, E2O = 195:
    Block count where all addresses used for pointer #1 - 20,172,500
    Block count where all addresses used for pointer #2 - 19,369,100
    Block count where all addresses used for pointer #3 - 15,864,800
    Block count where all addresses used for pointer #4 - 33,349,600

E1P1 = 829,047, E1P2 = 267,762, E1P3 = 641,793, E2P = 21, E2O = 130:
    Block count where all addresses used for pointer #1 - 16,497,400
    Block count where all addresses used for pointer #2 - 16,171,900
    Block count where all addresses used for pointer #3 - 15,467,200
    Block count where all addresses used for pointer #4 - 33,668,900

E1P1 = 829,069, E1P2 = 542,000, E1P3 = 962,246, E2P = 45, E2O = 2:
    Block count where all addresses used for pointer #1 - 18,188,700
    Block count where all addresses used for pointer #2 - 16,122,300
    Block count where all addresses used for pointer #3 - 17,720,500
    Block count where all addresses used for pointer #4 - 30,364,400

E1P1 = 829,864, E1P2 = 569,949, E1P3 = 394,175, E2P = 62, E2O = 90:
    Block count where all addresses used for pointer #1 - 17,162,300
    Block count where all addresses used for pointer #2 - 18,673,600
    Block count where all addresses used for pointer #3 - 16,471,300
    Block count where all addresses used for pointer #4 - 33,448,900

E1P1 = 832,256, E1P2 = 1,877,334, E1P3 = 654,804, E2P = 67, E2O = 255:
    Block count where all addresses used for pointer #1 - 16,608,800
    Block count where all addresses used for pointer #2 - 16,640,600
    Block count where all addresses used for pointer #3 - 20,387,300
    Block count where all addresses used for pointer #4 - 29,668,700

E1P1 = 834,060, E1P2 = 1,601,331, E1P3 = 2,032,446, E2P = 225, E2O = 65:
    Block count where all addresses used for pointer #1 - 16,707,900
    Block count where all addresses used for pointer #2 - 16,606,900
    Block count where all addresses used for pointer #3 - 17,308,900
    Block count where all addresses used for pointer #4 - 31,554,800

E1P1 = 838,264, E1P2 = 1,154,288, E1P3 = 220,424, E2P = 91, E2O = 9:
    Block count where all addresses used for pointer #1 - 18,053,800
    Block count where all addresses used for pointer #2 - 15,940,500
    Block count where all addresses used for pointer #3 - 17,224,300
    Block count where all addresses used for pointer #4 - 29,725,000

E1P1 = 846,298, E1P2 = 2,091,208, E1P3 = 1,342,334, E2P = 96, E2O = 91:
    Block count where all addresses used for pointer #1 - 16,555,200
    Block count where all addresses used for pointer #2 - 18,232,600
    Block count where all addresses used for pointer #3 - 15,116,400
    Block count where all addresses used for pointer #4 - 31,318,200

E1P1 = 853,474, E1P2 = 1,729,457, E1P3 = 1,088,964, E2P = 7, E2O = 103:
    Block count where all addresses used for pointer #1 - 17,156,700
    Block count where all addresses used for pointer #2 - 16,958,400
    Block count where all addresses used for pointer #3 - 16,278,500
    Block count where all addresses used for pointer #4 - 30,192,300

E1P1 = 88,935, E1P2 = 352,161, E1P3 = 636,018, E2P = 140, E2O = 76:
    Block count where all addresses used for pointer #1 - 16,557,900
    Block count where all addresses used for pointer #2 - 16,470,100
    Block count where all addresses used for pointer #3 - 18,909,100
    Block count where all addresses used for pointer #4 - 29,589,000

E1P1 = 887,177, E1P2 = 1,893,309, E1P3 = 588,961, E2P = 161, E2O = 80:
    Block count where all addresses used for pointer #1 - 16,833,500
    Block count where all addresses used for pointer #2 - 17,471,800
    Block count where all addresses used for pointer #3 - 16,520,000
    Block count where all addresses used for pointer #4 - 32,550,400

E1P1 = 891,831, E1P2 = 429,490, E1P3 = 1,072,448, E2P = 87, E2O = 136:
    Block count where all addresses used for pointer #1 - 19,542,200
    Block count where all addresses used for pointer #2 - 16,822,200
    Block count where all addresses used for pointer #3 - 17,797,900
    Block count where all addresses used for pointer #4 - 32,376,400

E1P1 = 919,585, E1P2 = 1,206,771, E1P3 = 746,493, E2P = 96, E2O = 105:
    Block count where all addresses used for pointer #1 - 19,671,900
    Block count where all addresses used for pointer #2 - 16,341,600
    Block count where all addresses used for pointer #3 - 16,364,500
    Block count where all addresses used for pointer #4 - 34,660,500

E1P1 = 924,345, E1P2 = 1,786,156, E1P3 = 1,159,507, E2P = 21, E2O = 70:
    Block count where all addresses used for pointer #1 - 17,213,800
    Block count where all addresses used for pointer #2 - 16,186,900
    Block count where all addresses used for pointer #3 - 16,955,500
    Block count where all addresses used for pointer #4 - 27,906,600

E1P1 = 95,471, E1P2 = 1,411,722, E1P3 = 380,217, E2P = 235, E2O = 153:
    Block count where all addresses used for pointer #1 - 15,805,900
    Block count where all addresses used for pointer #2 - 17,169,900
    Block count where all addresses used for pointer #3 - 18,346,300
    Block count where all addresses used for pointer #4 - 29,920,700

E1P1 = 953,628, E1P2 = 1,326,212, E1P3 = 116,683, E2P = 26, E2O = 113:
    Block count where all addresses used for pointer #1 - 15,859,900
    Block count where all addresses used for pointer #2 - 17,637,100
    Block count where all addresses used for pointer #3 - 17,350,900
    Block count where all addresses used for pointer #4 - 32,394,100

E1P1 = 958,355, E1P2 = 1,438,238, E1P3 = 712,700, E2P = 11, E2O = 224:
    Block count where all addresses used for pointer #1 - 15,242,200
    Block count where all addresses used for pointer #2 - 15,804,700
    Block count where all addresses used for pointer #3 - 16,422,000
    Block count where all addresses used for pointer #4 - 28,762,100

E1P1 = 965,070, E1P2 = 1,765,740, E1P3 = 260,116, E2P = 174, E2O = 187:
    Block count where all addresses used for pointer #1 - 15,603,400
    Block count where all addresses used for pointer #2 - 16,293,400
    Block count where all addresses used for pointer #3 - 22,069,200
    Block count where all addresses used for pointer #4 - 30,089,200

E1P1 = 97,502, E1P2 = 302,526, E1P3 = 1,512,094, E2P = 81, E2O = 188:
    Block count where all addresses used for pointer #1 - 15,644,400
    Block count where all addresses used for pointer #2 - 18,505,500
    Block count where all addresses used for pointer #3 - 19,538,600
    Block count where all addresses used for pointer #4 - 38,379,700

E1P1 = 973,347, E1P2 = 1,278,574, E1P3 = 1,535,756, E2P = 95, E2O = 92:
    Block count where all addresses used for pointer #1 - 16,469,400
    Block count where all addresses used for pointer #2 - 15,796,200
    Block count where all addresses used for pointer #3 - 16,481,900
    Block count where all addresses used for pointer #4 - 28,988,500

E1P1 = 993,322, E1P2 = 1,492,312, E1P3 = 1,384,269, E2P = 3, E2O = 202:
    Block count where all addresses used for pointer #1 - 15,320,600
    Block count where all addresses used for pointer #2 - 15,283,900
    Block count where all addresses used for pointer #3 - 18,931,300
    Block count where all addresses used for pointer #4 - 30,496,800

E1P1 = 999,312, E1P2 = 1,808,166, E1P3 = 1,532,387, E2P = 60, E2O = 212:
    Block count where all addresses used for pointer #1 - 15,147,400
    Block count where all addresses used for pointer #2 - 17,777,500
    Block count where all addresses used for pointer #3 - 17,733,000
    Block count where all addresses used for pointer #4 - 32,524,700
```

Appendix E

This is the code used to produce the initial 2,048 byte key table for this Vernam Two system. ‘GRN’ accesses the random number generator and returns a random value from 0 to 255 inclusive. ‘ls’ merely changes the number to an ASCII string and truncates the leading space character.

```
Sub MakeKeyTable()
    Dim i As Integer, used As String, n As Integer

    ' Do loop to make a proper 2,048-value randomly set key
    Do
        ' Initialize the used number string as 256 – ‘0’ digits
        used = String$(256, "0")
        ' Loop to initialize the key table and record the value that was set
        For i = 0 To 2047: n = GRN: e1Key(i) = n: Mid$(used, n + 1, 1) = "1": Next i
    ' Keep going if not all values between 0 and 255 were in the key
    Loop While used <> String$(256, "1")
    ' Open the key table for writing
    Open App.Path + "\vernamTwo.tbl" For Output As #1
    ' Write the key table to the file
    For i = 0 To 2046: Print #1, ls(e1Key(i)); ",";: Next i: Print #1, ls(e1Key(i))
    ' Close the file so the input function operates
    Close #1
End Sub
```

The 2,048 key numbers used for this technology demonstration document:

Copy/paste this text into a text editor if you wish to use this key.

