Analysis and Details of the Random Cipher Output Mode Of Operation Primitives

Author: Dan P. Milleville

48 Groton School Road, Ayer, MA 01432-1000, U.S.A.
dmilleville@verizon.net

Abstract. Consider that Hardware and Software attack Technologies seem to be advancing at an exponential pace. Should it be acceptable to believe that all of the current Modes Of Operation (MOO) will still be 100% safe from technology attacks 5 to 30 years or more into the future? Predictions about DES's security when it was first developed proved to be wrong; with the volume of information and data being protected by current MOO's, the security industry cannot afford to be wrong again. This is not to say that just because the experts were wrong about the DES that they are wrong now. They have never had and do not have perfect vision into the future about what will develop in the security attacking technology arena. Suppose some 'brainiac' teenager devises a sophisticated attack technology that no one thought of and one or more of the accepted MOO's are broken; then we will all be racing to recover. With these potential advances in hardware and attack technology could come the time when none of the currently accepted modes of operation are safe from an attack. We ought to consider not designing ciphers that are even more complex, as this will just escalate the 'leap-frog' race between cipher developers and attackers. The problem isn't the complexity; the mathematical connection between the plaintext/ciphertext pair and the connection to only one key needs to be expanded to multiple key connections. This MOO is presented as one potential solution to be considered to combat this potential problem by attempting a solution along this path. This proposal does not involve any new cipher engine technology.

Key words. Pseudo random number generator, Non-Deterministic random number generator, Vernam, AES, Engine orientation, Overhead data placement, Checksum, XOR.

1. Introduction

This MOO design has both a Pre and Post-Processor, unlike all the current MOO's that do not have both. In this design, they are there to prevent any predictable access to either side of the central (AES) engine, as well as allow the flexibilities specified throughout this document.

There is a fundamental problem with using only one key for the main or central engine in an MOO. If a brute force attack were ever found to be successful in a reasonable timeframe for an MOO, the single key could then be used to obtain all the data protected by that key. If the key is forced to change between blocks, they would need to start the attack process all over again for each successive block.

Suppose the number of keys that would be available to this engine was expanded. Suppose it was randomly determined which way the engine was facing for each block, forcing the attacker to fail should their process assume the wrong orientation. Suppose the key used for a block was randomly selected and forced to be different from the preceding block. This forces an attacker to simultaneously process multiple blocks of text in order to carry out any attack. Key reuse in non-adjacent blocks is possible and would result in attack failure to incorrectly assume that any two non-adjacent blocks were or were not encrypted with different keys. This would force the attack algorithm to deal with the additional problem of determining what other block, if any, was encrypted with the key used for the current block under attack before the key is reconstructed.

Another fundamental problem is in attempting to design a cipher engine or MOO so that it is extremely difficult to recreate a correct key. Suppose an MOO was designed so that it was easy to create more keys than what could be stored by all the computer disks in the world for just one plaintext/ciphertext pair. The attack process would then become drastically compounded. The attack effort would switch from producing a key that decrypts successfully to attempting to decide which key would be 'correct'. Once you are familiar with this design, this is discussed in more detail in the Conclusion at the end of this document.

Considering current technology in Flash (non-volatile) memory, more than one key being available for a particular MOO to encrypt a block should now be considered and allowed. When a 4 Gigabyte Flash device can now fit on a person's thumbnail, it could easily hold the cipher image and multiple key files, each containing multiple keys. It could hold well over 100 years of monthly key changes. It could be placed within a CPU Processor Chip and designed to be accessible only to the processor. Security of the cipher image and key so stored could be assured to the user if the Chip was developed with proper physical protection technology (steel and/or lead with a lithium ion battery to power destruction if needed).

This MOO also does not require the delivery of any seed number to the legitimate recipient. Since it is no longer required, increased security can be assured since even if the attacker had the same PRN code, they would have no seed to begin their attack. By including the randomly selected data within the payload, this will be the only MOO that can use a Non-Deterministic Random Number Generator during the encryption function whenever one is approved for use in cryptographic systems.

Since there is no seed transmitted, the random numbers selected have to be included in the payload in order for the legitimate receiver to decrypt the text properly. At least some past attempts at including numbers along with the payload have resulted in these numbers being loaded into fixed locations where mathematical processes were then able to discover their location. The numbers inserted by this MOO design are not only placed in randomly defined locations that vary between blocks and files, they are also encrypted along with the payload. The key used to encrypt these numbers is, even to the legitimate receiver’s system, unknown. No character position in any ciphertext block can be excluded from possibly holding what will eventually become an overhead digit. For the first block only, the legitimate receiving system randomly picks various Post-Processor keys, decrypts the ciphertext and performs expected checksum and data checks on the decrypted text. These are detailed on pages 74 and 75 of this document during the decryption of the test text. If any of the data tests fail, another key is tried and the decryption and test process is repeated. When all the tests finally pass, the then-decrypted numbers are extracted from their random locations for use in completing the decryption process.

This MOO also brings into reality the ability to construct massive numbers of keys with any given plaintext/ciphertext pair. With other MOO’s, if an attack system were to ever successfully produce a key, that key would be the only one that could perform the decryption; that is definitely not true with this design. This is demonstrated in comparing Example 1a (pages 18 to 21) with Examples 2 (pages 26 to 29), 3 (pages 30 to 33) and 4 (pages 34 to 37). For each example, there are 25 additional same-case examples produced by the demonstration system that are available upon request. Disk space available in the world does not allow all possibilities to be produced for just one block as you will observe later, not to mention the time needed to produce them. Any input data stream can be easily changed into any output data stream with a Vernam Engine that uses 3 key segments – even data streams that are identical as in Examples 9b #1 and #2.

Take the case where the input and output data streams are identical. If there is one segment to Xor with the Vernam input, as with the original design, then there is only one value (0) that is available for all key locations; with all ‘0’s in the key, this would obviously be an unacceptably ‘random’ key. If there are two segments, then there are 256 pairs of values that will Xor to ‘0’; with a limited number of selections, some of the possible segments could appear to be random, others would not; but both segments would be equal, definitely not acceptably random segments. With 3 segments to Xor between the input and output character data, there are a total of 65,536 different possible sets of 3 values that will all Xor to zero; with only 111 sets needed per operation, all segments can be created randomly and different, as will be shown in Example 9b #1 on page 66 with 25 other samples available.

The segments that the Pre and Post-Processor use during the encryption or decryption process are obtained from a large single-dimensioned key structure and Xor'ed with the input to produce the desired output. So considering the variety of output text streams that can emanate from the AES engines by varying the keys used as well as the engine orientations, the 3 segments of keys can be created in a random way that can produce the steady-state output ciphertext just by varying the random makeup of the 3 segments.

Attack technologies rely on changes to the ciphertext when a change is made to the plaintext or key. Suppose there is a change in the keys but no change occurs in the plaintext/ciphertext pair? The attack cannot work no matter what technology is employed because <u>there is no mathematical information available since there is no change to calculate any specific key reconstruction</u>.

The MOO technology demonstration system will be producing all the data displays to provide evidence of this ability to mathematically hide the content of any key. The format of the output of the demonstration system has been changed slightly to accommodate the space requirements of this document; but the original files are available for inspection.

Why make it easy for attackers?

The entire intent of making it actually easy to create a working key is to give attackers a false sense of security that they are going down the correct path. It isn't until they have gone through all the effort of (what they might believe is) successfully creating a significant number of keys for many blocks that they realize they have run out of room to store all the different keys they find they still need to create; they then realize they have failed. Take a 200-block ciphertext file and look at just one key of 1,024 used in creating the first block. There are 199 chances out of 1,024, or about 1 in 5, that this first key is used somewhere else in this file. So just within the first 20 blocks, chances are very high that one of the keys used in those first 20 blocks is used elsewhere in a non-adjacent block. If the attacker has not reused any of the keys within those 200 blocks, they can very reasonably know that something in their process has failed to create the correct key in spite of their apparent success. If this is the case, then they have to try and duplicate at least one key used in one block somewhere in some other non-adjacent block; but which blocks use the same key? What criteria could be used to flag a key's reuse without knowing the numerical makeup of the key? Is the key used to encrypt the current block <u>not</u> reused at all? That is a possibility with only 200 blocks. And if it is reused, how many times and what other block or blocks were encrypted using that key? One point they will need to realize is that with this size of ciphertext (200 blocks), they would not be able to reconstruct the entire key that contains 1,024 keys. Testing that I have performed dictates that they would need at least 7,000 blocks to have a good chance for all keys to have been used, 15,000 for a virtual guarantee. With all other MOO's these are not obstacles that an attacker has to face. The more obstacles they are forced to face, the greater the difficulty for an attack, if it is attempted.

2. Preliminaries

Currently, no MOO has the ability to vary the key in the central engine and have the Pre or Post-Processor capable of holding the ciphertext to a ‘no change’ state in more than one block. In other words, take a given plaintext input and ciphertext output. Vary the key to <u>any</u> key that would <u>ever</u> be provided to the AES engine. In less than 4 seconds, the Vernam’s keys can be re-randomized within to cause the ciphertext to not change from the initial ciphertext sample. If this were done easily and frequently, there cannot be any mathematical information that could be contained within the plaintext/ciphertext pair since there is no change to analyze. Any of the keys that could be randomly produced could have been used to construct the ciphertext.

Also currently, no MOO has the ability to vary the plaintext input, vary the keys and still hold the ciphertext to the same state as the above sample in less than 4 seconds. Ease of key reproduction introduces a new roadblock, one that should stop all attacks on this design, both now and at any time in the future, regardless of any technological advances.

Take a block of randomly determined characters, not knowingly from any cipher engine, equal in length to the block size of a given cipher, and create another seemingly endless set of keys capable of taking the plaintext sample to this randomly produced block in less than 4 seconds. This, too, is virtually not possible with any of the current MOO’s, but is possible with this design. Vary the plaintext input with the false ciphertext output block and again produce numerous sets of keys that will take the changed plaintext input to the false ciphertext output in less than 4 seconds.

As will be observed, all this is made possible by well known cipher engines, AES and Vernam; no new technology is involved in performing the translation of the plaintext to ciphertext and back. The methodology surrounding the known cipher engines that takes no part in the translation of the input or output texts makes all this possible.

There are some questions that need to be answered in reviewing this document. Does it continue to be the goal to make cipher engines so that it is an insurmountable effort to reengineer the one and only key that would successfully perform the decryption? Should the attack effort be transferred away from reengineering a key that gives attackers a challenge? Should it be to attempt to determine which one of a truly exhaustive number of keys is correct that were produced from just one ciphertext block? Would this force even advanced <u>as-yet unknown</u> hardware and software technologies to be totally incapable of yielding the correct key?

3. MOO Design

The design is illustrated here in Figure 1, with the cipher engines that execute all the text translations indicated by shaded boxes:

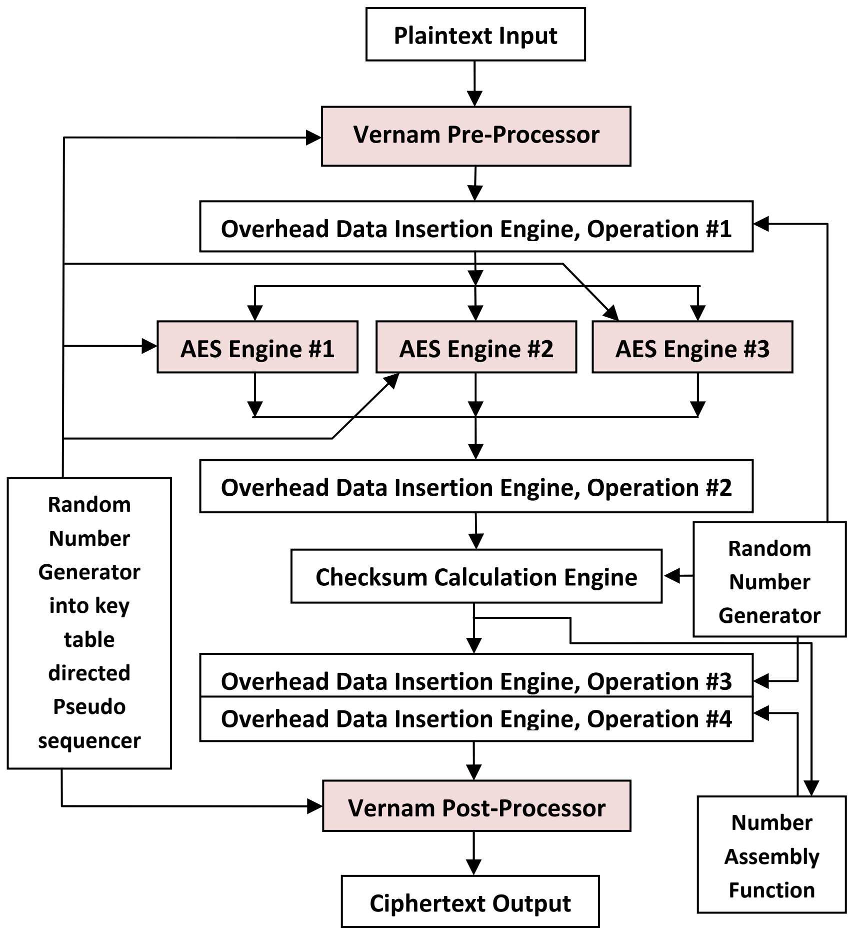

Plaintext Input
Vernam Pre-Processor
Overhead Data Insertion Engine, Operation #1
AES Engine #1
AES Engine #2
AES Engine #3
Overhead Data Insertion Engine, Operation #2
Random Number Generator into key table directed Pseudo sequencer
Checksum Calculation Engine
Random Number Generator
Overhead Data Insertion Engine, Operation #3
Overhead Data Insertion Engine, Operation #4
Vernam Post-Processor
Number Assembly Function
Ciphertext Output

Figure 1.