```
111,204,227,8,145,67,11,91,214,111,168,200,76,86,92,172,227,123,35,131,95,11,180,32,233,75,104,28,236,59,128,248,178,50,42,60,116,247,201,252,26,63
82,206,38,216,98,142,194,118,243,172,96,240,178,107,87,71,86,29,71,140,129,7,44,225,179,249,42,36,178,70,235,16,229,235,224,11,37,26,178,247,44,213
129,37,206,159,107,77,245,235,71,88,81,48,216,203,118,30,141,76,185,51,3,211,153,204,30,235,46,35,189,89,61,151,245,215,157,221,222,188,172,203,139
71,219,253,197,167,47,106,223,169,18,245,146,76,168,91,88,78,75,74,55,172,251,243,61,105,166,59,76,58,37,212,205,212,173,158,127,24,71,220,55,205,240
140,41,237,213,87,237,27,72,19,113,198,135,247,206,49,215,102,190,190,186,210,232,74,85,128,175,106,153,223,190,213,179,82,144,76,12,200,164,7,85,87
173,153,113,231,54,49,79,84,78,163,246,23,40,213,240,94,134,102,46,160,214,117,86,226,153,227,33,116,85,50,245,1,229,141,38,118,125,98,252,102,112,27
219,250,140,125,174,117,126,152,95,219,157,106,136,106,164,157,19,48,63,118,189,94,14,134,28,97,87,17,214,24,41,214,165,196,204,53,216,54,220,152,46
187,122,145,199,63,215,59,195,229,213,128,84,143,16,241,245,186,64,91,152,197,248,10,20,226,198,167,125,229,2,26,247,29,174,154,95,49,115,138,9,137
21,59,208,194,222,72,95,8,43,242,216,93,208,158,181,85,195,243,170,232,203,217,15,35,198,241,158,214,66,13,49,69,223,149,130,19,109,105,61,155,250,1
227,184,39,169,149,45,17,14,83,114,98,147,193,31,128,164,104,195,67,59,142,222,214,225,148,193,214,70,49,84,21,78,67,222,168,168,34,42,29,163,37,59
232,132,87,99,80,244,186,11,147,228,228,99,100,38,9,61,221,22,30,88,90,32,79,96,129,251,168,34,100,11,178,5,252,84,183,83,173,18,97,83,21,60,86,87,0
115,203,43,234,182,245,11,106,5,23,195,82,197,234,37,84,122,118,87,59,5,163,107,246,83,122,231,113,5,148,174,102,47,254,246,228,43,161,203,175,119
105,210,180,20,102,198,203,135,178,232,109,22,90,120,166,183,155,156,148,18,113,157,242,249,210,157,214,154,166,72,155,221,163,200,118,82,157,251,115
213,182,183,221,205,163,125,169,248,148,25,126,254,186,97,238,252,36,124,76,193,125,243,172,186,179,34,18,15,231,58,84,32,65,190,221,80,42,39,236,48
62,4,112,194,72,78,207,133,236,98,246,161,229,86,190,176,11,211,32,26,182,215,133,220,152,102,208,204,158,207,87,152,33,217,170,222,149,2,198,174,1,1
2,205,177,106,17,178,28,123,110,163,61,95,153,5,242,200,60,40,87,112,12,130,176,160,117,147,128,243,110,115,119,105,36,187,196,252,230,32,237,20,245
90,221,161,139,249,146,192,66,232,242,213,140,210,52,145,176,170,99,161,117,97,13,215,44,211,56,246,128,163,44,36,56,182,137,56,5,70,39,200,233,176
184,24,184,112,235,79,26,87,50,112,112,51,173,210,252,227,5,113,255,174,247,1,190,176,230,73,231,32,123,170,79,75,13,179,98,103,167,65,72,31,63,177
217,53,228,73,201,223,37,145,53,21,60,243,55,241,242,103,47,6,157,245,224,117,149,38,60,167,82,50,217,81,214,44,129,196,28,39,95,176,84,238,248,145
117,215,119,178,26,156,235,81,234,46,181,34,156,37,182,178,1,160,234,134,137,203,72,61,192,142,43,182,63,88,190,250,66,4,214,248,110,99,245,232,27
112,210,131,150,228,241,180,153,93,35,117,181,180,201,132,144,203,95,176,10,13,83,74,223,244,71,198,230,229,234,224,79,200,44,33,226,186,250,142,199
121,241,113,102,64,10,239,160,231,52,103,176,121,106,188,175,114,177,151,117,99,130,77,20,252,231,6,233,21,156,204,55,90,176,111,22,131,212,204,100
225,133,129,195,228,255,169,78,120,168,221,184,193,114,12,24,44,60,136,42,198,73,245,211,250,252,56,129,249,169,153,0,253,2,95,243,185,201,129,75,99
39,71,117,51,165,115,124,195,31,88,95,204,194,192,203,246,13,183,180,231,10,153,253,142,208,106,33,49,29,109,34,142,224,222,81,174,4,55,39,83,175,170
58,237,155,112,225,48,106,127,30,40,207,13,56,136,51,32,173,5,212,76,74,109,126,142,186,155,69,137,57,124,182,72,82,176,232,254,111,75,22,212,123,147
43,29,57,84,182,19,27,50,236,41,44,120,0,42,224,27,73,187,244,172,36,79,147,73,65,220,156,65,29,128,78,79,4,243,94,90,156,86,110,169,29,91,100,40,153
242,56,189,213,150,55,43,83,74,1,158,247,43,74,0,186,44,200,205,156,14,69,118,104,242,17,182,227,26,128,112,4,81,174,27,212,202,241,201,119,123,179
41,245,216,20,23,195,163,204,45,185,97,68,82,65,48,254,222,43,94,34,231,95,66,55,112,13,222,78,103,192,57,104,138,162,55,229,199,225,197,250,203,150
230,99,215,59,189,139,175,105,170,223,99,162,116,78,179,199,253,245,67,54,178,56,4,126,241,163,92,27,243,27,236,238,20,21,75,19,110,59,133,89,193,174
192,84,99,222,213,99,65,156,231,19,205,52,120,226,51,130,209,146,104,9,158,35,91,253,26,242,92,206,78,34,139,241,185,115,143,102,201,63,214,117,254
221,151,189,191,14,57,151,17,135,214,99,225,106,226,48,58,87,146,119,78,28,180,141,46,67,237,103,38,13,64,157,228,14,194,114,239,119,50,212,130,204
205,229,38,71,177,217,131,121,250,25,96,113,50,219,146,19,130,203,91,224,32,89,218,78,108,120,188,153,117,61,233,92,228,109,250,145,91,237,186,27,18
232,45,69,160,143,23,236,205,244,10,40,169,247,143,171,70,236,205,46,200,61,68,240,172,16,146,99,104,43,172,82,80,73,223,200,39,134,195,126,67,152
245,208,202,41,109,29,147,34,175,62,174,178,219,11,123,172,138,236,168,93,243,19,110,7,162,154,153,164,70,119,70,213,22,85,79,165,57,131,108,217,179
31,139,253,5,75,237,235,238,70,3,72,184,117,65,83,18,223,22,240,227,41,170,201,56,254,231,243,178,223,176,227,185,8,151,255,178,123,204,37,41,157,20
219,165,15,136,39,103,230,34,74,27,174,80,226,120,157,138,87,226,184,7,4,59,45,186,221,111,125,114,24,87,13,169,96,53,14,157,63,130,244,237,250,211
102,255,84,67,46,159,193,40,246,4,7,233,186,201,1,57,247,66,32,65,214,99,229,43,103,56,164,139,6,62,4,69,92,222,0,149,54,92,180,37,91,55,168,3,41,55
233,173,164,67,200,132,134,27,122,174,165,53,85,27,47,81,40,132,157,133,226,251,222,89,227,40,252,245,181,165,74,244,0,235,229,43,114,60,92,8,220,113
151,30,90,138,36,243,48,15,177,1,79,122,221,136,77,65,35,94,240,145,48,23,100,105,95,143,235,49,125,177,227,11,247,235,116,81,169,244,189,13,48,186
210,202,177,33,176,110,170,5,29,199,127,168,9,37,61,120,147,195,8,224,86,108,25,255,230,41,28,38,86,190,121,115,83,241,206,26,108,246,99,255,17,239
149,125,72,180,121,95,83,134,227,234,226,36,11,95,82,31,111,238,54,248,244,77,223,193,104,87,65,135,130,42,130,4,234,89,89,82,70,137,87,98,175,74,135
128,52,105,130,93,134,147,87,58,152,15,51,163,107,139,196,177,242,61,101,31,31,179,207,200,63,235,153,42,36,192,90,96,33,95,107,83,230,43,81,151,215
74,252,39,73,55,210,34,83,136,16,242,21,102,67,170,226,31,185,33,226,217,253,116,100,31,199,184,21,41,91,0,101,174,45,108,39,207,118,203,94,19,249
167,142,20,141,46,61,7,193,23,219,123,245,169,44,251,221,150,5,30,79,229,252,62,59,78,233,2,51,25,132,144,58,132,141,169,75,163,244,181,128,184,50
140,128,232,158,205,52,16,121,133,40,193,87,158,128,80,202,223,71,210,170,162,121,250,187,108,77,120,149,48,232,213,228,176,112,86,240,238,88,128,99
239,243,45,251,235,98,15,213,94,207,183,13,145,178,73,26,18,163,255,30,50,239,86,33,22,201,12,148,23,74,30,4,130,108,126,236,91,36,15,156,118,20,160
187,229,239,52,199,22,78,220,204,12,174,240,245,223,136,144,117,192,154,33,30,181,213,55,230,210,172,21,9,128,8,187,146,130,186,123,87,16,213,202,38
244,83,82,36,49,21,107,216,148,233,81,24,136,61,180,113,231,154,97,189,30,149,6,169,251,6,211,195,40,152,49,227,90,234,237,61,223,61,236,209,78,77
232,242,1,166,203,114,60,195,92,231,244,115,211,63,16,236,60,10,88,14,190,254,120,223,157,64,203,161,134,156,126,173,118,83,253,74,192,148,243,17,83
198,225,30,73,82
```

The code used to convert this 2,048 byte key into an 8,389,631 byte key is as follows:

```
' This function loads the the key.
Sub loadKeyTable()
    Dim vTblPtr As Long, trTblPtr As Integer, trPosPtr As Integer, v2Ptr As Long, i As Long, j As Integer, setToNum As Integer, p1 As Long, _
        p2 As Long, tempStr As String, prevKey As Long, v2limit As Long, p3 As Integer, p4 As Integer, origKey(2047) As Byte, pass1 As Boolean, _
        cNum As Byte, mstrXposeKey(4096) As Long, numCnt(255) As Long, p5 As Integer, s1 As Integer, e1 As Long, cnt(5) As Long, pCnt As Byte, _
        lastDiff As Integer

    Open App.Path + "\vernamTwo.tbl" For Input As #1 ' Open the key table file and initialize the variables used to accept 256 characters for the two 256-number keys
    For i = 0 To 2047: Input #1, e1Key(i): origKey(i) = e1Key(i): Next i    ' Input the 512 random numbers
    Close #1    ' Close the table file
        trTblPtr = 0: pCnt = 0    ' Initialize the variables
    For vTblPtr = &H800& To &H800000 Step &H800&   ' Now to create the 8 megabyte Vernam 2 key from these 2,048 bytes
        ' If not halfway in the construction,
        If vTblPtr <> &H400000 Then
            If (vTblPtr + &H7FF&) < 8389631 Then p1 = vTblPtr + &H7FF& Else p1 = 8389631            ' Initialize the reverse and end pointers
            For i = vTblPtr - &H800& To vTblPtr - 4 Step 4: e1Key(p1) = (e1Key(i) + 1) And &HFF&: p1 = p1 - 1: Next i            ' Copy steps of 4, back loading direction
            For i = vTblPtr - &H7FF& To vTblPtr - 3 Step 4: e1Key(p1) = (e1Key(i) + 1) And &HFF&: p1 = p1 - 1: Next i
            For i = vTblPtr - &H7FE& To vTblPtr - 2 Step 4: e1Key(p1) = (e1Key(i) + 1) And &HFF&: p1 = p1 - 1: Next i
            For i = vTblPtr - &H7FD& To vTblPtr - 1 Step 4: e1Key(p1) = (e1Key(i) + 1) And &HFF&: p1 = p1 - 1: Next i
            i = 9
        Else
            v2Ptr = vTblPtr - &H800&: trPosPtr = 1: tempStr = String$(&H100&, "0")    ' Init previous 512-number key pointers and the numbers used string
            p2 = vTblPtr - 1            ' Set the pointer to look at the topmost value for the second attempt at extraction
            For i = v2Ptr To (v2Ptr + &H800&)           ' Loop to extract the transposition key from the 512 numbers
                setToNum = e1Key(i) + 1
                If Mid$(tempStr, setToNum, 1) = "0" Then _
                    tk(0, trPosPtr) = setToNum: Mid$(tempStr, setToNum, 1) = "1": If trPosPtr < 256 Then trPosPtr = trPosPtr + 1 Else Exit For
                setToNum = e1Key(p2) + 1: p2 = p2 - 1
                If Mid$(tempStr, setToNum, 1) = "0" Then _
                    tk(0, trPosPtr) = setToNum: Mid$(tempStr, setToNum, 1) = "1": If trPosPtr < 256 Then trPosPtr = trPosPtr + 1 Else Exit For
            Next i
            For i = 1 To 8195            ' We need to make 8,194 more sets of transposition keys from the key just formed
                lastDiff = 0
                Do                ' Need to make sure no location being transposed is too close to the previous location
                    p1 = 0: p2 = 256: p4 = i - 1: p5 = 1
                    For j = 2 To 256
                        If Abs(tk(p4, j) - tk(p4, j - 1)) < 3 Then
                            Do While Abs(tk(p4, p2) - tk(p4, j)) < 3 Or p2 = j Or lastDiff = Abs(tk(p4, p2) - tk(p4, j))
                                p2 = p2 - p5
                                If p2 < 1 Then p2 = 256
                            Loop
                            p3 = tk(p4, j): tk(p4, j) = tk(p4, p2): tk(p4, p2) = p3: lastDiff = Abs(tk(p4, p2) - tk(p4, j))
                            p1 = p1 + 1
                        End If
                    Next j
                    p5 = p5 + 1
                Loop While p1 > 0
                If (i And 1) = 0 Then                ' Every even key,
                    p2 = i And 63: p4 = i - 1                    ' Form it from the previous key
                    For p1 = 1 To 256: tk(i, p1) = ((tk(p4, tk(p4, p1)) + p2) Mod 256) + 1: Next p1
                Else
                    p2 = i And 31: p4 = i - 1
                    For p1 = 1 To 256: tk(i, tk(p4, p1)) = ((p1 + p2) Mod 256) + 1: Next p1                    ' Form it as the 'decrypt' version of the previous key
                End If
            Next i
            For i = 0 To 8195            ' Copy the first 255 values of each of the keys to the upper 255 locations
                For p1 = 1 To 255: tk(i, p1 + 256) = tk(i, p1): Next p1
            Next i
            tempStr = String$(2048, "0"): p3 = 1: p4 = 0            ' Now we have to  make a MASTER transposition key because we have 2,048 numbers to transpose in one operation
            Do   ' Loop through the transposition keys just made until the master key is produced
                For j = 1 To 256            ' Loop through all the positions in each key
                    p2 = tk(p4, j) + ((j And 7) * 256)                    ' Get the table value, load it and
                    Do While Mid$(tempStr, p2, 1) = "1": p2 = ((p2 + 255) And &H7FF&) + 1: Loop                    ' If the position is needed, place the value
                    mstrXposeKey(p3) = p2: Mid$(tempStr, p2, 1) = "1": p3 = p3 + 1
                Next j
                p4 = p4 + 883
            Loop While tempStr <> String$(2048, "1")
            For i = 1 To 2047: mstrXposeKey(i + 2048) = mstrXposeKey(i): Next i            ' Copy the table to the upper part
            ' Set the transposition position and key number pointers
            p1 = 1: p2 = vTblPtr + 2047
            p3 = ((vTblPtr / &H800&) Mod 2040) + 7
            For i = vTblPtr To p2: e1Key(i) = (e1Key(v2Ptr + mstrXposeKey(p3) - 1) + 1) And &HFF&: p3 = p3 + 1: Next i  ' Transpose the previous Vername 2,048-number table
        End If
        If (vTblPtr Mod &H8000) = 0 Then DoEvents        ' Allow any other activity
    Next vTblPtr
    ' Now to find ALL values in the 4 sections that are identical and replace them with numbers from the original key
    p3 = &H7FF&: cnt(0) = 0: cnt(1) = 0: cnt(2) = 0: cnt(3) = 0: cnt(4) = 0: cnt(5) = 0: pass1 = True
    Do
        cNum = 0
        If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = 0:       p2 = &H200100: GoSub 100 ' Sets 1 and 2
        cNum = cNum + 1: If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = 0:       p2 = &H400200: GoSub 100 ' Sets 1 and 3
        cNum = cNum + 1: If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = 0:       p2 = 6292224: GoSub 100 ' Sets 1 and 4
        cNum = cNum + 1: If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = &H200100: p2 = &H400200: GoSub 100 ' Sets 2 and 3
        cNum = cNum + 1: If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = &H200100: p2 = 6292224: GoSub 100 ' Sets 2 and 4
        cNum = cNum + 1: If pass1 = True Or cnt(cNum) > 0 Then cnt(cNum) = 0: p1 = &H400200: p2 = 6292224: GoSub 100 ' Sets 3 and 4
        pass1 = False
    Loop While (cnt(0) + cnt(1) + cnt(2) + cnt(3) + cnt(4) + cnt(5)) > 0
    Exit Sub

100 For vTblPtr = 0 To 2097407
        ' If the two key positions are equal, look for one in the original key to replace it and count this substitution
        If e1Key(p1) = e1Key(p2) Then _
            Do: p3 = (p3 + 1) And &H7FF&: Loop While e1Key(p2) = origKey(p3): e1Key(p2) = origKey(p3): cnt(cNum) = cnt(cNum) + 1
        ' Advance the pointers
        p1 = p1 + 1: p2 = p2 + 1
        ' Allow other activities
        If (vTblPtr Mod 100000) = 0 Then DoEvents
    Next vTblPtr
    Return
End Sub
```

Appendix F

The byte distribution of the original 2,048 byte key, illustrated on page 31, is as follows, from 1 to 18 occurences:

```
Byte 0 occurs 7 times.         Byte 64 occurs 4 times.        Byte 128 occurs 14 times.      Byte 192 occurs 8 times.
Byte 1 occurs 9 times.         Byte 65 occurs 10 times.       Byte 129 occurs 7 times.       Byte 193 occurs 9 times.
Byte 2 occurs 4 times.         Byte 66 occurs 5 times.        Byte 130 occurs 12 times.      Byte 194 occurs 5 times.
Byte 3 occurs 3 times.         Byte 67 occurs 9 times.        Byte 131 occurs 5 times.       Byte 195 occurs 11 times.
Byte 4 occurs 11 times.        Byte 68 occurs 2 times.        Byte 132 occurs 6 times.       Byte 196 occurs 4 times.
Byte 5 occurs 11 times.        Byte 69 occurs 5 times.        Byte 133 occurs 6 times.       Byte 197 occurs 4 times.
Byte 6 occurs 5 times.         Byte 70 occurs 8 times.        Byte 134 occurs 8 times.       Byte 198 occurs 8 times.
Byte 7 occurs 6 times.         Byte 71 occurs 9 times.        Byte 135 occurs 5 times.       Byte 199 occurs 7 times.
Byte 8 occurs 6 times.         Byte 72 occurs 9 times.        Byte 136 occurs 8 times.       Byte 200 occurs 11 times.
Byte 9 occurs 5 times.         Byte 73 occurs 9 times.        Byte 137 occurs 5 times.       Byte 201 occurs 9 times.
Byte 10 occurs 6 times.        Byte 74 occurs 12 times.       Byte 138 occurs 5 times.       Byte 202 occurs 5 times.
Byte 11 occurs 10 times.       Byte 75 occurs 8 times.        Byte 139 occurs 7 times.       Byte 203 occurs 14 times.
Byte 12 occurs 5 times.        Byte 76 occurs 7 times.        Byte 140 occurs 5 times.       Byte 204 occurs 10 times.
Byte 13 occurs 11 times.       Byte 77 occurs 6 times.        Byte 141 occurs 5 times.       Byte 205 occurs 10 times.
Byte 14 occurs 7 times.        Byte 78 occurs 14 times.       Byte 142 occurs 8 times.       Byte 206 occurs 5 times.
Byte 15 occurs 7 times.        Byte 79 occurs 10 times.       Byte 143 occurs 5 times.       Byte 207 occurs 6 times.
Byte 16 occurs 7 times.        Byte 80 occurs 5 times.        Byte 144 occurs 4 times.       Byte 208 occurs 5 times.
Byte 17 occurs 7 times.        Byte 81 occurs 9 times.        Byte 145 occurs 8 times.       Byte 209 occurs 2 times.
Byte 18 occurs 7 times.        Byte 82 occurs 12 times.       Byte 146 occurs 7 times.       Byte 210 occurs 10 times.
Byte 19 occurs 9 times.        Byte 83 occurs 15 times.       Byte 147 occurs 8 times.       Byte 211 occurs 7 times.
Byte 20 occurs 9 times.        Byte 84 occurs 10 times.       Byte 148 occurs 7 times.       Byte 212 occurs 7 times.
Byte 21 occurs 10 times.       Byte 85 occurs 6 times.        Byte 149 occurs 8 times.       Byte 213 occurs 14 times.
Byte 22 occurs 8 times.        Byte 86 occurs 10 times.       Byte 150 occurs 4 times.       Byte 214 occurs 13 times.
Byte 23 occurs 7 times.        Byte 87 occurs 18 times.       Byte 151 occurs 7 times.       Byte 215 occurs 8 times.
Byte 24 occurs 7 times.        Byte 88 occurs 7 times.        Byte 152 occurs 8 times.       Byte 216 occurs 6 times.
Byte 25 occurs 4 times.        Byte 89 occurs 6 times.        Byte 153 occurs 12 times.      Byte 217 occurs 7 times.
Byte 26 occurs 10 times.       Byte 90 occurs 8 times.        Byte 154 occurs 5 times.       Byte 218 occurs 1 times.
Byte 27 occurs 12 times.       Byte 91 occurs 10 times.       Byte 155 occurs 5 times.       Byte 219 occurs 7 times.
Byte 28 occurs 6 times.        Byte 92 occurs 8 times.        Byte 156 occurs 10 times.      Byte 220 occurs 6 times.
Byte 29 occurs 9 times.        Byte 93 occurs 4 times.        Byte 157 occurs 13 times.      Byte 221 occurs 11 times.
Byte 30 occurs 11 times.       Byte 94 occurs 7 times.        Byte 158 occurs 8 times.       Byte 222 occurs 11 times.
Byte 31 occurs 9 times.        Byte 95 occurs 14 times.       Byte 159 occurs 2 times.       Byte 223 occurs 14 times.
Byte 32 occurs 9 times.        Byte 96 occurs 5 times.        Byte 160 occurs 6 times.       Byte 224 occurs 7 times.
Byte 33 occurs 9 times.        Byte 97 occurs 6 times.        Byte 161 occurs 5 times.       Byte 225 occurs 7 times.
Byte 34 occurs 10 times.       Byte 98 occurs 7 times.        Byte 162 occurs 4 times.       Byte 226 occurs 11 times.
Byte 35 occurs 6 times.        Byte 99 occurs 15 times.       Byte 163 occurs 12 times.      Byte 227 occurs 12 times.
Byte 36 occurs 10 times.       Byte 100 occurs 6 times.       Byte 164 occurs 6 times.       Byte 228 occurs 9 times.
Byte 37 occurs 10 times.       Byte 101 occurs 2 times.       Byte 165 occurs 6 times.       Byte 229 occurs 12 times.
Byte 38 occurs 8 times.        Byte 102 occurs 10 times.      Byte 166 occurs 4 times.       Byte 230 occurs 8 times.
Byte 39 occurs 10 times.       Byte 103 occurs 7 times.       Byte 167 occurs 5 times.       Byte 231 occurs 12 times.
Byte 40 occurs 10 times.       Byte 104 occurs 7 times.       Byte 168 occurs 9 times.       Byte 232 occurs 11 times.
Byte 41 occurs 10 times.       Byte 105 occurs 7 times.       Byte 169 occurs 12 times.      Byte 233 occurs 8 times.
Byte 42 occurs 8 times.        Byte 106 occurs 10 times.      Byte 170 occurs 10 times.      Byte 234 occurs 8 times.
Byte 43 occurs 12 times.       Byte 107 occurs 6 times.       Byte 171 occurs 1 times.       Byte 235 occurs 12 times.
Byte 44 occurs 10 times.       Byte 108 occurs 7 times.       Byte 172 occurs 10 times.      Byte 236 occurs 11 times.
Byte 45 occurs 6 times.        Byte 109 occurs 6 times.       Byte 173 occurs 7 times.       Byte 237 occurs 9 times.
Byte 46 occurs 8 times.        Byte 110 occurs 7 times.       Byte 174 occurs 13 times.      Byte 238 occurs 6 times.
Byte 47 occurs 4 times.        Byte 111 occurs 6 times.       Byte 175 occurs 7 times.       Byte 239 occurs 6 times.
Byte 48 occurs 10 times.       Byte 112 occurs 12 times.      Byte 176 occurs 13 times.      Byte 240 occurs 8 times.
Byte 49 occurs 9 times.        Byte 113 occurs 9 times.       Byte 177 occurs 8 times.       Byte 241 occurs 9 times.
Byte 50 occurs 9 times.        Byte 114 occurs 7 times.       Byte 178 occurs 14 times.      Byte 242 occurs 11 times.
Byte 51 occurs 7 times.        Byte 115 occurs 8 times.       Byte 179 occurs 8 times.       Byte 243 occurs 14 times.
Byte 52 occurs 6 times.        Byte 116 occurs 5 times.       Byte 180 occurs 9 times.       Byte 244 occurs 11 times.
Byte 53 occurs 5 times.        Byte 117 occurs 13 times.      Byte 181 occurs 6 times.       Byte 245 occurs 16 times.
Byte 54 occurs 5 times.        Byte 118 occurs 10 times.      Byte 182 occurs 9 times.       Byte 246 occurs 8 times.
Byte 55 occurs 12 times.       Byte 119 occurs 7 times.       Byte 183 occurs 5 times.       Byte 247 occurs 9 times.
Byte 56 occurs 9 times.        Byte 120 occurs 9 times.       Byte 184 occurs 8 times.       Byte 248 occurs 6 times.
Byte 57 occurs 6 times.        Byte 121 occurs 7 times.       Byte 185 occurs 6 times.       Byte 249 occurs 5 times.
Byte 58 occurs 6 times.        Byte 122 occurs 5 times.       Byte 186 occurs 13 times.      Byte 250 occurs 10 times.
Byte 59 occurs 11 times.       Byte 123 occurs 9 times.       Byte 187 occurs 6 times.       Byte 251 occurs 7 times.
Byte 60 occurs 9 times.        Byte 124 occurs 3 times.       Byte 188 occurs 3 times.       Byte 252 occurs 11 times.
Byte 61 occurs 14 times.       Byte 125 occurs 8 times.       Byte 189 occurs 7 times.       Byte 253 occurs 8 times.
Byte 62 occurs 4 times.        Byte 126 occurs 7 times.       Byte 190 occurs 9 times.       Byte 254 occurs 7 times.
Byte 63 occurs 9 times.        Byte 127 occurs 3 times.       Byte 191 occurs 1 times.       Byte 255 occurs 7 times.
```

As you can see, the distribution is very erratic as it should be for only 2,048 randomly set numbers from 0 to 255. But to be useful for this system, it needs to be much larger and more evenly distributed. On the next page is a breakdown of the distribution after the pseudo-random expansion function developed for this system using the above key table as the input.

The byte distribution of the resulting 8,389,631 byte pseudo-randomly expanded from the 2,048 byte key, produced by the function in Appendix E, whos numeric distribution was shown is as follows:

```
Byte 0 occurs 32,746 times.      Byte 64 occurs 32,657 times.    Byte 128 occurs 32,932 times.   Byte 192 occurs 32,773 times.
Byte 1 occurs 32,809 times.      Byte 65 occurs 32,815 times.    Byte 129 occurs 32,746 times.   Byte 193 occurs 32,811 times.
Byte 2 occurs 32,662 times.      Byte 66 occurs 32,704 times.    Byte 130 occurs 32,884 times.   Byte 194 occurs 32,678 times.
Byte 3 occurs 32,633 times.      Byte 67 occurs 32,817 times.    Byte 131 occurs 32,675 times.   Byte 195 occurs 32,858 times.
Byte 4 occurs 32,855 times.      Byte 68 occurs 32,602 times.    Byte 132 occurs 32,724 times.   Byte 196 occurs 32,659 times.
Byte 5 occurs 32,851 times.      Byte 69 occurs 32,690 times.    Byte 133 occurs 32,733 times.   Byte 197 occurs 32,653 times.
Byte 6 occurs 32,680 times.      Byte 70 occurs 32,780 times.    Byte 134 occurs 32,769 times.   Byte 198 occurs 32,783 times.
Byte 7 occurs 32,708 times.      Byte 71 occurs 32,786 times.    Byte 135 occurs 32,688 times.   Byte 199 occurs 32,755 times.
Byte 8 occurs 32,723 times.      Byte 72 occurs 32,790 times.    Byte 136 occurs 32,774 times.   Byte 200 occurs 32,857 times.
Byte 9 occurs 32,709 times.      Byte 73 occurs 32,787 times.    Byte 137 occurs 32,680 times.   Byte 201 occurs 32,809 times.
Byte 10 occurs 32,715 times.     Byte 74 occurs 32,889 times.    Byte 138 occurs 32,685 times.   Byte 202 occurs 32,681 times.
Byte 11 occurs 32,826 times.     Byte 75 occurs 32,788 times.    Byte 139 occurs 32,734 times.   Byte 203 occurs 32,933 times.
Byte 12 occurs 32,687 times.     Byte 76 occurs 32,741 times.    Byte 140 occurs 32,697 times.   Byte 204 occurs 32,822 times.
Byte 13 occurs 32,856 times.     Byte 77 occurs 32,715 times.    Byte 141 occurs 32,700 times.   Byte 205 occurs 32,835 times.
Byte 14 occurs 32,740 times.     Byte 78 occurs 32,936 times.    Byte 142 occurs 32,758 times.   Byte 206 occurs 32,703 times.
Byte 15 occurs 32,756 times.     Byte 79 occurs 32,824 times.    Byte 143 occurs 32,690 times.   Byte 207 occurs 32,734 times.
Byte 16 occurs 32,748 times.     Byte 80 occurs 32,680 times.    Byte 144 occurs 32,661 times.   Byte 208 occurs 32,674 times.
Byte 17 occurs 32,750 times.     Byte 81 occurs 32,806 times.    Byte 145 occurs 32,767 times.   Byte 209 occurs 32,597 times.
Byte 18 occurs 32,723 times.     Byte 82 occurs 32,877 times.    Byte 146 occurs 32,738 times.   Byte 210 occurs 32,828 times.
Byte 19 occurs 32,800 times.     Byte 83 occurs 32,971 times.    Byte 147 occurs 32,779 times.   Byte 211 occurs 32,736 times.
Byte 20 occurs 32,794 times.     Byte 84 occurs 32,818 times.    Byte 148 occurs 32,740 times.   Byte 212 occurs 32,744 times.
Byte 21 occurs 32,809 times.     Byte 85 occurs 32,718 times.    Byte 149 occurs 32,787 times.   Byte 213 occurs 32,944 times.
Byte 22 occurs 32,780 times.     Byte 86 occurs 32,821 times.    Byte 150 occurs 32,649 times.   Byte 214 occurs 32,919 times.
Byte 23 occurs 32,758 times.     Byte 87 occurs 33,040 times.    Byte 151 occurs 32,745 times.   Byte 215 occurs 32,781 times.
Byte 24 occurs 32,744 times.     Byte 88 occurs 32,763 times.    Byte 152 occurs 32,768 times.   Byte 216 occurs 32,710 times.
Byte 25 occurs 32,666 times.     Byte 89 occurs 32,723 times.    Byte 153 occurs 32,875 times.   Byte 217 occurs 32,741 times.
Byte 26 occurs 32,827 times.     Byte 90 occurs 32,769 times.    Byte 154 occurs 32,699 times.   Byte 218 occurs 32,569 times.
Byte 27 occurs 32,879 times.     Byte 91 occurs 32,835 times.    Byte 155 occurs 32,697 times.   Byte 219 occurs 32,738 times.
Byte 28 occurs 32,708 times.     Byte 92 occurs 32,787 times.    Byte 156 occurs 32,825 times.   Byte 220 occurs 32,727 times.
Byte 29 occurs 32,789 times.     Byte 93 occurs 32,647 times.    Byte 157 occurs 32,909 times.   Byte 221 occurs 32,871 times.
Byte 30 occurs 32,858 times.     Byte 94 occurs 32,734 times.    Byte 158 occurs 32,781 times.   Byte 222 occurs 32,852 times.
Byte 31 occurs 32,813 times.     Byte 95 occurs 32,934 times.    Byte 159 occurs 32,596 times.   Byte 223 occurs 32,941 times.
Byte 32 occurs 32,794 times.     Byte 96 occurs 32,698 times.    Byte 160 occurs 32,713 times.   Byte 224 occurs 32,750 times.
Byte 33 occurs 32,796 times.     Byte 97 occurs 32,736 times.    Byte 161 occurs 32,682 times.   Byte 225 occurs 32,741 times.
Byte 34 occurs 32,832 times.     Byte 98 occurs 32,721 times.    Byte 162 occurs 32,662 times.   Byte 226 occurs 32,846 times.
Byte 35 occurs 32,717 times.     Byte 99 occurs 32,966 times.    Byte 163 occurs 32,894 times.   Byte 227 occurs 32,887 times.
Byte 36 occurs 32,820 times.     Byte 100 occurs 32,716 times.   Byte 164 occurs 32,696 times.   Byte 228 occurs 32,803 times.
Byte 37 occurs 32,840 times.     Byte 101 occurs 32,594 times.   Byte 165 occurs 32,714 times.   Byte 229 occurs 32,896 times.
Byte 38 occurs 32,768 times.     Byte 102 occurs 32,825 times.   Byte 166 occurs 32,659 times.   Byte 230 occurs 32,753 times.
Byte 39 occurs 32,835 times.     Byte 103 occurs 32,755 times.   Byte 167 occurs 32,680 times.   Byte 231 occurs 32,879 times.
Byte 40 occurs 32,809 times.     Byte 104 occurs 32,740 times.   Byte 168 occurs 32,794 times.   Byte 232 occurs 32,859 times.
Byte 41 occurs 32,829 times.     Byte 105 occurs 32,753 times.   Byte 169 occurs 32,889 times.   Byte 233 occurs 32,759 times.
Byte 42 occurs 32,767 times.     Byte 106 occurs 32,821 times.   Byte 170 occurs 32,830 times.   Byte 234 occurs 32,775 times.
Byte 43 occurs 32,866 times.     Byte 107 occurs 32,715 times.   Byte 171 occurs 32,582 times.   Byte 235 occurs 32,886 times.
Byte 44 occurs 32,847 times.     Byte 108 occurs 32,745 times.   Byte 172 occurs 32,817 times.   Byte 236 occurs 32,859 times.
Byte 45 occurs 32,724 times.     Byte 109 occurs 32,700 times.   Byte 173 occurs 32,750 times.   Byte 237 occurs 32,800 times.
Byte 46 occurs 32,770 times.     Byte 110 occurs 32,761 times.   Byte 174 occurs 32,916 times.   Byte 238 occurs 32,713 times.
Byte 47 occurs 32,661 times.     Byte 111 occurs 32,725 times.   Byte 175 occurs 32,736 times.   Byte 239 occurs 32,697 times.
Byte 48 occurs 32,835 times.     Byte 112 occurs 32,885 times.   Byte 176 occurs 32,927 times.   Byte 240 occurs 32,767 times.
Byte 49 occurs 32,795 times.     Byte 113 occurs 32,801 times.   Byte 177 occurs 32,788 times.   Byte 241 occurs 32,790 times.
Byte 50 occurs 32,800 times.     Byte 114 occurs 32,751 times.   Byte 178 occurs 32,937 times.   Byte 242 occurs 32,867 times.
Byte 51 occurs 32,737 times.     Byte 115 occurs 32,762 times.   Byte 179 occurs 32,770 times.   Byte 243 occurs 32,965 times.
Byte 52 occurs 32,718 times.     Byte 116 occurs 32,683 times.   Byte 180 occurs 32,805 times.   Byte 244 occurs 32,852 times.
Byte 53 occurs 32,705 times.     Byte 117 occurs 32,908 times.   Byte 181 occurs 32,706 times.   Byte 245 occurs 33,008 times.
Byte 54 occurs 32,680 times.     Byte 118 occurs 32,837 times.   Byte 182 occurs 32,802 times.   Byte 246 occurs 32,775 times.
Byte 55 occurs 32,888 times.     Byte 119 occurs 32,765 times.   Byte 183 occurs 32,688 times.   Byte 247 occurs 32,795 times.
Byte 56 occurs 32,802 times.     Byte 120 occurs 32,787 times.   Byte 184 occurs 32,785 times.   Byte 248 occurs 32,712 times.
Byte 57 occurs 32,713 times.     Byte 121 occurs 32,739 times.   Byte 185 occurs 32,728 times.   Byte 249 occurs 32,700 times.
Byte 58 occurs 32,710 times.     Byte 122 occurs 32,687 times.   Byte 186 occurs 32,888 times.   Byte 250 occurs 32,836 times.
Byte 59 occurs 32,875 times.     Byte 123 occurs 32,797 times.   Byte 187 occurs 32,721 times.   Byte 251 occurs 32,746 times.
Byte 60 occurs 32,791 times.     Byte 124 occurs 32,625 times.   Byte 188 occurs 32,633 times.   Byte 252 occurs 32,841 times.
Byte 61 occurs 32,945 times.     Byte 125 occurs 32,781 times.   Byte 189 occurs 32,732 times.   Byte 253 occurs 32,771 times.
Byte 62 occurs 32,647 times.     Byte 126 occurs 32,742 times.   Byte 190 occurs 32,804 times.   Byte 254 occurs 32,744 times.
Byte 63 occurs 32,803 times.     Byte 127 occurs 32,640 times.   Byte 191 occurs 32,583 times.   Byte 255 occurs 32,722 times.
```

This key is divided into 4 sets of keys, each 2,097,407 bytes each. Each set was tested against all others and itself for stream repeating. Set 1 = 0 to 2,097,407, set 2 = 2,097,408 to 4,194,815, set 3 = 4,194,816 to 6,292,223, set 4 = 6,292,224 to 8,389,631. Set comparisons executed: Sets 1 with 1, 1 with 2, 1 with 3, 1 with 4, 2 with 2, 2 with 3, 2 with 4, 3 with 3, 3 with 4 and 4 with 4. The results of each test, including the time it took and the number of ‘IF’ statements that were executed to obtain the results, are listed on the next page. Notice that at no time did any number stream longer than 5 bytes was found to exist within any single or between any two groups. At the same time, the hexadecimal number streams found were also recorded in a different file but not included here. These files are available upon request, along with a file containing the hexadecimal 8,389,631 byte key pseudo-randomly expanded.

Testing performed to confirm this system's design

To test this system, a process was developed to simulate the encryption using 100 million blocks of text, 12.8 Gigabytes of plaintext. The process created just the Effective Key Streams (**EKS**) and recorded the 5 pointer values used to create each stream. This produced a single data file almost 30 Gigabytes in size, 10 files of this size were produced.

A process was created to take the first **EKS** and compare it with all the other 99,999,999 **EKS**'s. The process did not just test to see if the stream was equal, it compared all 128 numbers in the two streams and counted the quantity of numbers that were found to be equal. The process would then input **EKS** #2 and execute the same comparison with the remaining 99,999,998 **EKS**'s. This would continue comparing the first 100 **EKS** streams with all the remaining streams. It took several days for this process to complete, the following is a representative sample of the output of the first two tests of just the first of 10 files tested:

```
Comparing EKS #1 with EKS's 2 through 100,000,000, date: 08-26-2014, time: 09:49:25:
  Key numbers matching 0 times were found 60,584,643 times.
  Key numbers matching 1 times were found 30,417,944 times.
  Key numbers matching 2 times were found 7,577,292 times.
  Key numbers matching 3 times were found 1,251,176 times.
  Key numbers matching 4 times were found 152,629 times.
  Key numbers matching 5 times were found 15,015 times.
  Key numbers matching 6 times were found 1,215 times.
  Key numbers matching 7 times were found 82 times.
  Key numbers matching 8 times were found 3 times.