The design consists of a special Vernam Pre-Processor, three parallel and reversible AES engines and another special Vernam Post-Processor as the cipher engines. Overhead Data Insertion and Checksum Engines are placed at designated points.

Both of the Vernam Processors access a fixed single-dimensioned key structure of 113,886 randomly set bytes. The Vernam engines are of a special design in that they encrypt a single block of text utilizing three – 94-byte segments for the Pre-Processor and three – 111-byte segments for the Post-Processor, all obtained from this main 113,886 byte key. These three segments are selected randomly and at the direction of a pseudo-selection process, details follow. The values from the three segments are then Xor’ed with the input data to form the output for the AES engines or ciphertext.

4. Comparisons, Similarities and Differences

Comparing some factors of this MOO's design to the other accepted MOO's:

Item	ECB	CBC	CFB	OFB	CTR	RCO
1	No	Pre	Post	Post	Post	**Both**
2	N/A	Xor	Xor	Xor	Xor	**Xor**
3	Yes	No	No	No	Yes	**Yes**
4	Yes	???	???	???	???	**No**
5	N/A	Yes	Yes	Yes	Yes	**No**
6	No	No	No	No	No	**Yes**
7	No	No	No	No	No	**Yes**

1. MOO contains a Pre or Post Processor during Encryption.
2. The Math operation used in the processor(s).
3. During Decryption, the process can recover after receiving a corrupted block.
4. If a brute force attack of the center cipher engine is successfully developed, the MOO is considered broken.
5. Pre and/or Post processor behavior is predictable.
6. For multiple blocks, the MOO has the ability to hold the ciphertext to no change for any center engine key change.
7. With no plaintext changes, the ciphertext varies with repeated encryptions.

If an attack should ever succeed on at least one MOO, an acceptance gap will occur if an already accepted new MOO that would successfully combat the new attack technology is not already in place and in common use. If the successful attack occurs, there will also be all the work of converting the then-currently encrypted texts to the newly accepted design. This is not to mention all the costly work attempting to recover from any known or potential breach of security due to the failure. Recovery will not be possible in some cases.

Is the security community willing to risk this sort of scenario, pressing all their hopes on the technology that exists today will still be solid tomorrow when it cannot be absolutely guaranteed?

Dr. Burt Kaliski of RSA Data Security, states in a speech entitled 'Long-Term Security Assurance and Quantum Key Distribution', located at website:

http://www.rsa.com/blog/blog_entry.aspx?id=1093

The second paragraph states:

> "A well-designed system will have algorithm agility, so that algorithms can be upgraded routinely. If an attack on an algorithm should be discovered at some point in the future, then presumably by that time the algorithm would not have been in use for many years, and any data or keys previously protected by it will no longer be of value."

Current MOO's are not in a state of transition out of use in spite of the possibility that there might be a successful attack launched on any one of them at some point in the future. If any of the current MOO's can claim that there will never be a successful attack against it regardless of any technological advances in attack hardware and/or software, then that MOO is adequately protected. Otherwise, steps need to be taken now to insure that any MOO that might be susceptible to possible attack in the future is gradually phased out of use before the time that a successful attack arrives.

This MOO has the 'algorithm agility' mentioned in the quote above. All that would be required would be to change the block size processed by the Pre and Post-Processors.

5. High Level Sequence - Encryption

The sequence for encryption is as follows:

1. Select the Post-Processor seed, range 0 to 1,023 and Post-Processor Offset seed, range 0 to 110.
2. Select 8 random numbers for use during encryption of this block:
    a. Pre-Processor seed, range 0 - 1,023
    b. Pre-Processor offset seed, range 0 - 93
    c. AES Engine seed, range 0 - 1,023
    d. AES Operational Orientation mask, range 0 - 7
    e. Overhead Placement Engine key, range 0 - 1,023
    f. Checksum Engine key, range 0 - 1,023
    g. Next block Post-Processor seed, range 0 - 1,023
    h. Next block Post-Processor offset seed, range 0 - 111
3. Execute Pseudo algorithm for selecting Pre-Processor segment key access numbers
4. Execute Vernam Pre-Processor, Xor'ing values from 3 segments selected with the ASCII of the plaintext.
5. Operational Placement Engine inserts 4 hex digits containing next block's Post-Processor and offset seeds (g. and h.).
6. Execute Pseudo algorithm for selecting AES engine key numbers.
7. Execute 3 AES engines on 3 segments of payload using Engine Orientations randomly selected.
8. Operational Placement Engine inserts 6 hex digits, this block's Pre-Processor and AES seeds.
9. Checksum Engine calculates checksums using selected key
10. Operational Placement Engine inserts 20 hex digits, includes checksums and Checksum Engine Key numbers.
11. Using Operational Placement key number based upon first and last payload nibbles plus 3 digits of Post-Processor number, place the randomly selected Operational Placement key and the low nibble of the Post-Processor seed number.
12. Execute Pseudo algorithm for selecting Post-Processor segment pointers.
13. Execute Vernam Post-Processor, Xor'ing values from 3 segments selected with the internal data stream numbers.
14. Swap first and last characters with two in the middle as directed by an Overhead Placement key.
15. Convert bytes of data to characters, write out the ciphertext block.
16. Transfer 'Next Block' Post-Processor numbers to 'this block' locations and, if more encrypting, go back to (2.).

The Overhead Placement key number, call it ‘A’, used to determine the locations of most of the overhead digits placed during the encryption process, needs to be safely delivered to the legitimate receiving decryption process. During step 11, the values of the end nibbles that are defined as to not be the recipient locations of other overhead digits, are obtained and mathematically combined with 3 bits from the current Post-Processor seed number, call it ‘B’, so that its range could be any number from 0 to 1,023 inclusive. The Placement key number ‘B’ is then used to place the digits that contain the value of number ‘A’.

After the Post-Processor has executed and produced the ciphertext character block, the end characters contain the encrypted ‘fixed purpose’ nibbles used to provide needed information to the decrypt engine. In this MOO, ‘fixed purpose’ characters are not permitted. The current Post-Processor seed number, call it ‘C’, is used to reference the Overhead Placement key. Two locations from Overhead Placement key ‘C’ contain the locations of two characters in the middle of the ciphertext block. Adjacent duplicate numbers within the Placement key are allowed, but cannot be used in this case; two values are therefore obtained that are found to be different. The end characters are then swapped as per these two positions so that now these ‘fixed purpose/fixed position’ characters are no longer in fixed positions. In the demonstration system, there are 1,024 Overhead Placement keys. Since the two locations in key number ‘C’ have 1,024 possible pairs of values. The numbers are mathematically brought into the range of 2 to 110 inclusive, the formula:

Number = (<number_from_key_table> Mod 109) + 2

The [<number> mod 109] is in the range of 0 to 108. Adding 2 to it brings it into the range of 2 to 110. The use of these two values to reposition these two ‘fixed purpose/fixed position’ characters will place them in any other location.

When the decrypt process randomly arrives at the Post-Processor seed number, it uses that number, the same ‘C’ in the encrypt process above, to reference that Overhead Placement key used by the encrypt process to swap the end characters. The decrypt process executes the swap again, placing the characters back at the ends regaining their ‘fixed purpose/fixed position’ status. The Post-Processor executes its decryption process, recovering the original end nibbles used to form ‘B’ above. Their values are obtained from the ends, along with the same 3 bits from the Post-Processor seed number, reforming the number ‘B’. The Overhead Placement key ‘B’ is then used to extract the digits defined by that key that contain the Overhead Placement key ‘A’ described above.

6. High Level Sequence - Decryption

The sequence for decryption is as follows:

1. Select an unused block number from 0 to 1,023 inclusive; set the offset pointer at 0. If all block numbers have been used, decryption has failed, block is corrupted.
2. Execute Pseudo algorithm for selecting Post-Processor segment pointers (same algorithm as the encrypt engine).
3. Execute Post-Processor on 3 digits of data, then:
    a. Obtain the first and last nibbles of the data stream, add in 2 bits of the Post-Processor seed number forming an Overhead placement engine key number
    b. Obtain one nibble, position defined by the table number just formed, and test for its being equal to the lower nibble of the then-current Post-Processor seed. If it is not equal, increment the offset; if less than 111, go back to (2.), if equal, go back to (1.).
4. Execute Post-Processor on all of the ciphertext data
5. Extract 4 nibbles as directed by the Overhead Placement engine key number obtained in (a.) above. If 3 of the nibbles combined form an Overhead Placement Key number greater than 1,023, go back to (2.) or (1.).
6. Using the newly formed Overhead Placement Key number, extract 20 digits. Obtain the Checksum key number. If greater than 1,023, go back to (2.) or (1.).
7. Extract the Pre-Processor offset, next block offset and checksums. If the checksums are greater than the maximum allowed, go back to (2.) or (1.).
8. Calculate the checksums of the data stream.
9. If the checksums do not match the extracted, go back to (2.) or (1.).
10. Extract 6 more digits, create the Pre-Processor and AES seeds, and the Operational Orientation mask for the AES.
11. Execute the AES engines on the internal data segements, reverse orientations of the encryption sequence.
12. Extract 4 digits, form the Post-Processor seed and offset for the next block.
13. Execute Pseudo algorithm for selecting Pre-Processor segment key access numbers (same function as encrypt).
14. Execute Pre-Processor on remaining AES output data stream.
15. Convert Pre-Processor output data stream to ASCII characters, write this to the plaintext output file.
16. If a ‘text termination’ indication was not detected, use the Post-Processor seed extracted in step (12.) and go to (4.) to continue decrypting.

## 7.	Checksum Engine

The checksum engine used in this MOO is designed so that with any given single internal data stream, 1,024 different checksums can be obtained so that the digits cannot take any fixed value as is the case with current checksum systems. The design is different so as to produce a much wider range of values than what is normally encountered. In testing with the current checksum key loaded, the checksums obtained range from 9,875,627 to 16,677,545 for a 99-byte internal data stream.

The key table is constructed in a two-step random process:

1. A 256 location structure is loaded as follows:
    a. Location 0 contains a random value from 0 to 1,023
    b. Location 1 contains a random value from 1,024 to 2,047.
    c. This continues until location 255 contains a random value from 261,120 to 262,143.
2. These locations are then selected at random and loaded into a second structure that becomes the checksum key table.

When an internal data stream is checksumed, the 99 bytes reference the randomly selected key and the values from each key reference are summed. The first 20 locations of the first two keys are provided:

```
chkSum(0,0) =   2,062    chkSum(0,10) = 233,538
chkSum(0,1) = 164,389    chkSum(0,11) = 253,635
chkSum(0,2) = 195,521    chkSum(0,12) = 103,620
chkSum(0,3) = 111,445    chkSum(0,13) = 154,103
chkSum(0,4) = 129,758    chkSum(0,14) =  72,262
chkSum(0,5) = 107,277    chkSum(0,15) =  91,337
chkSum(0,6) = 252,581    chkSum(0,16) = 237,781
chkSum(0,7) =  92,409    chkSum(0,17) =  84,412
chkSum(0,8) =  50,657    chkSum(0,18) = 132,787
chkSum(0,9) =  21,117    chkSum(0,19) = 166,462

chkSum(1,0) =  53,647    chkSum(1,10) = 148,747
chkSum(1,1) = 105,137    chkSum(1,11) = 229,501
chkSum(1,2) = 260,736    chkSum(1,12) = 193,503
chkSum(1,3) =  24,208    chkSum(1,13) = 179,625
chkSum(1,4) = 223,507    chkSum(1,14) = 184,262
chkSum(1,5) =  94,805    chkSum(1,15) =  84,611
chkSum(1,6) = 169,279    chkSum(1,16) = 174,996
chkSum(1,7) = 208,461    chkSum(1,17) = 200,336
chkSum(1,8) =  12,869    chkSum(1,18) = 204,347
chkSum(1,9) =  32,310    chkSum(1,19) = 196,176
```

For example, when an ASCII 14 character is encountered, 72,262 is added to the accumulator if key 0 is used, 184,262 if key 1 is used. The value added to the accumulator for any ASCII can be any number from 0 to 262,143 depending upon the key used.