Comparing EKS #2 with EKS's 3 through 100,000,000, date: 08-26-2014, time: 11:18:01:
  Key numbers matching 0 times were found 60,604,502 times.
  Key numbers matching 1 times were found 30,408,283 times.
  Key numbers matching 2 times were found 7,570,353 times.
  Key numbers matching 3 times were found 1,247,186 times.
  Key numbers matching 4 times were found 153,409 times.
  Key numbers matching 5 times were found 14,967 times.
  Key numbers matching 6 times were found 1,202 times.
  Key numbers matching 7 times were found 90 times.
  Key numbers matching 8 times were found 6 times.
```

As you can see by the date/time stamp between the two, it took approximately 1 ½ hours to perform the comparisons of all numbers in the remaining **EKS** streams for each **EKS** tested. As you can see, over 90% of the remaining streams for each tested **EKS** had less than 2 numbers identical. The most important statistic of the first 100 streams in this file is that NO stream within the 100 million stream collection equaled ANY of the first 100 streams, providing evidence of this design's ability to create **non-repeating EKS** streams can be produced with this system in spite of a fixed key being used. The testing of the entire file would obviously take years to complete, but the author feels that further testing would be just as revealing. If there was ANY weakness in the pseudo-random algorithms for pointer advancements, this examination of the first 100 **EKS** streams would be sufficient to show that it would not be able to produce what it needs to produce.

Obviously, that chance, no matter how slight (1 chance in 8,000,000,000,000,000,000) still exists, hence the stage 2 transposition engine to literally shred the output of this Vernam Two engine.

Even if it should ever be exposed that two streams had every pointer (both Vernam 2 and Transposition) identical, reverse engineering of the streams and key used would result in total failure. This is because of the Algebraic Law prohibiting the solving of a 4-unknown single equation for 1 value for each unknown and the impossible variety of transposition keys to perform its Engine 2 operation with no ability to eliminate any of the keys produced. The lack of knowledge of the makeup of the Midstream between the two engines would also complicate any effort to attack this system.

For comparison, I had my non-pseudo-random number generator (that has passed the NIST standard for pseudo-random numbers) generate another block of 12.8 billion numbers from 0 to 255, the same quantity as the 100 million **EKS**'s generated by my app. The following results were obtained:

```
Comparing key stream #1 with key streams 2 through 100,000,000, date: 09-06-2014, time: 03:30:43:
  Key numbers matching 0 times were found 60,591,110 times.
  Key numbers matching 1 times were found 30,421,265 times.
  Key numbers matching 2 times were found 7,570,745 times.
  Key numbers matching 3 times were found 1,247,623 times.
  Key numbers matching 4 times were found 152,911 times.
  Key numbers matching 5 times were found 14,984 times.
  Key numbers matching 6 times were found 1,252 times.
  Key numbers matching 7 times were found 102 times.
  Key numbers matching 8 times were found 6 times.

Comparing key stream #2 with key streams 3 through 100,000,000, date: 09-06-2014, time: 04:30:35:
  Key numbers matching 0 times were found 60,593,444 times.
  Key numbers matching 1 times were found 30,417,389 times.
  Key numbers matching 2 times were found 7,573,638 times.
  Key numbers matching 3 times were found 1,246,538 times.
  Key numbers matching 4 times were found 152,943 times.
  Key numbers matching 5 times were found 14,752 times.
  Key numbers matching 6 times were found 1,208 times.
  Key numbers matching 7 times were found 80 times.
  Key numbers matching 8 times were found 4 times.
  Key numbers matching 9 times were found 1 times.
```

Notice there is no entry 'Key numbers matching 256 times were found 'n' times' that would indicate 'n' match(es) found. The entire run showing all 100 comparisons is available for your examination; please request and I will be happy to send it to you. Notice that the results are virtually identical with the tests on the output of my application on the previous page that provides a stream of 100 million non-repeating pseudo-random numbers to use in the Vernam Two encryption engine from a fixed key of only 8 Mbytes expanded from a key of 2,048 bytes.

To test the non-repeatability within and between the 4 – 2-Megabyte key segments, tests were performed to find duplicate key streams of 4 or more numbers anywhere within and between the key segments. The 'IF statements per discovery' shows the number of checks that were performed that failed to find a duplicate string segment of 4 numbers or more for each one that was found.

Here are the results of those tests. Note that no stream of 6 or more numbers was duplicated anywhere and only 4 sets of 5 numbers was duplicated, and processing for all these tests took about 40 hours of processing time to check 8 Megabytes of key numbers, using the number of 'IF' statements stated for each test:

For testing key set 1:
Final results - IF's executed to get this = 2,208,173,362,572, IF statements per discovery = 5,320,899,668
Strings of 4 found = 415, strings of 5 found = 0, strings of 6 found = 0
Took 2 hours 37 minutes and 6.447266 seconds

For comparing key sets 1 and 2:
Final results - IF's executed to get this = 4,416,348,688,661, IF statements per discovery = 3,911,734,888
Strings of 4 found = 1,129, strings of 5 found = 0, strings of 6 found = 0
Took 5 hours 10 minutes and 34.4375 seconds

For comparing key sets 1 and 3:
Final results - IF's executed to get this = 4,416,348,722,198, IF statements per discovery = 5,527,345,084
Strings of 4 found = 799, strings of 5 found = 0, strings of 6 found = 0
Took 5 hours 13 minutes and 30.08203 seconds

For comparing key sets 1 and 4:
Final results - IF's executed to get this = 4,416,348,598,925, IF statements per discovery = 6,041,516,551
Strings of 4 found = 730, strings of 5 found = 1, strings of 6 found = 0
Took 5 hours 7 minutes and 56.97461 seconds

For testing key set 2:
Final results - IF's executed to get this = 2,208,173,396,474, IF statements per discovery = 5,087,957,134
Strings of 4 found = 433, strings of 5 found = 1, strings of 6 found = 0
Took 2 hours 35 minutes and 20.15723 seconds

For comparing key sets 2 and 3:
Final results - IF's executed to get this = 4,416,348,697,424, IF statements per discovery = 4,511,081,406
Strings of 4 found = 979, strings of 5 found = 0, strings of 6 found = 0
Took 5 hours 9 minutes and 25.27734 seconds

For comparing key sets 2 and 4:
Final results - IF's executed to get this = 4,416,348,870,631, IF statements per discovery = 5,333,754,674
Strings of 4 found = 828, strings of 5 found = 0, strings of 6 found = 0
Took 5 hours 14 minutes and 51.64453 seconds

For testing key set 3:
Final results - IF's executed to get this = 2,208,173,374,502, IF statements per discovery = 5,691,168,490
Strings of 4 found = 388, strings of 5 found = 0, strings of 6 found = 0
Took 2 hours 36 minutes and 28.13672 seconds

For comparing key sets 3 and 4:
Final results - IF's executed to get this = 4,416,348,960,759, IF statements per discovery = 4,425,199,359
Strings of 4 found = 997, strings of 5 found = 1, strings of 6 found = 0
Took 5 hours 12 minutes and 6.382813 seconds

For testing key set 4:
Final results - IF's executed to get this = 2,208,173,503,541, IF statements per discovery = 5,282,711,730
Strings of 4 found = 417, strings of 5 found = 1, strings of 6 found = 0
Took 2 hours 16 minutes and 27.64063 seconds

Appendix G

An actual encryption sequence showing the 4 register pointer sequences is on this page. On the next page is the first 112 hex digits (of a possible 256) of the Effective Key Stream that would have resulted from these sets of pointers.