8. Pseudo Processes for selecting numbers

I. Pseudo process, Pre-Processor segment number selection

The Pre-Processor's seed number is randomly selected, along with a total of 9 other numbers used to complete the encryption of a block. In the demonstration system, the random numbers that are used in the pseudo selection process in the three areas are checked to not being equal to any other so as to not cancel their effect when Xor'ed. Some of these numbers are then applied to a pseudo selection process to determine the block starting points for the 3 94-byte key segments. A chain table, the one used in the technology demonstration system is provided, in part, in Appendix C. The chain table used in the demonstration system contains 1,024 numbers from 0 to 1,023 inclusive. Take the following example for the Pre-Processor selection process:

Random selections to start for one particular block: Pre-Processor seed = **909**, Pre-Processor initial offset = **18**, AES seed = 433, Post-Processor seed = 887, Checksum key number = 402, Overhead Data key number = 653.

```
tblRef1 = 909 Xor 433 Xor 887 = 331
block1 = chainTbl(331) = 138
prePrPtr1 = (138 * 111) + 18 = 15,336

tblRef2 = 138 Xor 887 Xor 402 = 623
block2 = chainTbl(623) = 795
offset2 = chainTbl(18) Mod 111 = 10
prePrPtr2 = (795 * 111) + 10 = 88,255

tblRef3 = 795 Xor 402 Xor 653 = 4
block3 = chainTbl(4) = 658
offset3 = chainTbl(10) Mod 111 = 96
prePrPtr3 = (658 * 111) + 96 = 73,134
```

From the initial Pre-Processor seed **909**, segment pointers 15,336, 88,255 and 73,134 are used with these random numbers selected. If the Chain Table is changed, the same seed will result in a change in the segment pointers set. So the makeup of the chain table is crucial to proper segment initialization.

Additional samples are detailed in Appendix B

II. Pseudo process, AES Engine key number selection

The process to select the key numbers from the AES seed start selected (during encryption) or extracted (during decryption) is simpler due to no multiplication of the numbers by 111 or additional offset to reset and add needed. The steps:

1. The AES, Pre-Processor and Post-Processor seeds are Xor'ed together and the value references the chain table. If the resulting number is the same as the AES seed, then the number becomes the chain table reference using the seed number only as the reference. This is the first AES key number.
2. The AES, Post-Processor seeds and Checksum numbers are Xor'ed together and the value references the chain table. A do loop is executed to make sure this second number is not equal to the AES seed or the first number determined in (1.). This is the second AES key number.
3. The AES seed, Checksum and Overhead key numbers are Xor'ed together and the value references the chain table. A do loop is executed to ensure this third number is not equal to the AES seed or either of the first two numbers determined in (1.) and (2.). This is the third AES key number.

Example, numbers selected: AES seed = 669, Pre-Processor seed = 178, Post-Processor seed = 188, Checksum key = 892 and Overhead Data key = 772. Sequence:

```
V1 = 669 Xor 178 Xor 188 = 659
AesKey1 = chainTbl(659) = 375
<375 is not equal to 669, number is ok>

V1 = 669 Xor 188 Xor 892 = 349
AesKey2 = chainTbl(349) = 852
<852 is not equal to 669 or 375, number is ok>

V1 = 669 Xor 892 Xor 772 = 741
AesKey3 = chainTbl(741) = 136
<136 is not equal to 669, 375 or 852, number is ok>
```

Using seed 669, AES key numbers 375, 852 and 136 were then obtained. If the Chain Table is changed, the same seed will obviously change the key numbers selected. So the makeup of the chain table is crucial to proper key number initialization.

Additional samples are detailed in Appendix B

III. Pseudo process, Post-Processor Engine segment number selection

This process is necessarily simpler. The first two engines used the random numbers selected for various engines. But when the decryption engine is attempting to determine the randomly selected seed and offset, it does not have the other randomly selected numbers as they have yet to be extracted from the decrypted text. The Post-Processor has to be successful in decrypting the initial ciphertext before the randomly selected numbers for the other engines can be extracted. Numbers that do not exist for the decrypt engine obviously cannot be used to determine the offsets. Therefore, the process only involves the initial and subsequent references to the chain table using only the randomly set Post-Processor seed and offset numbers. An actual sequence, with the Post-Processor seed equal to 577 and initial offset equal to 53:

```
V1 = chainTbl(577) = 25
postProcessorPtr1 = (25 * 111) + 53 = 2,828

offset2 = chainTbl(53) Mod 111 = 2
V2 = chainTbl(25) = 366
postProcessorPtr2 = (366 * 111) + 2 = 40,628

offset3 = chainTbl(2) Mod 111 = 4
V3 = chainTbl(366) = 7
postProcessorPtr3 = (7 * 111) + 4 = 781
```

From the initial seed of 577, offsets 2,828, 40,628 and 781. If the Chain Table is changed, the same seed will obviously change the key numbers selected. So the makeup of the chain table is crucial to proper segment initialization. If all of the other keys are identical but the chain is different between two users, they cannot decrypt each other's text.

IV. Pseudo process flexibility

The pseudo process selected for any engine is not set in stone, at least not for consideration of this MOO. It can obviously be varied for any particular design, either improved or changed. If it is varied from that specified in this document, then it should be obvious that even if the same master key was used, the system would not be capable of decrypting text produced by this demonstration system and vice versa. It may be determined that some other pseudo process is better for any of the engines than what is specified here and the suitability of the specified pseudo algorithms is therefore beyond the scope of this document. They are included here to offer one design possibility. Also beyond scope is whether or not the design of the pseudo process should be fixed for an accepted version of this MOO.

9. Encryption – creating a flexible unknown

Within this design document are significant entries detailing several encryption sequences. All of these entries were extracted from files produced by the technology demonstration system. For practical reasons, not all the files produced are included. Example 1 takes up 4 pages in this document with some format changes to fit this size page. For example, the set of data lines for the Pre-Processor take up 3 sets of lines in this document; they occupy one set of lines in the original document and there is a space character separating the two hex digits for each character of the input text – far too wide for this document's format. If you need to see all of any specific or all examples, please feel free to contact the author, information is on the first page, for the original example files used in this document's creation. All of them are purely text files, easily viewed by any text file editor. PDF files containing 'Landscape' formatted text of these example files on legal size paper are also available.

A normal ciphertext encryption output and the vital information concerning the encryption process is illustrated in the 4-page Example 1.

The Pre and Post-Processor displays consist of 5 rows of hexadecimal digits. If you Xor each column's first 4 digits, you will arrive at the value of the 5th digit. Example: the 1st column, '5', 'E', '7' and '5' = '9'. 4th column, '8', '7', '3' and 'A' = '6'.

In the sections regarding data insertions, the individual hex digits are placed right below the position numbers in the data sample. Take the first example of inserting '31E5' in the line. Follow the dashed line to the '1', near the end (numbers are red) and look just beneath it and see the first '3'. Go back to where the red '2' is and see just beneath it is the ‘1’. The next line up where the red ‘3’ is and see just beneath it is the ‘E’, and back to the last line where the red ‘4’ is and see just beneath it is the ‘5’. The decryption process will extract them in reverse order, extracting the ‘5’, then ‘E’ then

1234

‘1’ and then ‘3’ to form the original data line of ‘31E5’. There are 1,024 patterns of places to insert these digits, so any location, except the first or last position, can receive an overhead digit.

```
-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

Without the red position digits flagging the actual locations of these digits, how would an attack algorithm be capable of extracting them even if it did know there was a ‘3’, ‘1’, ‘E’ and ‘5’ to extract? Any incorrect extractions would be fatal to any attack effort. Not only would they probably extract payload digits but would also be leaving actual Overhead data behind as payload and reconstructing an incorrect Overhead Placement key.

The AES section is self-explanatory, displaying the key, input and output texts for each respective engine.

At the beginning of each example is a statement that contains the random numbers set for the displayed example and allows the demonstration system to duplicate the documented scenario when the same numbers listed are provided. It will also be assumed that the input text, not in this list of numbers, is also the same. This statement is colored orange only in the first example. Due to the numbers and the ranges that each number can take, duplication of the specified scenario may be important in further consideration and possible demonstration of this MOO.

For all examples, 25 additional runs were accomplished executing the same operation as each illustration. If you wish to receive these additional examples, notify the author, information provided at the beginning of this document, and it will be provided to you. To request specific sets, request set ‘Example <n>’ and whether you want the original data files or a PDF file containing an only slightly modified version of the files.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor 228, 93, AES 751, Post-Processor 948, 92, Checksum Sub-Key 416 and Overhead Placement Sub-Key 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 5468697320697320612074657374206F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D6265722075736167652E1E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

```
AES Engine Keys, using seed 751, and directions:
Engine #1 Key 1,013, Used in Encrypt Orientation =
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Engine #2 Key 959, Used in Encrypt Orientation =
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Engine #3 Key 579, Used in Encrypt Orientation =
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

AES Engines (each) <Input Stream> | <Output Stream>:
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  FC954033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186E00019EE5C8162E

D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D38D215F18D7 |
  4C4EB5E4F9F05CCC8A147513C1BFCD9E6A2352F90A09590BC29BD79B76C431D6

18F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA812A38068151FA39 |
  ECBB71741D5E423AA3D02C1B59BE47374F911D2433B10C70FFA577E345386055

The Pre-Processor and AES seeds, 228 and 751
respectively, along with the 3 AES engine orientation
                                     123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:

-------------------------------1---------------4-6---
FC954033A67B860C05B0C8D51DDA9C104CB645F877CBEA221F86E

------25----------------------------3
00019EEEE5C8162E4C4EB5E4F9F05CCC8A1474513C1BFCD9E6A23

52F90A09590BC29BD79B76C431D6ECBB71741D5E423AA3D02C1B5

9BE47374F911D2433B10C70FFA577E345386055

The Checksum is then calculated and the checksums and
checksum sub-key number (416) are converted to the
            123456789ABCDEFGHIJK
hex string <6DAAB706CD5DA34071A0> and inserted in the
data stream:

--------6------------G----------------K--------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

---------------E---2-H---------------------7---------
86E00019EEEE5C83162DE74C4EB5E4F9F05CCC8A14740513C1BFC

9---C---J------------3-------8-------------A---1----5
CD9ED6A2A352F90A09590ABC29BD769B76C431D6ECBDB716741DB

---------D--------------4---B---F----I
5E423AA3DA02C1B59BE47374AF9151D24433B110C70FFA577E345

386055
```

Note: Without the red placement numbers illustrated above and taking the original hex string to extract, how would any attack algorithm be capable of determining which hex digits to extract and in what order? Take the first ‘6’ digit – which of the 11 ‘6’ digits would be the correct digit to extract?