```
Block #1  >>> Pointer 1 = 849,400  , pointer 2 = 1,993,583, pointer 3 = 1,638,026, pointer 4 = 679,944
Block #2  >>> Pointer 1 = 172,333  , pointer 2 = 901,793  , pointer 3 = 77,250   , pointer 4 = 1,190,614
Block #3  >>> Pointer 1 = 1,394,432, pointer 2 = 5,447    , pointer 3 = 458,773  , pointer 4 = 842,849
Block #4  >>> Pointer 1 = 381,945  , pointer 2 = 1,664,467, pointer 3 = 1,309,029, pointer 4 = 8,360
Block #5  >>> Pointer 1 = 1,515,905, pointer 2 = 71,457   , pointer 3 = 98,583   , pointer 4 = 1,193,319
Block #6  >>> Pointer 1 = 291,191  , pointer 2 = 1,557,617, pointer 3 = 1,144,772, pointer 4 = 697,766
Block #7  >>> Pointer 1 = 1,989,111, pointer 2 = 1,539,673, pointer 3 = 1,701,758, pointer 4 = 54,824
Block #8  >>> Pointer 1 = 1,990,115, pointer 2 = 212,573  , pointer 3 = 1,958,718, pointer 4 = 1,309,778
Block #9  >>> Pointer 1 = 1,099,358, pointer 2 = 2,019,526, pointer 3 = 417,488  , pointer 4 = 34,905
Block #10 >>> Pointer 1 = 284,664  , pointer 2 = 1,606,743, pointer 3 = 1,570,948, pointer 4 = 536,900
Block #11 >>> Pointer 1 = 210,934  , pointer 2 = 1,483,575, pointer 3 = 1,570,467, pointer 4 = 971,160
Block #12 >>> Pointer 1 = 1,958,868, pointer 2 = 1,342,947, pointer 3 = 251,005  , pointer 4 = 1,705,046
Block #13 >>> Pointer 1 = 35,614   , pointer 2 = 1,966,219, pointer 3 = 728,576  , pointer 4 = 1,468,835
Block #14 >>> Pointer 1 = 1,107,861, pointer 2 = 1,388,775, pointer 3 = 496,944  , pointer 4 = 1,497,051
Block #15 >>> Pointer 1 = 271,099  , pointer 2 = 1,827,874, pointer 3 = 195,556  , pointer 4 = 382,136
Block #16 >>> Pointer 1 = 1,738,550, pointer 2 = 1,464,967, pointer 3 = 472,666  , pointer 4 = 1,979,914
Block #17 >>> Pointer 1 = 609,269  , pointer 2 = 1,427,787, pointer 3 = 783,817  , pointer 4 = 1,198,736
Block #18 >>> Pointer 1 = 422,575  , pointer 2 = 1,033,842, pointer 3 = 1,224,646, pointer 4 = 861,253
Block #19 >>> Pointer 1 = 38,666   , pointer 2 = 712,855  , pointer 3 = 1,510,112, pointer 4 = 341,898
Block #20 >>> Pointer 1 = 2,079,833, pointer 2 = 1,687,484, pointer 3 = 1,857,983, pointer 4 = 872,729
Block #21 >>> Pointer 1 = 865,531  , pointer 2 = 1,813,812, pointer 3 = 1,375,149, pointer 4 = 686,427
Block #22 >>> Pointer 1 = 938,477  , pointer 2 = 912,977  , pointer 3 = 1,175,022, pointer 4 = 926,576
Block #23 >>> Pointer 1 = 1,140,639, pointer 2 = 2,036,071, pointer 3 = 499,473  , pointer 4 = 74,262
Block #24 >>> Pointer 1 = 1,037,114, pointer 2 = 1,707,987, pointer 3 = 1,260,047, pointer 4 = 1,717,803
Block #25 >>> Pointer 1 = 210,269  , pointer 2 = 1,901,365, pointer 3 = 1,400,067, pointer 4 = 1,287,770
Block #26 >>> Pointer 1 = 814,755  , pointer 2 = 257,134  , pointer 3 = 959,468  , pointer 4 = 1,939,714
Block #27 >>> Pointer 1 = 1,103,913, pointer 2 = 594,136  , pointer 3 = 1,583,376, pointer 4 = 741,160
Block #28 >>> Pointer 1 = 438,670  , pointer 2 = 976,561  , pointer 3 = 1,150,694, pointer 4 = 1,400,269
Block #29 >>> Pointer 1 = 980,487  , pointer 2 = 487,158  , pointer 3 = 1,443,694, pointer 4 = 383,797
Block #30 >>> Pointer 1 = 842,231  , pointer 2 = 1,535,193, pointer 3 = 1,701,740, pointer 4 = 1,031,179
Block #31 >>> Pointer 1 = 1,653,690, pointer 2 = 1,718,587, pointer 3 = 1,817,145, pointer 4 = 878,654
Block #32 >>> Pointer 1 = 1,681,376, pointer 2 = 39,335   , pointer 3 = 516,249  , pointer 4 = 376,199
Block #33 >>> Pointer 1 = 277,139  , pointer 2 = 1,303,610, pointer 3 = 1,741,796, pointer 4 = 657,167
Block #34 >>> Pointer 1 = 665,770  , pointer 2 = 715,304  , pointer 3 = 568,042  , pointer 4 = 741,417
Block #35 >>> Pointer 1 = 914,600  , pointer 2 = 552,436  , pointer 3 = 1,353,837, pointer 4 = 1,060,996
Block #36 >>> Pointer 1 = 1,795,827, pointer 2 = 1,309,542, pointer 3 = 455,675  , pointer 4 = 1,091,281
Block #37 >>> Pointer 1 = 2,053,927, pointer 2 = 474,967  , pointer 3 = 1,517,375, pointer 4 = 1,498,398
Block #38 >>> Pointer 1 = 1,085,315, pointer 2 = 200,847  , pointer 3 = 1,016,592, pointer 4 = 2,083,357
Block #39 >>> Pointer 1 = 1,626,265, pointer 2 = 1,685,712, pointer 3 = 1,087,928, pointer 4 = 526,088
Block #40 >>> Pointer 1 = 1,881,132, pointer 2 = 826,548  , pointer 3 = 1,322,140, pointer 4 = 1,057,023
Block #41 >>> Pointer 1 = 1,426,001, pointer 2 = 1,127,874, pointer 3 = 151,861  , pointer 4 = 556,323
Block #42 >>> Pointer 1 = 1,597,741, pointer 2 = 899,169  , pointer 3 = 77,240   , pointer 4 = 1,972,274
Block #43 >>> Pointer 1 = 867,939  , pointer 2 = 224,574  , pointer 3 = 1,991,533, pointer 4 = 1,862,433
Block #44 >>> Pointer 1 = 56,087   , pointer 2 = 1,523,931, pointer 3 = 1,775,424, pointer 4 = 813,989
Block #45 >>> Pointer 1 = 1,934,684, pointer 2 = 1,875,333, pointer 3 = 351,389  , pointer 4 = 1,594,081
Block #46 >>> Pointer 1 = 1,659,254, pointer 2 = 1,497,425, pointer 3 = 1,144,537, pointer 4 = 1,909,301
Block #47 >>> Pointer 1 = 181,006  , pointer 2 = 926,403  , pointer 3 = 200,226  , pointer 4 = 220,083
Block #48 >>> Pointer 1 = 1,961,890, pointer 2 = 196,079  , pointer 3 = 1,024,765, pointer 4 = 1,086,911
Block #49 >>> Pointer 1 = 787,733  , pointer 2 = 1,420,293, pointer 3 = 333,228  , pointer 4 = 348,561
Block #50 >>> Pointer 1 = 877,975  , pointer 2 = 1,551,717, pointer 3 = 366,509  , pointer 4 = 583,808
Block #50 >>> Pointer 1 = 123,148  , pointer 2 = 786,913  , pointer 3 = 68,609   , pointer 4 = 1,688,902
Block #51 >>> Pointer 1 = 947,359  , pointer 2 = 2,051,700, pointer 3 = 1,351,502, pointer 4 = 1,043,017
Block #52 >>> Pointer 1 = 1,033,345, pointer 2 = 69,572   , pointer 3 = 295,183  , pointer 4 = 171,513
Block #53 >>> Pointer 1 = 543,245  , pointer 2 = 886,858  , pointer 3 = 658,824  , pointer 4 = 183,334
Block #54 >>> Pointer 1 = 2,094,801, pointer 2 = 1,138,678, pointer 3 = 1,495,391, pointer 4 = 132,119
Block #55 >>> Pointer 1 = 1,903,691, pointer 2 = 744,716  , pointer 3 = 805,725  , pointer 4 = 1,125,685
Block #56 >>> Pointer 1 = 1,739,594, pointer 2 = 670,347  , pointer 3 = 739,898  , pointer 4 = 271,517
Block #57 >>> Pointer 1 = 1,715,325, pointer 2 = 1,931,820, pointer 3 = 818,554  , pointer 4 = 1,287,675
Block #58 >>> Pointer 1 = 69,025   , pointer 2 = 98,573   , pointer 3 = 893,313  , pointer 4 = 563,349
Block #59 >>> Pointer 1 = 502,096  , pointer 2 = 1,075,113, pointer 3 = 610,407  , pointer 4 = 622,094
```

The Effective Key Streams produced by the pointers on the previous page:

```
5F2F325D8D5605567BEF62A02D19EB5AF38777CAAFA09E26A7DE8665C996D87C4237614D99EA6E2C6CD9CEEFAA5759FA2A1FE2B50708FF69
06190A2CA010FA9854E88A6AE0C4CD3F20BAEF154366B8307CF82C028D06F3BFF25F2A80B6851A0E148BFF1FEB20887EB69A974FC4402A57
1284D301A83CEA4A03053214DAB7E3A4E797BD2F8E2405607F069D6583BEBDF9450EE04ED7AA51DD4C1FF1FEF0006A12AA81392D74EF0D73
AB9BD1E13A092D6FFDBB8D92B2DEB3DBF44F37B06F558268D28A24E59410609460246DDBC1A55FC08A396E03024AF617A72F6F953307481D
9C8C00F55899CCB641FB9995533C8BE89E863245448249869C8E46D03BCE4850B782AFF5AD80A495584920D07CE2BC735C9557067518A645
76CE193EB296953ED5E76C24F13E751286D31178F7F78E558C8DBD14B90D07BEB3ED1A6C20A5764528FCE9423B7DC7D12925DC4B04193895
32D9F0023ADC1DB6EB84C5DDE93E8314B369D59ADCB2B572AA551583782FF09E289182F73A1D660DA509E69678152D1F6E941E2B758E11AF
EE3DD4EE7948B3B1CDA7346CA5005D10ED1B3994F6A11A7A3280259FAC853A6F0832DB217894E796BF8BDB165EC68F83F60248FBB0044544
EA5ECBDA57860B4BDFBE762FCF815F222F0E902EE2AE07C261F2811D25F41E8A23EE615EA102E64724848E70B6876A7F6D370FF3412CA4CD
CC8920EC78C9D86D4867417FE33AA9DA448FC4F75A24363659BC36FF5F82AF12E3AE543A44F1F5C62439882303715BF3336C5ED29EE73D53
95A958EF5B3B6C69481F9DBCD60C2FA3BFE8761527CA289470E74361D9E3E677C42CF67293FF02727745BB7C544FA966E6D97885813B9B38
A2F121CCF30ECD3F19DEF5B9F00605E850F83568CD196FE5FA53DD37AC46EF3012CA823AA36858E32FE794D26EBCB596E97012D48FD8DC67
5D0C9EB2FCECDE6CD557FE6BB3AB4106342CC0839E331F3B851D9C30556AE4E4BA01B0BB8AE9915B8D52CFF0A46F876A61039F6600F93162
228984FBB4DBA6C58BEAFEB941E58EA9C163C86AAEEC2EF6A82CC3099BDCA82F0BF76C95F316C1B41797EFCBC2F1DC8335B77D7326442FBB
EF306DCF87CF247A631459F74594D6FC20551E7F8A215CB66CC5AEA68711A10C6C53A3E71829E775BED7CDA62854896D41005DA81F93462E
2E24E64B707255ED0609E803BEC88FC81D6BE3228413EE4D5FE39569857255B2CB552E7FD5531196FDA9F9D5FA16CADD94653C2688D45C83
FF54951BDE953CA377A3ABBBE24BCDB728F55B2931232D665F28A1E797D33FDF0C6CC853C80AAEB7E3442ED28D02170988BD9E22518F56D1
88039E47460D5821292323F6F5BB50C38139AB88CD299FD1F5BA472536D2685157EDCD6A707AFE2CE32EC415EF9A893ADEF02B5CDBCF76F2
489EB35496B8348230F45AF5E922F4E79C5718F0B2C0D1B239411DA1D9C2C7FFB829CA1533ACE0F607C5512ED72AEB53D5F55F65B767BD71
8067969D590A868C7FBB164176C1562F1241D8628CC7377864DB85FEF83A951067ABBCC04C673668BA8AB3E3ED0A0C0E5D1086789A964BEC
0E111FB3489165FD05C7EC450BC483A3F5E267C060BA45E7E983A94A9DE5F8BA7ED24577631B6207D224D4FE7116C93B4831EE9ABBADA55E
A1C0509203617E656159CAA5016BC0AC58A88B5334D5683783197A727D38D4F5574DFBAE5C0B86D7B2949A2E57C84C00519116214525BEB3
0D1A70EF99E1F83553F09119AA72187B0BFF883BA479CFE8A626F5174B28354FA2D56C00ABEF8540837B6E6386FF84193367CF7B2A0058CF
84147F8899E15B02F21A9A2EA4C203DF7E24E9F878CC977856319F903078E94A034D113931E7542407B69211F1BBABC9B3196103BD3B6CF1
9836895CD1C685942CDA9A07685DFF018E1C7047E87C92222AA7BDD60C9B8891EADC90C81D887A07EE32C224E8A8E80CDA3EB1594B919CC5
8CD146E9975EC36A4D446D1D40E6DFDCFA910602589B8F38654C99092AC26ED8EB5930BD57B2B72E544AF5409A40CFCA5B220A0BB05A9658
5DFF40A9C3E5749DDFA12C34E3106E97ED444A8F440D919D60EABA8312E89B1773E02147D262F4D8DA1C015C93ABEF1F97B0D425F8FDD158
2E1044EBD450BD14B254E86697FA4CFF86C9D8531375C21688758775201570C8E3E60E7F6C756AF72358E6D2060FC57A0A3CD03409B17A45
5EE9194F75AE75A2018C2DD6F92790565746D6C2F8E70D31F09C6CCB8073E732885F94795CE36073212CEC068A15B380176F946B25DED1C2
CB2760ECDFEA985A64032B04874108BCB9F3321C4952D500A3002953153027A91DF2E25C2D9B61583C98A3E6C199495021CD80D608D90C06
3A42C0D6DDE1234D4BAE7F76A283A267EF89B0B776B0DA45D2C61E97E548F492E899AEEF45D032803DFF3E61388FBF04CE3B665B71F72E14
309AA35AACC3171DE4022557B27F0B6456D95F9465B05A5DF2B34CB5C6DD21F39A69961D1DA0CE5DF6A2B58E77BDDF183E20181FDCE9F705
9BE7D6376B27A4CCAEDC7434FC162E5794D8300A1995C99FEE2BF043FE6B9D76C4443387F9F2C5A3902AF820391C62CF6226D06B84CDF92B
48E19B24B6B35777738CCF4F48EBBB6FFFF822F11604246AD129B84CC7973A41D8339B105A1E0AFEC16A9CC2A1ED42794C3EE01DA812D763
B2FA9A7378E86999B8B5C0BE2F3F0078AA471D8E78C3F3FA17B71E362E9E96D46D149EE8590CA7E828B44247646D3D62B53E4035CE8FC996
C3494CA7978E09FB6E7F635F820F29B15E565C5BAED519F5BC7472FDB595910D28E818CD34C213036C72F47B9585C2013118AE93F026520D
34086E1B1814E5D09036EEB90BD86E7F09E55C74FB92693D40D9767C87F5B168E5173CEC26259F49D4FC68C5BB06D6F5C6762835E8C2C479
EF7C1DAAEA871F77ACF58793357A69404ECDF792250D8C32DB9A64E941D378C3030C2E24FE3D850053C1BB6E6A78106AC09159680E6E9AC6
FACE14E4BA950BBE39D76562303770DE23E4F10A717498AFFD669D090002E6A82CF4AE20FFDB1EFA089035B4944910F81419E6A61F33B389
8111BB24747C725C67A51DA51D0B17ECC0511BAF6DC936F28B1185FC316F1CAF7F0B51E36F42F72056C76445FADB06B98DB8C51724603D57
4B6BA4676C25C332C0524F573FE76C57D884A26AE5F95ECAFB43F803DF16EDFE9DF90F927F3905C8DEB2C58FE882B98F07A17134AFC87DA3
FC5862106EAEDFED1152D4AD272D5CF7C809B5799B6348F8E59A9571BE222FCC94070F3EC32B6954698644CA2040D17D9FFEC7F2B9C1A66D
B211254F798C68A08FC69AE14A867FE03F24F2F03B7891721074F28DCAAFC4A041764691A1DC8343A969F0006689AE1EC1D9B7B03ECA80FB
7B5060673112EE1F4F7F35C63DDB99408CEA1B24753AF584181937512B820DC765613618C2A40CBD745D3D97405A00BD1045DAFD6ECA3C49
F0586F16D2DF67CD7918BF0F6341443B27A8535A0BB1918FA3F3CF207F9A9303E5FDAEA16CD81F2AB87DA785343396E5924ACA2BF78523CB
8683674A2A77756CCEBBA5035ED95E1898B1145FCB9036165122CDA514FBA2DE47C5157D07784F621617A69DBC2337373E24094D5E191161
EECC555766759E7CBD729959BEF264155D6563021B08BA78A0A148CDD912C171FB352D4CC3B82BD7D42F89979349C7FC93BFFF5E8FBAA271
BE106E865A763C753883C25E16432F26F2EC623F3374F96FD8BB28AB8BFBE79F43D83EB78545327BDC70CBCA9701C3145552073E8E957C72
4D0891DFAC98CBABE790E51CD948362B438ADD50CA0164E0F2EB30F0EC57F132DB5607C961FB9119CB5A8CE1B9E745B6BBF60F8081D463E2
41C506A0619DF5E685B8F33B2BCE7683C351190B0B458F90EAE2C896D9E033FDC45425D828B1DA5C531BFDFE413954D23D1A1A22925D93AD
04A1387A8812B779C37628FDD2F38B304545B3F6A66B71DB0159B1F5413841923B53554897E714A19396D659030C033A4A01D1AA75264870
67C2B889EC2F3D5C0689A2F1F9C56510DAF54AC4C15AAA796DE6FB05EB9963DE6BEA254F3612055E7B06ADD0A0681771C0A9B61DB09D716B
1A45E2E7FE87E7D6485B66CFED5C466B767C15E18111130C5AE7BAAB016E4011AE10D87B3B73131626AFDEE6A136251A025199E44D9C23BD
316BE4E00E72B018EA72FC4D13EDFDB6D143CF45926CCBE5E3AEA8DF1B3E0BBF3DB1D9599253C2AA41C89472E6056D23B9E83DE607229B8A
3161977E7902D3AD54E251D7E9EC518F1F63EB84ED4D6DE48777708BB21EE686D4E295B6CDF7E0B510A2B20E75696D39A1ACA852EE80782E
0CACC7C6B12B0D26BD569F5C63E916DBD31EF7D6DF2708EEFDF796622FC59F084461629D2B62CFF4D0FD12BD8D712CA764FFF015D92E2924
99AE773753EA98A19BC03DA7473D2040B3A0197434DE18962073E4777EF151A8B05AAB956B48D311CFEB7BF4AB1635FD426A10B12148464F
9DE56C48F311E728961AB1D03E319C25E9CF1ADCAFDE4A4C6EC01F913A8C0A6793692FCEA5B072ED26D42FFE57BE588AF6272F2347B48866
F7711314DB3589D87554417888ADFA8F55958872E144A2AA599FC5753C0B043C1BBB6A8110172FAE4073A74C0E6F9464FBE1EFCA413FF173
49E12DCE1D7C1E33A213A896C1B93267C67BF0D44BDA0781666E3EFB072E12604BEDA9A6A4EDF0B11A6346351F49CDE946CA8540F4490030
```

A test was run to just produce 2 billion (2,000,000,000) sets of pointers and test to see what the pointer values would be as well as see when all pointer values from 0 to 2,097,151 inclusive would be used (to test the thoroughness of the engine pointer advancement pseudo-algorithm). The following is a sample of what it found. All log files are available for examination if needed – contact the author. Each log file contained over 500 test points where the low 22 bits of the ciphertext block number were all equal to 1 to start the 10-record set, 6 of them are shown below. This will show the unlikelyhood of any two sets of pointers being equal since one of the factors that is incorporated in calculating pointer 4's value is the lower 22 bits of the block number.