```
Using the hex digits on either end of the data stream
plus 2 bits from the Post-Processor seed, assembled
as the new Overhead Key Number, the lower 4 bits of
the current Post-Processor seed (4) plus the Overhead
key number (425), used before this point, are
                               1234
converted to the hex digits <1A94> and inserted in
the data stream:

-----------------------------------------------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

-------------------------------4-------------------1-
86E00019EEEE5C83162DE74C4EB5E4F49F05CCC8A14740513C11B

----------------3------------------------------------
FCCD9ED6A2A352F990A09590ABC29BD769B76C431D6ECBDB71674

-------2
1DB5E42A3AA3DA02C1B59BE47374AF9151D24433B110C70FFA577

E345386055

Vernam Post-Processor Input Data, keys and Output:

Data   From  AES: FC9540337A67B860C05B00C8D51DDA9C104
Segment @  6,197: 028BF3E11432A31FAEA9B6C27391969CBC7
Segment @ 96,182: 0B5DA6856539976104047739530D51570BAB
Segment @ 11,292: C99A2770F5BD0F087D7341B3FAC23E3E128
Post Data Output: 3CD93227FED18316178580800F9B674E040

CB6045F877CBEA221F86E00019EEEE5C83162DE74C4EB5E4F49F0
3F99898B1084D11BCC1D2B18899D5453D66E7E1307D1A241117CE
9552A5D9D8354076DE7E4617E9A62682997A71EB915E47F8814E7
AF42E4A0B355F1CA868EB4AEB7F4D71C829C6E1264800AC9BA5F2
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

5CCC8A14740513C11BFCCD9ED6A2A352F990A09590ABC29BD769B
BE66FAC82F385584D10BEBCD1814748A19A740DB49EF1D6487E17
B664369A9138D1CD33E84235087C84B4FE8101DAF3ACA8AC69C13
FD77E303AB05B1720E0D44F5129D05A08A0A8609C631BBAA67026
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

76C431D6ECBDB716741DB5E42A3AA3DA02C1B59BE47374AF9151D
BE95D73CE1C830934B2A52F58BC678E9635A5195FD7FF88DF3495
FA581B3FBFB270D4E238A900097CF1E81D706D67DAE5D81B2B693
7867240D410A482C748057EB327C0862D3D3253A099410242E9EF
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

24433B110C70FFA577E345386055
07DE010F2FE4B5E135A153CD618C
2DFE1EB017BDABF3CE51FD0D9602
5C46E22F5A54E2F7E936D496EAD4
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

```
Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.
```

Example 1a

Suppose an attacker had the input and output data sequences for the Pre-Processor. The question that needs to be answered is:

> How would the attacker obtain the specific values in the three segments that took each input byte to the output byte, not to mention the address of each segment (required for correct key table reconstruction)?

In the first example, input ‘54’, key numbers ‘ED’, ‘7E’ and ‘54’, output ‘93’, assume they have the input ‘54’ and output ‘93’. The Xor of these two is ‘C7’. Because this is the Vernam algorithm, there is no mathematical ability available. This is unlike the original Vernam where if you have the input and output, the key number is the Xor of the input and output bytes, in this case ‘C7’. In this design, all the attacker would be able to obtain is the Xor value of all 3 segment bytes, ‘C7’. With over 16 million different combinations that could result in that single value of ‘C7’ that they might be able to obtain, their likelihood of randomly determining all 3 values (‘ED’, ‘7E’ and ‘54’) correctly is not very realistic, not to mention they have to do this 93 more times, and this is just for the Pre-Processor. As for the address, there is nothing mathematical that can be extracted from the ciphertext to recreate the pointer. As detailed earlier, they are determined using a chain table whose contents does not create any of the ciphertext.

With the output stream from the Pre-Processor being kept internal to this system, their likelihood is even less, with nothing to go on.

Example 1b starting on the next page shows another encryption of the same text with the keys and seeds changing, initially displaying the variety of key table selections and Overhead Data Placement variations possible.

The seed and number selections for this data: 136,107,979,3,941,779,745,76,698,105

Sub-Key Seeds used: Pre-Processor 136, 107, AES 979, Post-Processor 745, 76, Checksum Sub-Key 779 and Overhead Placement Sub-Key = 941

Vernam Pre-Processor Input Data and Output Stream:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 5468697320697320612074657374206F66
Segment @101,006: 8C7609CEB646EC6A5166FC298A9CF9974A
Segment @ 88,293: 15C20D271602C5D4D2C9BF3DD448F5C116
Segment @ 20,790: EF8D1B2EF316032A18940205422E30C46A
Prep Data Output: 305071A63F3E53B8BB482E7C79DA6ED777

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
7C329C69D454011E468030BA37A03B97E8FA214BE3054C21997C
5F57E3176378382D1F87841D188220E916A69EE9E02AB01D462C
C480E03931F1A6D5CD08DEFB377DAB4E5135B2EE8B707E4D6544
ABC5FC28EBADFA92FD7B0333767FD65FDD497924ED7FF203D570

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
5DC949B22C4E23250B49AE554B5467D602C2ABA50B8DD7B84D90
46C265ECF7232638A2B2082755571F0A6E085DA0FF9A118977D7
4A5677349F507EE704EF91F81B2783BB56D877481F1F8715C9DC
243E2F4A224F14978D5956ED6C55DB335F71E92384642E439AFE

n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D6265722075736167652E1E4141
ADD941F7F9B3AEF3752DB7B2EBF8188E9964AD84D97834E901
ADFE7FD0CCBE2374734C32629857E7FCE17AC9BDAD25CFCC58
03166703F193583975D668C0001643A242D7AAABA860197B23
7011304AE4CDBAD316C59B791FD5D9FC1A848FBCC27CA31F3B
```

The next ciphertext block's randomly selected table and offset seeds, 698, 105 are converted to the hex

```
        1234
string <ABA9> and inserted in the data stream:

------------------------------------------4----------
305071A63F3E53B8BB482E7C79DA6ED777ABC5FC289EBADFA92FD

--------------------------------------------------1--
7B0333767FD65FDD497924ED7FF203D5704A5677349F507EE7A04

-----------------------------------------------------
EF91F81B2783BB56D877481F1F8715C9DC7011304AE4CDBAD316C

-3--------2
5A9B791FD5BD9FC1A848FBCC27CA31F3B
```

AES Engine Keys, using seed 979, and directions:

Engine #1 Key 909, Used in Decrypt Orientation =
04303E91963C174EC2CE5FBFDC7DA24885F03AF275952ABE78964A39B708A451

Engine #2 Key 13 Used in Decrypt Orientation =
A9F74DB5ADE5CF8C54B2661225557A1C74659B0EE2881FF1272F0DC60F80391B

Engine #3 Key 508 Used in Encrypt Orientation =
2800712137F669EEF39D20FF5EC1CA096A580C6A0F4FB30D9C8AA68F4A2519A0

AES Engines (each) <Input Stream> | <Output Stream>:

```
305071A63F3E53B8BB482E7C79DA6ED777ABC5FC289EBADFA92FD7B0333767FD |
  DCE906CE28EA58B1A6DF18D404E595B78D74579BE7E6AC7F4D24B3D20AF40BBC

65FDD497924ED7FF203D5704A5677349F507EE7A04EF91F81B2783BB56D87748 |
  9D93D81A066B9546EA538986D62224500BF6965DDE043884F1A4D4749C0F6A45

1F1F8715C9DC7011304AE4CDBAD316C5A9B791FD5BD9FC1A848FBCC27CA31F3B |
  F19865EA73C45BFA3E130F973911BED7AFAA184D081996917C1C5C4783A81FA2
```

The Pre-Processor and AES seeds, 136 and 979 respectively, along with the 3 AES engine orientation

```
                                     123456
bits, are converted to a hex string <C883D3> and
```

inserted in the data stream:

```
-----------------5-----------------------------------
DCE906CE28EA58B1AD6DF18D404E595B78D74579BE7E6AC7F4D24

---------------------------------6------------------2
B3D20AF40BBC9D93D81A066B9546EA5383986D62224500BF69658

------------------------------------------------3----
DDE043884F1A4D4749C0F6A45F19865EA73C45BFA3E130F987391

-1----------------------------4
1CBED7AFAA184D081996917C1C5C47383A81FA2
```

The Checksum is then calculated and the checksums and checksum sub-key number (779) are converted to the

```
            123456789ABCDEFGHIJK
hex string <5CC2A6C760D0D092930B> and inserted in the
```

data stream:

```
-----------------------C------4----71----------------
DCE906CE28EA58B1AD6DF180D404E5295B7C58D74579BE7E6AC7F

---------------------9-------------------------------
4D24B3D20AF40BBC9D93D681A066B9546EA5383986D62224500BF

-E-----------------------5--K------B-D-----38--------
609658DDE043884F1A4D4749CA0FB6A45F1D9D865EAC773C45BFA

-------------A--J---2-F-------------------6I--------H
3E130F98739110CB0ED7CA9FAA184D081996917C1C635C47383A9

----G
81FA22
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (9) plus the Overhead key number (941), used before this point, are

```
                                1234
converted to the hex digits <3AD9> and inserted in
the data stream:
```

```
----------------2---3--------------------------------
DCE906CE28EA58B1AAD6DDF180D404E5295B7C58D74579BE7E6AC

--------------------1--------------------------------
7F4D24B3D20AF40BBC9D393D681A066B9546EA5383986D6222450

----------------------------------------------------4
0BF609658DDE043884F1A4D4749CA0FB6A45F1D9D865EAC773C49

5BFA3E130F98739110CB0ED7CA9FAA184D081996917C1C635C473

83A981FA22
```

Vernam Post-Processor Input Data and Output Stream:

```
Data   From  AES: DCE906CE28EA58B1AAD6DDF180D404E5295
Segment @ 97,756: A4EF36D9B83525104C11055317325A69D72
Segment @  3,610: B7B9EE15FF129543441363A7EDD811E69BD
Segment @ 46,695: 8CC1A175CEB6D7D8D839A79B3288E98C7C8
Post Data Output: 437E7F77A17B3F3A7AED1C9E48B6A6E6192

B7C58D74579BE7E6AC7F4D24B3D20AF40BBC9D393D681A066B954
FF139E5C917300C7008E39B09AAF6B9EF7C3CF938C91F2E392143
063805C1FFF5422407F6774EFC5E164468F58C1264B518DD82C04
C6F814B5E80F6330B88F725940D41FC39E5A452F7A6AB98505F7F
8816025CD112C6351388718395F768ED0AD09B97AF2649BD7EB6C

6EA5383986D62224500BF609658DDE043884F1A4D4749CA0FB6A4
814C0966C0D0F28AF972685748AB8CBBED8AC4D3A2B4B10E2BA96
B2F47D60F082354F1A60CC3B82E44ED1B2C2AE055C2F5C5694461
664A85090FBA305779C288412B997A4CA88BAFCEFC40BE539440A
3B57C936B93ED5B6CADBDA24845B6622CF4734BCD6AFCFABD0C59

5F1D9D865EAC773C495BFA3E130F98739110CB0ED7CA9FAA184D0
18F2E48C7D44596689652BB56F0124D62C2F5D28E582E7FC14706
A533DAB0B5EC16F0CB00634489D9D92071A3CB9C77C6E578E8289
04052B765B6E17626B9463B3B92B832AE71B2BBEF57466EC833B4
E6D988CCCD6A2FC860AAD17C4CFCE6AF2B877604B0FAFBC2672EB

81996917C1C635C47383A981FA22
46BE9875105729F347DE4F8C98DD
BC7E96E13CAA503BD4283C841168
5D944C3B99ACDE2FF723084C13B3
26CD2BB8749792231756D2C56024
```

After swapping the first digit pair with pair #58 and the last digit pair with pair #86, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
[~□w¡{?:zí□žH¶¦æ□(□`%Í□,cQ8‡□9_vŽÐ-□¹zòd›×ël;WÉ6¹>Õ¶Ê
ÛÚ$„Cf"ÏG4¼Ö¯Ï«ÐÅžm˜ŒÌÖ¢ü†□-□ÄÏÎ$ò¸w`K□¯¼&rë&Í+¸t—’#□
VÒÅ`j
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 36, ASCII 9, to character '□', ASCII 2.
Character 80, ASCII 10, to character '□', ASCII 2.

Example 1b

In Example 2 beginning on the next page, the demonstration system was set up to randomly reset the three AES keys and the first two Post-Processor key segments. It then calculated the third Post-Processor segment values so as to produce the same ciphertext output as in Example 1a.

Multiple Block Encryption Examples are available

The ‘Abstract’ paragraph at the beginning of this document was extracted and encrypted 10 times producing 10 ciphertext files containing 19 blocks of ciphertext using this system. Each file was then decrypted. This system concurrently produced demonstration outputs that detailed the keys, segments, overhead data positions, etc. that were used for each encrypt and decrypt operation. These individual output files are available for inspection. The files are in printable version within 5 separate PDF files; two encrypt/decrypt operations with the outputs from each operation interleaved for easy comparison within each file. Any number of these files are available upon request.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696E
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
F62052616E646F6D204E756D62657220757361676 52E1E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:

-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

The AES keys specified here were randomly recreated during encryption and temporarily replaced the original keys in the specified location.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 1,013, Used in Encrypt Orientation =
3DF5C74F457D43E10E2311E314E23699127F5842EB6E0EA43CF0CD5F2616F45E

Engine #2 Key 959, Used in Encrypt Orientation =
18959C280B64A45C52FBEF80FB7632666A5C04D96334A1150CFD3A03B9A0D43B

Engine #3 Key 579, Used in Encrypt Orientation =
FBCBDC5D0E04752E73A38AC5845C05C22C0E26F329E3BB163F3385B416A07E34

AES Engines (each) <Input Stream> | <Output Stream>:

```
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  4FFF8AE2E78984E098EB4DB39305F6033E2B27CA17E6480F6241F03ED5EA3695

D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D38D215F18D7 |
  AB6DAF99EE192EC391CA19EC6673EEF66E56FF8015403158DAAC50AAF7A358DC

18F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA812A38068151FA39 |
  5F5C5A0B0E48F2FDEFF0FE37BA341769C13B33C507F9CB73631EAD6EFFF160CE
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation

```
                                      123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
4FFF8AE2E78984E098EB4DB39305F60033E2B27CA17E64820FF62

------25-----------------------------3
41F03EEED5EA3695AB6DAF99EE192EC391CA149EC6673EEF66E56

FF8015403158DAAC50AAF7A358DC5F5C5A0B0E48F2FDEFF0FE37B

A341769C13B33C507F9CB73631EAD6EFFF160CE
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the

```
            123456789ABCDEFGHIJK
hex string <6D1A2B26C444614331A0> and inserted in the
data stream:
```

```
--------6------------G----------------K--------------
4FFF8AE2BE78984E098EB34DB39305F60033E20B27CA17E64820F

---------------E---2-H----------------------7--------
F6241F03EEED5EA1369D53AB6DAF99EE192EC391CA1429EC6673E

9---C---J------------3-------8-------------A---1----5
CEF646E5A6FF80154031518DAAC506AAF7A358DC5F54C5A60B0E2

---------D--------------4---B---F----I
48F2FDEFF60FE37BA341769CA13B433C4507F19CB73631EAD6EFF

F160CE
```

```
Using the hex digits on either end of the data stream
plus 2 bits from the Post-Processor seed, assembled
as the new Overhead Key Number, the lower 4 bits of
the current Post-Processor seed (4) plus the Overhead
key number (425), used before this point, are
                                1234
converted to the hex digits <1A94> and inserted in
the data stream:

-----------------------------------------------------
4FFF8AE2BE78984E098EB34DB39305F60033E20B27CA17E64820F

-----------1------------------------------4----------
F6241F03EEE1D5EA1369D53AB6DAF99EE192EC391C4A1429EC667

-----------------------3-----------------------------
3ECEF646E5A6FF801540315918DAAC506AAF7A358DC5F54C5A60B

-----------------------------2
0E248F2FDEFF60FE37BA341769CA1A3B433C4507F19CB73631EAD

6EFFF160CE

Vernam Post-Processor Input Data, keys and Output:

Data   From  the   AES: 4FFF8AE2BE78984E098EB34DB3930
Seg Re-Random @  6,197: 93A9A7E4D435EA9F9E29BC4881CAF
Seg Re-Random @ 96,182: 0B378685B95EA5E1391F47B810B18
Seg Calculated@ 11,292: EBB899A42DC25426B93DC83D2D731
Post-Proc  Data Output: 3CD93227FED18316178580800F9B6

5F60033E20B27CA17E64820FF6241F03EEE1D5EA1369D53AB6DAF
DA2D8A8B6DAABF4F1DD54F508D63FAC632E76D05237E417FD2FA5
8C79DEBC1507CF6E7C685673971C4C01476FA11B02EFB8A0F12DF
7DD417C7B192068C30531EA78762080ABA2288BAACB4215BD457C
74E040CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A9

99EE192EC391C4A1429EC6673ECEF646E5A6FF801540315918DAA
05F024B0D3082DB7A4174E37B94F3DCE54E0C46695AA2EAE8CE4E
ECCAB0BC67EABA073EE456E411C62E982B7C5043A1287A181A03E
3D3BA68BCED61670D84B2443846776C4CD6CA7319DA5F80357F15
4DEF2BA9B9A545610026FAF7122093D45756CC94BC679DECD9CCF

C506AAF7A358DC5F54C5A60B0E248F2FDEFF60FE37BA341769CA1
FCF0940A885453318C1B2B7234EF6D4B3103DA9219A6291574F90
DE812738AAAE579B1F30032CC32EE9EE441B95419F73ACE9B89DE
7293A08FEF7B00060A51F3FC76FCF11057C5968289C3E221D8EAE
95E4B94A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441

A3B433C4507F19CB73631EAD6EFFF160CE
C2C91E33335C3955A001553656C090DF6A
300ACC0BE830689FB2BAF08BF03DFF3295
870915AEAED5C96F1CDBFB75ED3DF0F03E
D67EF45225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

Example 2

For the Pre-Processor, there are 256 possible values for each position in each of the three 94-byte segments. For one column of bytes, the number of possible selections is 256^3 = 16,777,216. For 94 columns, the number of possible keys is $16{,}777{,}216^{94}$ = 1.329 x 10^{679}. Three AES engines with 2^{256} = (1.15 x 10^{77} keys each)^3 = 1.52 x 10^{231}. Two of the three segments of the Post-Processor can be randomly set, yielding 256^2 = 65,536 possible values for each column. For 111 columns, the number of possible keys is $65{,}536^{111}$ = 4.26 x 10^{534}. All these multiplied = 8.60 x $10^{1{,}444}$. Even if only 0.000001% of these possible keys are realistically random, it becomes 8.60 x $10^{1{,}436}$, a great deal more than the number of possible AES 256-bit keys.

Because any attacker would be able to create this many keys with this MOO design that correctly decrypt a single block of ciphertext, they would be forced to give up on any attack. Even considering technology far into the future, they would not be able to carry out the necessary computations needed.

In Example 3 on the next 4 pages, the demonstration system was executed again, re-randomizing the same keys with the same settings as those in Example 2 above. It should now be evident that this document could easily exceed 1,000 pages and still include examples showing different random AES keys and the first two of three segments of the Post-Processor with all of them resulting in exactly the same ciphertext output.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D6265722075736167652E1E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex
```
          1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

The AES keys specified here were randomly recreated during encryption and temporarily replaced the original keys in the specified locations.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 1,013, Used in Encrypt Orientation =
```
771595BCC9000F6F9D691C4AF472E176134379140BE0A74C87BF4EE8FEDF6608
```

Engine #2 Key 959, Used in Encrypt Orientation =
```
664C487DABBACF6C3C584629C77A4AF3F6B987148B01EB6EFAD9535B74D0D248
```

Engine #3 Key 579, Used in Encrypt Orientation =
```
5445BD827CE2AC796B810A132F42DE11B7084D9AC19825127B46CCFFF059DD2E
```

AES Engines (each) <Input Stream> | <Output Stream>:
```
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  DA27BCCC07E0CA97690296EBB9439A8CA00B89DCC6281DEACA87CD0EC6219E59

D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D38D215F18D7 |
  168B4E9C9D488AEEC536C0F383582E2366DB933EFA912FC5E69AFFC3D0CFAA31

18F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA812A38068151FA39 |
  D0AB9A4789F6AA13BF54EDEBF93EB0495380A7BF41EC61EA0A7218CFE3950224
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation

```
                                             123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
DA27BCCC07E0CA97690296EBB9439A80CA00B89DCC6281D2EFACA

------25-----------------------------3
87CD0EEEC6219E59168B4E9C9D488AEEC536C40F383582E2366DB

933EFA912FC5E69AFFC3D0CFAA31D0AB9A4789F6AA13BF54EDEBF

93EB0495380A7BF41EC61EA0A7218CFE3950224
```

Checksum is calculated and checksums and checksum sub-key number (416) are converted to the hex string

```
 123456789ABCDEFGHIJK
<6D9BECA6CC60197AB1A0> into the data stream:
```

```
--------6------------G----------------K--------------
DA27BCCCC07E0CA976902A96EBB9439A80CA000B89DCC6281D2EF

---------------E---2-H----------------------7--------
ACA87CD0EEEC62199E5D9B168B4E9C9D488AEEC536C4A0F383582

9---C---J------------3-------8-------------A---1----5
CE23066DAB933EFA912FC95E69AFF6C3D0CFAA31D0ACB9A64789E

---------D--------------4---B---F----I
F6AA13BF514EDEBF93EB0495B3806A7B7F41E1C61EA0A7218CFE3

950224
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                                1234
converted to the hex digits <1A94> and inserted in
the data stream:
----------------3-----------------------2------------
DA27BCCCC07E0CA9976902A96EBB9439A80CA000AB89DCC6281D2

---------------------------------1-------------------
EFACA87CD0EEEC62199E5D9B168B4E9C91D488AEEC536C4A0F383

-----------------------------------------------------
582CE23066DAB933EFA912FC95E69AFF6C3D0CFAA31D0ACB9A647

--------------------------4
89EF6AA13BF514EDEBF93EB04945B3806A7B7F41E1C61EA0A7218

CFE3950224
```

Vernam Post-Processor Input Data, keys and Output:

```
Data   From  the   AES: DA27BCCCC07E0CA9976902A96EBB9
Seg Re-Random @  6,197: 502F4FC567D9046F9B17D9C71EAA2
Seg Re-Random @ 96,182: 56CA7AD0882D9E5964EB73C6502B8
Seg Calculated@ 11,292: E01BBBFED15B15897F1028282FA15
Post-Proc  Data Output: 3CD93227FED18316178580800F9B6

439A80CA000AB89DCC6281D2EFACA87CD0EEEC62199E5D9B168B4
4A451ACD2AFE06CD41ED2CB5734626B6621567B03DC82F6E3795F
729EEE6DAAD21FAFDBE37335B29AED37472AF2F1D82D1D26DAAA9
0FA134A469ABABF379E65BD94549C233D49AE86D6237626DBAEEB
74E040CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A9

E9C91D488AEEC536C4A0F383582CE23066DAB933EFA912FC95E69
B20A90F298CC20BD7AD600C0ECACEE893587BF033EDC2444910C9
0C3C2D98EBE07F92EB875B93805F2D8B62D8CA1C13BB6C7C6852B
1A108B8B4067DF7855D7522726FFB2E666D300B87EA9C728B5744
4DEF2BA9B9A545610026FAF7122093D45756CC94BC679DECD9CCF

AFF6C3D0CFAA31D0ACB9A64789EF6AA13BF514EDEBF93EB04945B
19C1C2FF0C2AF0429FFCEF6CF5BC70FF035AC71410FA51E932962
122DA43570E8BD4499FE02077B129F67FF9215CE7CA0A8ECB3101
31FE1C50DDB1A4256704368588587FA33B1F7F98BF0F947FB5879
95E4B94A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441

3806A7B7F41E1C61EA0A7218CFE3950224
081B17E6206F51B7316BBAF61852FC0372
9A3FC480C30E2583CF326CC62984C4466E
7C5C808332B9E93B6950E44DDB0AC33A37
D67EF45225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
″Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

Example 3

In Example 4 beginning on the next page, **<u>all</u>** keys and segments are randomly reset. The last <u>two</u> key segments of the Post-Processor have their values randomly reset until their combined Xor value results in the ciphertext not changing in the Post-Processor engine. Pertaining to the Post-Processor:

1. The key ‘segment 1’ byte is randomly reset.
2. A ‘Do’ loop is executed where:
    a. Two numbers from 0 to 255 are randomly selected.
    b. If the Xor of the input byte to the Post-Processor from the AES engine, the segment 1 number just created in (1.) and the two numbers selected in (a.) does not equal the ciphertext byte to duplicate, go back to (a.) to reselect two new random numbers.
3. Once the ‘Do’ loop exits, the Vernam Xor of the 4 numbers (input and the 3 table values) produces the ciphertext byte needed.