```
Block #1 >          P1=[7,902,840], P2=[5,346,543], P3=[4,032,778], P4=[1,346,022], E2P=22,   E2O=109
Block #2 >          P1=[5,377,781], P2=[3,535,628], P3=[849,394  ], P4=[6,822,413], E2P=1267, E2O=159
Block #3 >          P1=[1,287,325], P2=[6,124,964], P3=[6,594,675], P4=[2,420,499], E2P=2950, E2O=156
Block #4 >          P1=[3,094,946], P2=[6,484,536], P3=[1,614,575], P4=[5,411,392], E2P=7044, E2O=289
Block #5 >          P1=[1,897,507], P2=[4,456,180], P3=[3,417,340], P4=[6,349,308], E2P=2119, E2O=68
Block #6 >          P1=[5,265,372], P2=[1,880,149], P3=[7,724,976], P4=[2,927,596], E2P=6186, E2O=69
Block #7 >          P1=[3,466,978], P2=[4,372,197], P3=[385,716  ], P4=[8,047,812], E2P=2507, E2O=300
Block #8 >          P1=[180,510  ], P2=[4,105,153], P3=[6,365,602], P4=[5,574,323], E2P=5523, E2O=68
Block #9 >          P1=[2,896,340], P2=[1,362,992], P3=[5,297,868], P4=[7,565,998], E2P=781,  E2O=89
Block #10 >         P1=[1,981,584], P2=[7,389,500], P3=[3,969,470], P4=[4,262,141], E2P=3043, E2O=254

Block #4,194,313 >  P1=[7,462,363], P2=[5,772,250], P3=[3,856,658], P4=[51,823   ], E2P=3437, E2O=66,  correct = 0
Block #4,194,314 >  P1=[5,451,658], P2=[2,765,873], P3=[887,348  ], P4=[6,600,625], E2P=6477, E2O=141, correct = 0
Block #4,194,315 >  P1=[426,089  ], P2=[4,466,818], P3=[6,317,862], P4=[2,734,529], E2P=1232, E2O=264, correct = 0
Block #4,194,316 >  P1=[2,964,703], P2=[8,351,803], P3=[1,826,672], P4=[5,339,994], E2P=5053, E2O=248, correct = 0
Block #4,194,317 >  P1=[263,623  ], P2=[4,523,525], P3=[2,475,528], P4=[8,279,627], E2P=369,  E2O=130, correct = 0
Block #4,194,318 >  P1=[5,074,229], P2=[1,297,773], P3=[7,026,383], P4=[2,595,607], E2P=4939, E2O=243, correct = 0
Block #4,194,319 >  P1=[4,124,446], P2=[6,103,279], P3=[924,961  ], P4=[7,573,609], E2P=3728, E2O=180, correct = 0
Block #4,194,320 >  P1=[1,011,011], P2=[2,322,541], P3=[7,161,456], P4=[4,984,249], E2P=3517, E2O=244, correct = 0
Block #4,194,321 >  P1=[3,099,336], P2=[569,161  ], P3=[4,836,015], P4=[7,271,163], E2P=1252, E2O=98,  correct = 0
Block #4,194,322 >  P1=[1,679,353], P2=[7,822,496], P3=[4,192,349], P4=[4,572,886], E2P=6141, E2O=191, correct = 0

Block #6,291,457 >  P1=[8,385,271], P2=[5,620,207], P3=[3,143,361], P4=[1,067,877], E2P=4333, E2O=318, correct = 0
Block #6,291,458 >  P1=[4,297,169], P2=[3,031,696], P3=[1,035,845], P4=[7,038,859], E2P=2024, E2O=273, correct = 0
Block #6,291,459 >  P1=[780,341  ], P2=[5,524,970], P3=[6,829,903], P4=[2,373,070], E2P=5833, E2O=195, correct = 0
Block #6,291,460 >  P1=[2,994,434], P2=[6,467,760], P3=[1,049,261], P4=[5,276,273], E2P=171,  E2O=64,  correct = 0
Block #6,291,461 >  P1=[1,769,717], P2=[5,594,370], P3=[2,160,220], P4=[7,838,709], E2P=490,  E2O=280, correct = 0
Block #6,291,462 >  P1=[5,511,209], P2=[521,238  ], P3=[7,744,244], P4=[2,133,508], E2P=3889, E2O=84,  correct = 0
Block #6,291,463 >  P1=[2,880,534], P2=[5,590,516], P3=[1,250,635], P4=[7,398,359], E2P=7365, E2O=215, correct = 0
Block #6,291,464 >  P1=[244,500  ], P2=[3,466,427], P3=[8,067,043], P4=[5,838,822], E2P=2503, E2O=142, correct = 0
Block #6,291,465 >  P1=[3,648,941], P2=[645,040  ], P3=[5,024,727], P4=[6,451,635], E2P=2223, E2O=299, correct = 0
Block #6,291,466 >  P1=[2,055,661], P2=[7,119,455], P3=[4,058,527], P4=[4,641,129], E2P=6951, E2O=309, correct = 0

Block #10,485,769 > P1=[7,309,432], P2=[5,771,653], P3=[2,455,823], P4=[253,478  ], E2P=951,  E2O=65,  correct = 0
Block #10,485,770 > P1=[4,902,934], P2=[3,599,312], P3=[923,371  ], P4=[7,547,698], E2P=2872, E2O=150, correct = 0
Block #10,485,771 > P1=[229,398  ], P2=[5,580,164], P3=[6,297,640], P4=[3,214,079], E2P=1354, E2O=76,  correct = 0
Block #10,485,772 > P1=[3,889,170], P2=[7,028,313], P3=[1,510,204], P4=[4,198,870], E2P=1899, E2O=290, correct = 0
Block #10,485,773 > P1=[655,303  ], P2=[4,426,753], P3=[4,179,083], P4=[7,352,750], E2P=2047, E2O=190, correct = 0
Block #10,485,774 > P1=[5,660,508], P2=[1,873,501], P3=[8,216,470], P4=[4,087,636], E2P=3758, E2O=238, correct = 0
Block #10,485,775 > P1=[2,968,402], P2=[5,230,419], P3=[675,789  ], P4=[7,681,370], E2P=1321, E2O=106, correct = 0
Block #10,485,776 > P1=[26,128   ], P2=[3,195,238], P3=[6,689,728], P4=[5,000,089], E2P=6211, E2O=214, correct = 0
Block #10,485,777 > P1=[3,540,719], P2=[1,013,260], P3=[4,583,798], P4=[6,823,134], E2P=189,  E2O=300, correct = 0
Block #10,485,778 > P1=[2,041,101], P2=[7,201,317], P3=[2,428,639], P4=[5,318,728], E2P=7973, E2O=198, correct = 0