As you can see, this requires more work for this system, taking up to 3 seconds to produce each sample. This capability, producing a completely new set of <u>randomly</u> created keys (no ‘calculated’ values for the 3rd segment in the Post-Processor as in Examples 2 and 3) is made possible by utilizing at least 2 segments to Xor in the Post-Processor Vernam engine.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text Characters:   T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
ReRandom@112,536: ABCF6B7A8E163C5F19835DF8FF09369FD2
ReRandom@ 84,239: D8FF222E5BCF04E0E355835D52A7BBA1FD
ReRandom@ 48,603: 87630FE92A8E32FA719A9743E9A7848F09
Prep Data Output: A03B2FCEDF3E7965EA6C3D83377D29DE40
```

```
   t h e   T i t a n i u m   C r y p t o g r a p h i
207468652054697461 6E69756D2043727970746F677261706869
65F162D9025E0B9728F55361230D790A8C68D14CB40650180AC3
45826AD0A0B2D533E927CC57EBF09321DAE55F1339DE14159479
756693808988366A0E0FE8E1ED567D2244324D10CF989D21D2BF
7561F3EC0B3081BAAEB31EA2488BD47B6BCFB7202532B85C246C
```

```
 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
104A7137B8C73528880D2E0A544AFB266BD1AA9804927A6D4659
60E8DFF9DAAE2A2D9A54780AD71764F3EB27AC7363F37E4858EE
9B7C1D6A86772CB73FE9F56A326F2B80535867F2374623EB5B96
88FEE0DD976A56DF0DF5CD09C34BC421BAC10F39034244BA2C4E
```

```
 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D62657220757361676 52E1E4141
2CDB259D07101358FE2E814F033E30BC3675D2DA0E69EFAA74
D6F145D080E5DBA6CAF4ED6FE4F672D84550512616752C7243
4FA4B39169DF542927A92D70761BC38B6B3D3109B34F6B1843
DBAE81BD804EF3BA333D343DF3B6F3CF6D6BD392CE7DB68135
```

```
The next ciphertext block's randomly selected table
and offset seeds, 798, 69 are converted to the hex
        1234
string <31E5> and inserted in the data stream:
-----------------------------------------------------
A03B2FCEDF3E7965EA6C3D83377D29DE407561F3EC0B3081BAAEB

-----------------------------------------------------
31EA2488BD47B6BCFB7202532B85C246C88FEE0DD976A56DF0DF5

----------------------------------------------3------
CD09C34BC421BAC10F39034244BA2C4EDBAE81BD804EF3EBA333D

--2-4----------------1
34135DF3B6F3CF6D6BD3932CE7DB68135
```

The AES keys specified here were randomly recreated during encryption and temporarily replaced the original keys in the specified locations.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 1,013, Used in Encrypt Orientation =
```
0D7C84BE09DE62B3816E5A4CDFD47BDAB36E16CBCB5968F458ACFCDC2A4A7467
```

Engine #2 Key 959, Used in Encrypt Orientation =
```
6D9C2092873252EB2DAA1D8DA440572A42AC88E5D92528EC75AF0025F3462274
```

Engine #3 Key 579, Used in Encrypt Orientation =
```
6EB29A6617D32EC95B4B9D01A58DAC919E2D851AB6ACB74DA039364126876B12
```

AES Engines (each) <Input Stream> | <Output Stream>:
```
A03B2FCEDF3E7965EA6C3D83377D29DE407561F3EC0B3081BAAEB31EA2488BD4 |
  08539D55339367C1190B2833A5930FD054B706D96E1CE18A53B6632F4D55D4E6

7B6BCFB7202532B85C246C88FEE0DD976A56DF0DF5CD09C34BC421BAC10F3903 |
  0E6E7BA3FD641FAB7B4074E980D9D5C63959DAFAD4D77851CE0EA7C68802C500

4244BA2C4EDBAE81BD804EF3EBA333D34135DF3B6F3CF6D6BD3932CE7DB68135 |
  433EAF546FDDFB2AF3D52B4AB5237709721A375B7CDB99B4CB5BF0C05FDCD063
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation bits, are converted to a hex string <0E42EF> and inserted in the data stream:
```
                                     123456
bits, are converted to a hex string <0E42EF> and
```

```
-------------------------------1---------------4-6---
08539D55339367C1190B2833A5930FD0054B706D96E1CE128FA53

------25-----------------------------3
B6632FEE4D55D4E60E6E7BA3FD641FAB7B40744E980D9D5C63959

DAFAD4D77851CE0EA7C68802C500433EAF546FDDFB2AF3D52B4AB

5237709721A375B7CDB99B4CB5BF0C05FDCD063
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the hex string <6A739596B2E38ABD31A0> and inserted in the data stream:
```
            123456789ABCDEFGHIJK
hex string <6A739596B2E38ABD31A0> and inserted in the
```

```
--------6------------G----------------K--------------
08539D555339367C1190BD2833A5930FD0054B0706D96E1CE128F

---------------E---2-H----------------------7--------
A53B6632FEE4D55AD4EA630E6E7BA3FD641FAB7B407494E980D9D

9---C---J------------3-------8-------------A---1----5
B5C63395A9DAFAD4D778571CE0EA76C68802C5004332EAF6546F9

---------D--------------4---B---F----I
DDFB2AF3D852B4AB52377097321AE375BB7CD1B99B4CB5BF0C05F

DCD063
```

```
Using the hex digits on either end of the data stream
plus 2 bits from the Post-Processor seed, assembled
as the new Overhead Key Number, the lower 4 bits of
the current Post-Processor seed (4) plus the Overhead
key number (425), used before this point, are
                               1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
-----------------------4-----------------------------
08539D555339367C1190BD24833A5930FD0054B0706D96E1CE128

-----------------------------------------------------
FA53B6632FEE4D55AD4EA630E6E7BA3FD641FAB7B407494E980D9

------------------------------------------------2----
DB5C63395A9DAFAD4D778571CE0EA76C68802C5004332EAFA6546

------1-3
F9DDFB129AF3D852B4AB52377097321AE375BB7CD1B99B4CB5BF0

C05FDCD063
```

Vernam Post-Processor Input Data, keys and Output:

```
Data   From  AES: 08539D555339367C1190BD24833A5930FD0
ReRandom@  6,197: 81F8E3FBFF787465540FA53428B263BBC81
ReRandom@ 96,182: 0F35ACCF52EFE10F08B329E823FA787B5A9
ReRandom@ 11,292: BA47E046007F20005AA9B17887E925BE6B8
Post Data Output: 3CD93227FED18316178580800F9B674E040

054B0706D96E1CE128FA53B6632FEE4D55AD4EA630E6E7BA3FD64
EAE75F5CCA0A0FA59455F1C8814755C1ADB273EE8DFE3CC86FD41
BC4A4D6479865766EC34B7CB112CAC71C30D00D05024CBF7D0B25
9D0F983466CDCEA7DBF02C143D655C6C758C7195537D4A115E42B
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

1FAB7B407494E980D9DB5C63395A9DAFAD4D778571CE0EA76C688
C5AFEC69AD1FCBC7AE4FBB093D7599BB1345CE122DD5E2A469979
5F4240AF4B3CC61CEFB7C1A4777699DCB8EA8E8A8EDB5DD6C0A1E
2CFF72C3F3B7C2A16F31065DA70ECB04925E50803E197D2C9B156
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

02C5004332EAFA6546F9DDFB129AF3D852B4AB52377097321AE37
2CED997A62651273AE5E78D7FD89AA08F1313B938CDBE0ABCB1CC
F96DE697B28417126E13938C7DE23F4DE8683E79F3F67462A4045
9D2BA67611C640792F3B2F5A080D4424E4D502EB822047E61214A
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

5BB7CD1B99B4CB5BF0C05FDCD063
1C5584F924CB562FEA80C8E9F546
E878CB11C3BA050897D603BE7572
FDBF447210B89B3CE8B3ABE52D58
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□iÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.

Example 4

Take the case where the raw Xor of the AES input and the desired ciphertext output would normally produce a single Vernam Xor key that would <u>not</u> take on a random appearance. Because this version of the Vernam utilizes three segments, they can still be created in a random way where their values Xor'ed would satisfy the not-so-random single key conditions and still produce the needed freeze on ciphertext content.

In Example 5 starting on the next page, the Overhead Data Placement key was re-randomized from Example 1. What this means is that the key in the structure number has changed, <u>not the key number</u>. This is important because this key number is used in the pseudo algorithm for selecting other keys in other engines and would significantly change the data being acted on by this engine.

All other keys and seeds will remain the same in this example, so the encryption of the data remains the same. You will notice that the positions of the data inserted has changed, not the data or any of the payload. This will begin to illustrate that the random numbers used to place the overhead digits can potentially be inserted in any location, except the first and last, depending upon the makeup of the list of random numbers within the Overhead Data Placement key number randomly selected. The first and last digits are extracted by both encrypt and decrypt engines to help form the Overhead Data Placement key number used to extract the correct first set of data digits.

After swapping the first digit pair with pair #61 and The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D626572207573616765 2E1E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

--------------4------------------2--------------
5C629A76635D35C565417FE06381054904F81791EF5336592CD14

------------3-----------------------------------
D2C0637DEA88DE38D215F18D718F6DFE744C7DD2BEDDE534CA971

-------------------------1
D0C668235F9238CBA812A80681531FA39
```

```
AES Engine Keys, using seed 751, and directions:
Engine #1 Key 1,013, Used in Encrypt Orientation =
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Engine #2 Key 959, Used in Encrypt Orientation =
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Engine #3 Key 579, Used in Encrypt Orientation =
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

AES Engines (each) <Input Stream> | <Output Stream>:
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  FC954033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186E00019EE5C8162E

D35C565417FE06381054904F81791EF5336592CD14D2C0637DEA88DE38D215F1 |
  17EACC168043AD4AAB3FC322F0CFD4E79F953348BCE1F1BD79AFA9B32D2A68D6

8D718F6DFE744C7DD2BEDDE534CA971D0C668235F9238CBA812A80681531FA39 |
  5700BDD4BFF0B2EC1C6D955CE71C255676C84C5A566D26A62FDF93C6027A9233

The Pre-Processor and AES seeds, 228 and 751
respectively, along with the 3 AES engine orientation
                                     123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:

--2---6----------------------------------------------
FCE954F033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186E0

-----------1-----------------------------------------
0019EE5C81602E17EACC168043AD4AAB3FC322F0CFD4E79F95334

-------------------------3--------4---5
8BCE1F1BD79AFA9B32D2A68D645700BDD42BFFE0B2EC1C6D955CE

71C255676C84C5A566D26A62FDF93C6027A9233

The Checksum is then calculated and the checksums and
checksum sub-key number (416) are converted to the
            123456789ABCDEFGHIJK
hex string <6A93E276BCE0F4F2D1A0> and inserted in the
data stream:
-------9-----------------------------------------2--7
FCE954FB033A67B860C05B0C8D51DDA9C14CB645F877CBEA2A187

---E---------------------------8C--------------J-D---
6E040019EE5C81602E17EACC168043A60D4AAB3FC322F0CAFFD4E

-------5---B---------------K------1--------6---------
79F9533E48BECE1F1BD79AFA9B302D2A686D645700B2DD42BFFE0

----------G-I------A------F----------H-----------3---
B2EC1C6D95251CE71C2C55676CF84C5A566D2D6A62FDF93C69027

--4
A93233
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                                1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
-----------------------------------------4--------1--
FCE954FB033A67B860C05B0C8D51DDA9C14CB645F4877CBEA21A1

--------3--------------------------------------------
876E0400919EE5C81602E17EACC168043A60D4AAB3FC322F0CAFF

------------------------------------------2
D4E79F9533E48BECE1F1BD79AFA9B302D2A686D645A700B2DD42B

FFE0B2EC1C6D95251CE71C2C55676CF84C5A566D2D6A62FDF93C6

9027A93233
```

Vernam Post-Processor Input Data, keys and Output:

```
Data  from   AES: FCE954FB033A67B860C05B0C8D51DDA9C14
Segment @  6,197: 028BF3E11432A31FAEA9B6C27391969CBC7
Segment @ 96,182: 0B5DA68565399761040477395 3D51570BAB
Seg Calc@ 11,292: C9E633B88CE0D0D0DDE81A77A28E390BC38
Post-Data Output: 3CD93227FED18316178580800F9B674E040

CB645F4877CBEA21A1876E0400919EE5C81602E17EACC168043A6
3F99898B1084D11BCC1D2B18899D5453D66E7E1307D1A241117CE
9552A5D9D8354076DE7E4617E9A62682997A71EB915E47F8814E7
AF46FE10B355F1C9388F3AAAAE8BA7A5C99C411456627E454AFA4
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

0D4AAB3FC322F0CAFFD4E79F9533E48BECE1F1BD79AFA9B302D2A
BE66FAC82F385584D10BEBCD1814748A19A740DB49EF1D6487E17
B664369A9138D1CD33E84235087C84B4FE8101DAF3ACA8AC69C13
ACF1C2281C225279EA256EF4510C42799F7BD7212F35D082B2B97
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

686D645A700B2DD42BFFE0B2EC1C6D95251CE71C2C55676CF84C5
BE95D73CE1C830934B2A52F58BC678E9635A5195FD7FF88DF3495
FA581B3FBFB270D4E238A900097CF1E81D706D67DAE5D81B2B693
66CE7181DDBCD2EE2B6202BDF45AC62DF40E77BDC1B203E747837
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

A566D2D6A62FDF93C69027A93233
07DE010F2FE4B5E135A153CD618C
2DFE1EB017BDABF3CE51FD0D9602
DD630BE8F00BC2C15845B607B8B2
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□iÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.

Example 5

In Example 6a starting on the next page, the ‘chainTbl’ structure, used in the pseudo selection process, is the only change. A representation of this key is in Appendix D. In order to illustrate the technology, it was designed to affect only the pseudo process for the 3 AES key numbers. In the demonstration system only, that process uses a second structure that is normally and totally equal to the first structure used for the Pre and Post-Processors; in a real system, all engines would use the same chain table structure. Only the ‘chainTbl’ structure used for the AES pseudo selection process is changed. The process that recreates the table also moves the AES keys, <u>not changing them</u>, to match the change to the created chain key table. Since there is no change in the ciphertext between the two examples due to the AES keys not changing (only the internal number pointing to the AES key structure changed), <u>there is no mathematical information that could be extracted to yield the makeup of the ‘chainTbl’ structure</u> because the chain table did not direct the physical conversion of the input to output text. Any attack on any system relies on ciphertext changes to yield the makeup of the table; if there are no changes, the wrong assumption can easily be made that there were no key changes. Considering the number of possible ‘chainTbl’ structures, it is the author’s opinion that this alone would be enough to stop an attacker from even beginning an attack on this MOO. Isn’t the goal of any cryptographic design to try to at least begin to convince even the most hardened attacker that their efforts would never be fruitful?

After The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
63205379737465 6D20456E6372797074696F6E2053656374696F
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D6265722075736167652E1E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 573, Used in Encrypt Orientation =
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Engine #2 Key 747, Used in Encrypt Orientation =
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Engine #3 Key 613, Used in Encrypt Orientation =
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

AES Engines (each) <Input Stream> | <Output Stream>:

```
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  FC954033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186E00019EE5C8162E

D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D38D215F18D7 |
  4C4EB5E4F9F05CCC8A147513C1BFCD9E6A2352F90A09590BC29BD79B76C431D6

18F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA812A38068151FA39 |
  ECBB71741D5E423AA3D02C1B59BE47374F911D2433B10C70FFA577E345386055
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation bits, are converted to a hex string <0E42EF> and inserted in the data stream:

```
                                                  123456
bits, are converted to a hex string <0E42EF> and
```

```
-------------------------------1---------------4-6---
FC954033A67B860C05B0C8D51DDA9C104CB645F877CBEA221F86E

------25-----------------------------3
00019EEEE5C8162E4C4EB5E4F9F05CCC8A1474513C1BFCD9E6A23

52F90A09590BC29BD79B76C431D6ECBB71741D5E423AA3D02C1B5

9BE47374F911D2433B10C70FFA577E345386055
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the hex string <6DAAB706CD5DA34071A0> and inserted in the data stream:

```
                  123456789ABCDEFGHIJK
hex string <6DAAB706CD5DA34071A0> and inserted in the
```

```
--------6------------G---------------K--------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

---------------E---2-H----------------------7--------
86E00019EEEE5C83162DE74C4EB5E4F9F05CCC8A14740513C1BFC

9---C---J------------3-------8-------------A---1----5
CD9ED6A2A352F90A09590ABC29BD769B76C431D6ECBDB716741DB

---------D--------------4---B---F----I
5E423AA3DA02C1B59BE47374AF9151D24433B110C70FFA577E345

386055
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                               1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
-----------------------------------------------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

-------------------------------4-------------------1-
86E00019EEEE5C83162DE74C4EB5E4F49F05CCC8A14740513C11B

----------------3------------------------------------
FCCD9ED6A2A352F990A09590ABC29BD769B76C431D6ECBDB71674

-------2
1DB5E42A3AA3DA02C1B59BE47374AF9151D24433B110C70FFA577

E345386055
```

Vernam Post-Processor Input Data, keys and Output:

```
Data  from   AES: FC9540337A67B860C05B00C8D51DDA9C104
Segment @  6,197: 028BF3E11432A31FAEA9B6C27391969CBC7
Segment @ 96,182: 0B5DA6856539976104047739 53D51570BAB
Seg Calc@ 11,292: C99A2770F5BD0F087D7341B3FAC23E3E128
Post-Data Output: 3CD93227FED18316178580800F9B674E040

CB6045F877CBEA221F86E00019EEEE5C83162DE74C4EB5E4F49F0
3F99898B1084D11BCC1D2B18899D5453D66E7E1307D1A241117CE
9552A5D9D8354076DE7E4617E9A62682997A71EB915E47F8814E7
AF42E4A0B355F1CA868EB4AEB7F4D71C829C6E1264800AC9BA5F2
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

5CCC8A14740513C11BFCCD9ED6A2A352F990A09590ABC29BD769B
BE66FAC82F385584D10BEBCD1814748A19A740DB49EF1D6487E17
B664369A9138D1CD33E84235087C84B4FE8101DAF3ACA8AC69C13
FD77E303AB05B1720E0D44F5129D05A08A0A8609C631BBAA67026
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

76C431D6ECBDB716741DB5E42A3AA3DA02C1B59BE47374AF9151D
BE95D73CE1C830934B2A52F58BC678E9635A5195FD7FF88DF3495
FA581B3FBFB270D4E238A900097CF1E81D706D67DAE5D81B2B693
7867240D410A482C748057EB327C0862D3D3253A099410242E9EF
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

24433B110C70FFA577E345386055
07DE010F2FE4B5E135A153CD618C
2DFE1EB017BDABF3CE51FD0D9602
5C46E22F5A54E2F7E936D496EAD4
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.

Example 6a

In Example 6b beginning on the next page, the chain table for all engines was changed and, unlike the previous example, the positions of the keys, as well as the seeds and key numbers, were <u>not</u> changed. So any keys displayed were in memory at the time and not remade. This will result in the normal differences as if new seeds and key numbers had been selected even though the seeds were not changed. As before, the last segment in the Post-Processor will be calculated so as to not cause a change in the ciphertext output. This will illustrate the purpose for the chain table, and present an example of still another consideration for any attacker to give up on attacking this MOO.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data and Output Stream:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 5468697320697320612074657374206F66
Segment @ 74,574: CDA30270F2BA69728960778CB1C8E0B25C
Segment @ 24,165: 6F9F8A08BD3F482C73F3E568C338B9475A
Segment @ 72,312: DE22896F570153205FDD1EE1F4D168C926
Prep Data Output: 2876686438ED015EC46EF860F555115346

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
ACC4925341D7257A6ABF84562CBDFECC6290672BB196B36DBF35
14801F88A9A544A10AC07BA34D5DB218E2FFEE3E236484662B0B
BA6FAFC146F5E02C4E41139166669E3FB2C7E94CFBDB247E022E
225F4A7F8ED3E8834F5085116AA691994BD814360E5B7205FE79

 c   S y s t e m   E n c r y p t i o n   S e c t i o
63205379737465 6D20456E6372797074696F6E2053656374696F
C3BAA0797E2E91FE24DCE911DF134C129BE0221D18D448E93E07
8C9AECF493827CA79089B0771A3BEC08D7F2390F38B198596350
356C13A8F4E0DE129B8083D678ADEC06E5AC5FA07A0454118126
196C0C5C6A3856260F90B4D3CFFC3C68C0D12A920904E7D5B51E

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D62657220757361676552E1E4141
05EFB05C3FCFA307EEF0F14DD268698B611B6A7A0426B0B182
970F71D92138DA339A7F2DFD2C3705BD5B9C383AE05B5126B4
83F7AB53422B01BB0EFF1CE267C49948E290C849793EA44225
7F3738B732B817E25A3EB53FFBFE875EAD64FB6EF86D5B9452
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
         1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
2876686438ED015EC46EF860F555115346225F4A7F8ED3E8834F5

-----------------------------------------------------
085116AA691994BD814360E5B7205FE79196C0C5C6A3856260F90

-----------------------------------------------3-----
B4D3CFFC3C68C0D12A920904E7D5B51E7F3738B732B817EE25A3E

--2-4----------------1
B5135FFBFE875EAD64FB63EF86D5B9452
```

Author’s Note: Compare this Overhead Placement display with the one in Example 1. You will see the same data, ‘31E5’ being placed in the same positions in an almost entirely different data stream. You can do the same with the other placement displays.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 671, Used in Encrypt Orientation =
6ED8EF9AF9276B297E4355F1D37A544CFD88C17C485D5E8E97214584F8761EB2

Engine #2 Key 242, Used in Encrypt Orientation =
9E8FF01DD94F9AA6F036A29258B2515CB8A0A163FB0EF3D7A34500D9600712DE

Engine #3 Key 280, Used in Encrypt Orientation =
5893AD7199B4DA752D5FE95E73C35683FE7C66EA456F73539CF58479623B2094

AES Engines (each) <Input Stream> | <Output Stream>:

```
2876686438ED015EC46EF860F555115346225F4A7F8ED3E8834F5085116AA691 |
  3403636982F4912991E4E6251FAB95FE3B2BFBE755BF8B924F92638274E9B065

994BD814360E5B7205FE79196C0C5C6A3856260F90B4D3CFFC3C68C0D12A9209 |
  5A5F8C57FFF299F3D09649F14F1BE4E9C7A9DAA6E7C944ECE2F91EA6049A57D6

04E7D5B51E7F3738B732B817EE25A3EB5135FFBFE875EAD64FB63EF86D5B9452 |
  32EB1DCD583868D910D18865D3A9AD5E2741089456729FCB9F4C848BFCDF73BB
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation

```
                                      123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
3403636982F4912991E4E6251FAB95F0E3B2BFBE755BF8B29F24F

------25-----------------------------3
926382EE74E9B0655A5F8C57FFF299F3D096449F14F1BE4E9C7A9

DAA6E7C944ECE2F91EA6049A57D632EB1DCD583868D910D18865D

3A9AD5E2741089456729FCB9F4C848BFCDF73BB
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the

```
            123456789ABCDEFGHIJK
hex string <6D9B90D6CC63DEC691A0> and inserted in the
data stream:
```

```
--------6------------G----------------K--------------
34036369082F4912991E46E6251FAB95F0E3B20BFBE755BF8B29F

---------------E---2-H----------------------7--------
24F926382EE74E9EB06D595A5F8C57FFF299F3D09644D9F14F1BE

9---C---J------------3-------8-------------A---1----5
C4E93C7AA9DAA6E7C944E9CE2F91E6A6049A57D632ECB1D6CD589

---------D--------------4---B---F----I
3868D910DD18865D3A9AD5E2B7416089C4567129FCB9F4C848BFC

DF73BB
```

```
Using the hex digits on either end of the data stream
plus 2 bits from the Post-Processor seed, assembled
as the new Overhead Key Number, the lower 4 bits of
the current Post-Processor seed (4) plus the Overhead
key number (425), used before this point, are
                             1234
converted to the hex digits <1A94> and inserted in
the data stream:

-----------------------------------------------------
34036369082F4912991E46E6251FAB95F0E3B20BFBE755BF8B29F

-------1---------------------------------------------
24F9263182EE74E9EB06D595A5F8C57FFF299F3D09644D9F14F1B

------3----------------------------------------------
EC4E939C7AA9DAA6E7C944E9CE2F91E6A6049A57D632ECB1D6CD5

-------------4-----2
893868D910DD148865DA3A9AD5E2B7416089C4567129FCB9F4C84

8BFCDF73BB

Vernam Post-Processor Input Data and Output Stream:

Data  from   AES: 34036369082F4912991E46E6251FAB95F0E
Segment @  5,198: B290535FEF1455F4A2E853CCDE211E4816B
Segment @110,674: FD6FACEC376230844110FED55FC20993C7E
Seg Calc@ 49,473: 4725AEFD2E88AF746D636B7FAB67DB0025B
Post-Data Output: 3CD93227FED18316178580800F9B674E040

3B20BFBE755BF8B29F24F9263182EE74E9EB06D595A5F8C57FFF2
AD6D68C37FFB80C89D807D932E8F7E8A9D95C4C00896DF2C45FC9
1E5934A0B8881376652D17D354CF80FA8ADDB8DF9A38820B45024
46FD6ED7BE07E189ECE2AAC785E35B95B03D36C7B94AFF76A1F34
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

99F3D09644D9F14F1BEC4E939C7AA9DAA6E7C944E9CE2F91E6A60
8E25DBB7D55771B2ED37982521321E73DA63BE3ACBFBF4D96BE61
A55612240F1A07C00610D0A5D46F96EB822C4B463ECC33800076A
1B39BC40FF94A1C707D92680BD70778E6A145BA5F0202431D37D2
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

49A57D632ECB1D6CD5893868D910DD148865DA3A9AD5E2B741608
02FA8AB8AA6DA949571F510F7110455171ED2495744EAA3A58CC3
2276D4C0782FCFB29CA5791B197048FE9994440C3883C491D5E52
2347FAC30F44C4EAB7BC09862B8CF202CF2416F01C65C801ABA6D
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

9C4567129FCB9F4C848BFCDF73BB
4E4DDD340E12C22E657C6DA9B873
8043C3DA02B6C9F27F2BD9AAD9D7
006EBF7DFD1297D0FBF977B26F10
5225C6816E7D034065253F6E7D0F
```

```
After swapping the first digit pair with pair #61 and
the last digit pair with pair #64, each of the
resulting 2-digit hex pairs is then converted to an
ASCII character and would normally be written to the
ciphertext output file.  The ciphertext block:
```