Block #12,582,913 > P1=[7,981,227], P2=[4,946,885], P3=[2,468,985], P4=[1,525,344], E2P=3559, E2O=245, correct = 0
Block #12,582,914 > P1=[4,928,206], P2=[3,001,392], P3=[1,101,259], P4=[7,522,773], E2P=7449, E2O=166, correct = 0
Block #12,582,915 > P1=[1,447,935], P2=[6,068,252], P3=[7,864,214], P4=[3,723,628], E2P=767,  E2O=319, correct = 0
Block #12,582,916 > P1=[3,438,051], P2=[6,362,996], P3=[1,368,340], P4=[5,411,350], E2P=2498, E2O=172, correct = 0
Block #12,582,917 > P1=[1,153,822], P2=[6,189,979], P3=[3,874,675], P4=[7,405,232], E2P=3288, E2O=280, correct = 0
Block #12,582,918 > P1=[5,611,423], P2=[2,037,149], P3=[8,233,493], P4=[3,941,331], E2P=1895, E2O=300, correct = 0
Block #12,582,919 > P1=[3,852,574], P2=[6,151,368], P3=[531,930  ], P4=[6,585,572], E2P=119,  E2O=164, correct = 0
Block #12,582,920 > P1=[577,650  ], P2=[2,714,068], P3=[7,367,785], P4=[6,032,914], E2P=833,  E2O=134, correct = 0
Block #12,582,921 > P1=[3,054,716], P2=[1,855,134], P3=[5,996,112], P4=[6,313,261], E2P=1759, E2O=287, correct = 0
Block #12,582,922 > P1=[95,093   ], P2=[7,603,315], P3=[3,372,293], P4=[5,804,184], E2P=1485, E2O=179, correct = 0

Block #16,777,225 > P1=[6,870,105], P2=[5,884,625], P3=[3,234,505], P4=[1,489,925], E2P=2257, E2O=85,  correct = 0
Block #16,777,226 > P1=[4,905,823], P2=[4,139,993], P3=[1,662,763], P4=[6,747,549], E2P=2917, E2O=235, correct = 0
Block #16,777,227 > P1=[1,532,481], P2=[4,274,534], P3=[6,439,992], P4=[2,177,847], E2P=3780, E2O=82,  correct = 0
Block #16,777,228 > P1=[3,945,832], P2=[6,831,925], P3=[1,337,404], P4=[5,353,386], E2P=2513, E2O=198, correct = 0
Block #16,777,229 > P1=[278,300  ], P2=[5,973,567], P3=[4,135,973], P4=[7,985,854], E2P=2019, E2O=136, correct = 0
Block #16,777,230 > P1=[5,871,470], P2=[776,597  ], P3=[7,696,089], P4=[4,080,056], E2P=6903, E2O=106, correct = 0
Block #16,777,231 > P1=[2,539,253], P2=[5,613,757], P3=[1,963,432], P4=[7,908,093], E2P=1725, E2O=232, correct = 0
Block #16,777,232 > P1=[1,753,810], P2=[3,070,914], P3=[6,476,251], P4=[6,125,204], E2P=3803, E2O=75,  correct = 0
Block #16,777,233 > P1=[2,414,991], P2=[861,402  ], P3=[5,803,812], P4=[6,758,921], E2P=1129, E2O=100, correct = 0
Block #16,777,234 > P1=[1,480,138], P2=[6,887,829], P3=[3,525,142], P4=[4,440,228], E2P=3646, E2O=126, correct = 0
```

Solution to the ‘assignment’ on page 6

```
Plaintext ASCII Hex  - 54 68 69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78 74 20 74 6F 20 64 65 6D 6F 6E 73 74
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Key Stream @6,126,456- 77 C9 94 C2 7E EA 4B 02 C5 97 38 25 8F F0 1E 0C 0F 1D E5 6D 00 EA DE 9E 35 17 B3 10 12 82
Key Stream @3,858,927- 2C 12 C4 97 4E FE AE D1 6A 01 53 67 7D 88 4C D1 2D 78 0E 93 09 6A 3A 32 46 32 52 17 3F 0C
Key Stream @1,678,858- 59 0E 9A 4E 20 A3 76 99 61 2F C0 95 E7 B0 4F A9 81 59 AA C0 C5 18 8A 34 6D 28 63 65 15 88
Key Stream @7,605,059- 91 DB DC 17 29 2A 4C 2B 79 28 4D 07 9F 71 44 31 D4 B9 68 7E 2D C2 24 D3 A2 EA AF FD 95 8D
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C7 66 7F 7F 19 F4 AC 41 C4 F0 8B A0 E6 DC 79 31 12 FD 5D 60 95 35 6A 2F D9 8A 42 F1 DE FF

72 61 74 65 20 74 68 65 73 65 20 73 74 65 70 73 20 6F 66 20 74 68 65 20 27 52 61 6E 64 6F 6D 20 43 69 70 68 65
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
75 69 92 B4 4C 8A D3 69 32 EE 20 EB CB 2E 0E AA F4 4C 4C 7E 8D F6 02 99 F9 EB D0 C5 A4 9F 26 DD 47 83 A3 C4 03
0C DD C6 6F 4E 19 B8 56 AC 9D F9 91 47 BB BB 57 4A 1C AA D7 53 08 F5 E2 A4 A2 12 F6 FD 20 61 B0 66 1F 7D 77 6D
9D A7 14 54 96 16 09 CC BA 15 44 89 CA A8 6C 09 41 D3 F2 61 F0 99 CA B9 10 91 31 23 A8 08 09 5B 72 4A FC 29 99
47 7D 74 7A 39 59 AE 3B 03 7B 1A 34 C9 2A 94 F7 3F E2 5C 00 9E 71 89 F2 A2 8D AA 5E AA 4D 35 B2 68 75 9B B8 77
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
D1 0F 40 90 8D A8 A4 AD 54 78 A7 B4 FB 72 3D 70 E0 0E 2E E8 C4 7E D1 10 C8 07 38 20 3F 95 16 A4 78 CA C9 4A E5

72 20 4F 75 74 70 75 74 73 27 20 6D 6F 64 65 20 75 73 69 6E 67 20 55 4E 41 54 54 41 43 4B 41 42 4C 45 20 41 6C
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
B7 22 F0 80 CC BA 84 9A 20 47 F9 A1 D7 23 00 74 D0 64 09 CF 80 BB 2D F9 BB 0A D5 E5 12 58 F8 3E D8 CF 31 DF DD
E7 2A 75 A5 AF 9B B1 D0 AE C7 39 B4 F6 A9 7F 88 57 CD 78 BB 8D 51 02 41 58 CD 4A 69 40 8D 8E 12 0B AA BC DF 6E
D6 C2 81 A6 87 EC 82 24 64 BE 69 55 1A 81 A4 01 CC 8F E0 25 8A 4D 88 30 3A 8E D7 56 FB D1 3F 3C 7F 9B C3 C5 85
A7 01 F4 BA 44 5D 8B 11 3D F6 F7 31 19 70 D7 B3 BE 03 16 ED 34 92 F3 44 7D CA 78 95 E8 D4 D7 3D 31 4B E4 49 74
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
53 EB BF 4C D4 E0 49 0B A4 EF 7E 1C 4D 1F 69 6E 80 56 EE D2 D4 15 01 82 E5 D7 64 0E 02 9B DF 6F D1 F0 8A CD 2E

67 65 62 72 61 69 63 20 6C 61 77 20 66 6F 72 20 73 65 63 75 72 69 74 79
|| || || || || || || || || || || || || || || || || || || || || || || ||
D7 A9 30 4A 8C 28 EC 40 E5 BF 8F D8 55 1A C0 D1 65 18 5F 7B F6 5F 7B 59
DE 77 AD 7B AF F2 DC 7D 44 DF 31 4F 43 BD 64 B0 67 A4 B2 42 5B 54 9D 5B
04 E6 B5 08 66 F1 15 12 AB CC 67 9D 00 E2 46 B3 1A 88 CF 7D 74 1B 96 05
D1 BB 90 94 67 3B D8 60 CD EF 58 AD FC 34 17 66 1C 9C 47 06 F0 8C EE CD
|| || || || || || || || || || || || || || || || || || || || || || || ||
BB E6 DA DF 43 79 9E 6F AB 22 F6 87 8C 1E 87 94 77 CD 06 37 5B F5 EA B3
```

Solution to the ‘assignment’ on page 12

Answer to the question, ‘Internal Data Stream’ digit 226 is the first ‘B’ in the ciphertext line.

```
                                    1 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55 55 56
                       12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
Internal Data Stream - C7 66 7F 7F 19 F4 AC 41 C4 F0 8B A0 E6 DC 79 31 12 FD 5D 60 95 35 6A 2F D9 8A 42 F1 DE FF

  Transposition      - 2  11  2 11 22 11  1 11 11 21  1  2 1  11 11  2 22  2 2  21 1  21    11  1 12 1  12 1  21
 Key Number 52       - 22 75 73 28 30 11 60 24 17 14 53 83 16 40 80 33 40 95 04 44 19 56 94 95 80 34 29 72 17 37
 Offset Number 211   - 67 20 69 22 46 19 38 45 45 32 16 28 71 61 19 70 08 20 36 71 21 48 77 65 92 22 84 63 35 38
                       || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
 Ciphertext Output   - BD EB D9 62 CD 09 00 4E 8D DC 83 77 3D 0C 8C 56 4E 0B 8A 34 77 A0 22 F7 34 A7 A0 46 3A 85


                                                       1 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00 01 11 11 11 11 12 22 22 22 22 23 33 33
12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
D1 0F 40 90 8D A8 A4 AD 54 78 A7 B4 FB 72 3D 70 E0 0E 2E E8 C4 7E D1 10 C8 07 38 20 3F 95 16 A4 78 CA C9 4A E5

22  1 12  1    1  1      1  2  1 1  11 1  1  22    1     11 1  21  2 11       12          12     2  1 11 2  12
01 92 94 54 7  15 64 54 23 54 98 74 05 46 45 40 16 33 84 98 11 40  1 48 13 16 20 13 13  6 61 57 30 85 94 3  53
75 69 35 44 26 60 65 79 27 23 97 34 62 72 85 15 04 98 51 26 09 43 98 30 55 19 54 36 22 36 90 34 09 31 59 12 46
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
2D EC 90 24 8F 09 E6 DD BE AC E6 D5 14 41 9F 7C 9F BD F9 27 8F D7 13 D1 4F F8 7A AD 41 60 5B 44 9B BA D0 87 FE


11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 22 22 22 22
33 33 34 44 44 44 44 45 55 55 55 55 56 66 66 66 66 67 77 77 77 77 78 88 88 88 88 89 99 99 99 99 90 00 00 00 00
56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
53 EB BF 4C D4 E0 49 0B A4 EF 7E 1C 4D 1F 69 6E 80 56 EE D2 D4 15 01 82 E5 D7 64 0E 02 9B DF 6F D1 F0 8A CD 2E

12 12 2  1   1  1    1  12 21  1  2 11 22 1   1     1    12  1  1 22 11 1  11 22 22 21    22 1  2  12 11 22 1
92 61 47 7  23 18 47 57 55 20 61 91 28 53 55 36 71 90 89 71 12 41 04 73 31 38 15 20 28 83 21 6  12 92 39 53 62
82 12 61 17 84 73 37 78 63 55 78 04 04 17 88 32 94 54 48 71 83 25 08 95 86 09 76 12 95 79 06 05 90 18 17 52 59
|| || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || || ||
FE 16 6A E7 C5 CE 35 14 EE AD 9F DA 55 F8 CE 1F 7C 0E 4E 1E 4A 52 17 05 B1 90 43 90 FD 76 9F D7 70 02 46 B7 67


22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
01 11 11 11 11 12 22 22 22 22 23 33 33 33 33 34 44 44 44 44 45 55 55 55
90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56 78 90 12 34 56
|| || || || || || || || || || || || || || || || || || || || || || || ||
BB E6 DA DF 43 79 9E 6F AB 22 F6 87 8C 1E 87 94 77 CD 06 37 5B F5 EA B3

11  1 1  22 1  21  1  1  1 1  11 1   2    21  1  2     1   1        1
69 64 66 54 98 02 50 58 29 5  62 23 72 24 27 25 83 46 38  0 39 2  72 87
34 00 45 29 06 16 60 98 69 98 71 74 37 48 40 33 05 08 31 17 13 14 45 80
|| || || || || || || || || || || || || || || || || || || || || || || ||
6B FF 94 55 EB F8 18 F4 6D 4F 81 C2 A2 0F F6 AE 81 00 EA C1 3E 86 2E 2D
```