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

```
The following character conversions were executed to
allow a single line, 1-character-per-position,
display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.
```

Example 6b

Example 6c beginning on the next page is just another randomization of the chain table, all other factors remaining the same as the example in 6b above. Notice, again, that all seeds and random numbers remain the same, and the positions of all the Overhead Digits also remain the same. The Overhead data being inserted, except for the checksum data, also remains the same.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor = 228, 93, AES = 751, Post-Processor = 948, 92, Checksum Sub-Key = 416, Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data and Output Stream:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 5468697320697320612074657374206F66
Segment @ 78,459: AF678452A5E52B36ABFBB1901D666CAA2B
Segment @ 48,138: C053F3BB5DB94A56E84371D7C1DB73ACF1
Segment @ 35,636: FA286BE98B846DE6CA633E6C9147263306
Prep Data Output: C174757353B17FA6E8FB8A4E3E8E195ABA

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
672A6D2DC6238236172BB6EB3C2CA3962BD48D76F18B8A59F7A8
802BD5C6510698BEE70A76A7E8175D03D96D1FB1346F23CABBA7
CD23CCC23C31DB325AC8803E83A71629FCD020B984D380972C50
0A561C4C8B40A8CECB8729073ABCABCE7719C611264548740836

 c   S y s t e m   E n c r y p t i o n   S e c t i o
63205379737465 6D20456E6372797074696F6E2053656374696F
E66D85E4C554840F0A768801789B0DD417B90150DA0A83A77E62
5BCA0B0975F0747C56E70896C4A5F2A762940AFAE2EFA6577670
EAF869B8997D68534158A18744B1AF5728485F2A495B06CECBCA
347FB42C5AADFD4D3D8C4F738AF62050340A3AA022DB404AAAB7

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D62657220757361676 52E1E4141
EB63B5C00A9C8423B36EEA9F7AAAC2AFBBCCEACDDE2EA540DA
B2CC872803AEAAAEE8DAD07B0240BD9556A3BB6BA19A8F01EC
18BCD379F79DF7F6090EF25B0DA19D76AF241A285A25530B4D
2F33B3F090CBB61672F4BDD2172E906C37382AE940BF670B3A

The next ciphertext block's randomly selected table
and offset seeds, 798, 69 are converted to the hex
        1234
string <31E5> and inserted in the data stream:

-----------------------------------------------------
C174757353B17FA6E8FB8A4E3E8E195ABA0A561C4C8B40A8CECB8

-----------------------------------------------------
729073ABCABCE7719C611264548740836347FB42C5AADFD4D3D8C

----------------------------------------------3------
4F738AF62050340A3AA022DB404AAAB72F33B3F090CBB6E1672F4

--2-4----------------1
BD1D52172E906C37382AE3940BF670B3A
```

Author’s Note: Compare this Overhead Placement display with the one in the previous example (6b). The data being placed (31E5) is the same and the locations are the same.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 706, Used in Encrypt Orientation =
A84FC93CAFA2D44C2C6A0273EBCF29372AAB0D65583F3C5E1C2436B29984D1F0

Engine #2 Key 638, Used in Encrypt Orientation =
28E90195708BECA1261C5E1A92940052CC7ACEA6D2A7CC5ED84DF5D4DF3775D9

Engine #3 Key 687, Used in Encrypt Orientation =
4CDC09D89454E99C5A0FD6BC8A124F07C45207B86A779555F6E06DD892E70F0D

AES Engines (each) <Input Stream> | <Output Stream>:

```
C174757353B17FA6E8FB8A4E3E8E195ABA0A561C4C8B40A8CECB8729073ABCAB |
  9E5B953119565672D1BB2F5FCF42281223CCC36B75D8457091635F4EA4E045DB

CE7719C611264548740836347FB42C5AADFD4D3D8C4F738AF62050340A3AA022 |
  1A794C72F09BD415116E5D8D8F8433D9EF029656A9FBCA6D437EFABB03681E44

DB404AAAB72F33B3F090CBB6E1672F4BD1D52172E906C37382AE3940BF670B3A |
  F9F405CE0E63C1500D456853A9F38B19205432E4AFAA7D0434A5CE75238FBACA
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation

```
                                           123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
9E5B953119565672D1BB2F5FCF422810223CCC36B75D84527F091

------25-----------------------------3

635F4EEEA4E045DB1A794C72F09BD415116E54D8D8F8433D9EF02

9656A9FBCA6D437EFABB03681E44F9F405CE0E63C1500D456853A

9F38B19205432E4AFAA7D0434A5CE75238FBACA
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the

```
            123456789ABCDEFGHIJK
hex string <6AB1DD16BE870208E1A0> and inserted in the
data stream:
```

```
--------6-------------G---------------K--------------
9E5B9531D19565672D1BB82F5FCF422810223C0CC36B75D84527F

--------------E---2-H-----------------------7--------
091635F4EEEA4E0245DABE1A794C72F09BD415116E541D8D8F843

9---C---J------------3-------8-------------A---1----5
B3D97EF0A29656A9FBCA6BD437EFA6BB03681E44F9FE4056CE0ED

---------D--------------4---B---F----I
63C1500D4056853A9F38B1921054832E04AFA1A7D0434A5CE7523

8FBACA
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                                1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
------------------------------------3-------------2--
9E5B9531D19565672D1BB82F5FCF4228102293C0CC36B75D84A52

-----------------------------------------------------
7F091635F4EEEA4E0245DABE1A794C72F09BD415116E541D8D8F8

------4----------------------------------------------
43B3D947EF0A29656A9FBCA6BD437EFA6BB03681E44F9FE4056CE

--------------------1
0ED63C1500D4056853A91F38B1921054832E04AFA1A7D0434A5CE

75238FBACA
```

Vernam Post-Processor Input Data and Output Stream:

```
Data  from   AES: 9E5B9531D19565672D1BB82F5FCF4228102
Segment @ 97,106: 837F6498049B7092FA4BEEB118C15657C32
Segment @ 97,803: 66C0D0F28AF972685748AB8CBBED8AC4D3A
Seg Calc@ 58,942: 473D137CA126E48B979D7D92F378F9F504A
Post-Data Output: 3CD93227FED18316178580800F9B674E040

293C0CC36B75D84A527F091635F4EEEA4E0245DABE1A794C72F09
5438A7E3782B0A53569975FFF6AB8CE79E7BD883148404B414BD6
2B4B10E2BA9618F2E48C7D44596689652BB56F0124D62C2F5D28E
98A636C8A5E7406E6B01380C5418A0F9B552BE5530090B43E597A
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

BD415116E541D8D8F843B3D947EF0A29656A9FBCA6BD437EFA6BB
EC779C4C91B4A6302092A894F74A0B0EDED4F3033A4F142683A4C
582E7FC1470646BE9875105729F347DE4F8C98DDC033F707B158E
A0A117DE52F31EACB7B62B894D011035608E93FFB0186CA696DC0
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

03681E44F9FE4056CE0ED63C1500D4056853A91F38B1921054832
73BF8665D2191FDB30F3504061E2DEF0908D8F1352FA11D61E02F
36E3BBCEBF5038571091A8DAE8C2200F612B2263E50FC7DDBD1AD
0C5AFA37677AD8A747E3375C06DC084336CDA83C4539000690744
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

E04AFA1A7D0434A5CE75238FBACA
CB92DADB3D140A6782C999ED9BC2
0B8CEB15F2BBF2884500AB041AE7
72710D55DCD6CF0A6C992E0846E0
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□ìÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.

Example 6c

In Example 7 beginning on the next page, a string of just randomly created characters was set to be the ciphertext block. With it set to use all of the original keys used in Example 1, the demonstration system was set to modify just segment 3 of the Post-Processor.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor 228, 93, AES 751, Post-Processor 948, 92, Checksum Sub-Key 416 and Overhead Placement Sub-Key 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 5468697320697320612074657374206F66
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0842EB72EDDBEDD04EB9

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
F8791EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D62657220757361676521E4141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
C7DD2BEDDE534CA971D0C668235F9238CBA812A8068151FA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB

-----------------------------------------------------
5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2

----------------------------------------------3------
C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0

--2-4----------------1
C61658235F9238CBA812A38068151FA39
```

AES Engine Keys, using seed 751, and directions:
Engine #1 Key 1,013, Used in Encrypt Orientation =
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Engine #2 Key 959, Used in Encrypt Orientation =
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Engine #3 Key 579, Used in Encrypt Orientation =
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

AES Engines (each) <Input Stream> | <Output Stream>:

```
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A76635 |
  FC954033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186E00019EE5C8162E

D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D38D215F18D7 |
  4C4EB5E4F9F05CCC8A147513C1BFCD9E6A2352F90A09590BC29BD79B76C431D6

18F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA812A38068151FA39 |
  ECBB71741D5E423AA3D02C1B59BE47374F911D2433B10C70FFA577E345386055
```

The Pre-Processor and AES seeds, 228 and 751 respectively, along with the 3 AES engine orientation

```
                                                  123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
FC954033A67B860C05B0C8D51DDA9C104CB645F877CBEA221F86E

------25-----------------------------3
00019EEEE5C8162E4C4EB5E4F9F05CCC8A1474513C1BFCD9E6A23

52F90A09590BC29BD79B76C431D6ECBB71741D5E423AA3D02C1B5

9BE47374F911D2433B10C70FFA577E345386055
```

The Checksum is then calculated and the checksums and checksum sub-key number (416) are converted to the

```
                     123456789ABCDEFGHIJK
hex string <6DAAB706CD5DA34071A0> and inserted in the
data stream:
```

```
--------6------------G----------------K--------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

---------------E---2-H----------------------7--------
86E00019EEEE5C83162DE74C4EB5E4F9F05CCC8A14740513C1BFC

9---C---J------------3-------8-------------A---1----5
CD9ED6A2A352F90A09590ABC29BD769B76C431D6ECBDB716741DB

---------D--------------4---B---F----I
5E423AA3DA02C1B59BE47374AF9151D24433B110C70FFA577E345

386055
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                              1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
-----------------------------------------------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F

-------------------------------4-------------------1-
86E00019EEEE5C83162DE74C4EB5E4F49F05CCC8A14740513C11B

----------------3------------------------------------
FCCD9ED6A2A352F990A09590ABC29BD769B76C431D6ECBDB71674

-------2
1DB5E42A3AA3DA02C1B59BE47374AF9151D24433B110C70FFA577

E345386055
```

Vernam Post-Processor Input Data, keys and Output:

```
Data   From  AES: FC9540337A67B860C05B00C8D51DDA9C104
Segment @  6,197: 028BF3E11432A31FAEA9B6C27391969CBC7
Segment @ 96,182: 0B5DA685653997610404773953D51570BAB
Seg Calc@ 11,292: ADBEF65DF20BB8C9BE068332FBA636A75B6
Post Data Output: 58FDE30AF96734D7D4F042010EFF6FD74DE

CB6045F877CBEA221F86E00019EEEE5C83162DE74C4EB5E4F49F0
3F99898B1084D11BCC1D2B18899D5453D66E7E1307D1A241117CE
9552A5D9D8354076DE7E4617E9A62682997A71EB915E47F8814E7
8474F2B4443C94B7A78AB3560BFBE448703DED0B1D04F3C50C5B5
E5DF9B1EFB46EFF8AA6F3E59722E78C5BC3FCF14C7C5A39868F6C

5CCC8A14740513C11BFCCD9ED6A2A352F990A09590ABC29BD769B
BE66FAC82F385584D10BEBCD1814748A19A740DB49EF1D6487E17
B664369A9138D1CD33E84235087C84B4FE8101DAF3ACA8AC69C13
658888011597F00AFCDBE81496C9F7B3A61C7B640C2AA2EBF2784
3146CE47DF92678205C48C725003A4DFB8AA9AF026C2D5B8CB31B

76C431D6ECBDB716741DB5E42A3AA3DA02C1B59BE47374AF9151D
BE95D73CE1C830934B2A52F58BC678E9635A5195FD7FF88DF3495
FA581B3FBFB270D4E238A900097CF1E81D706D67DAE5D81B2B693
9B37325C6B4525B284928A289682D28BB3513A6C86229F08DA6F2
A93ECF89D982D2E3599DC4393E02F850CFBAB30545CBCB31931E9

24433B110C70FFA577E345386055
07DE010F2FE4B5E135A153CD618C
2DFE1EB017BDABF3CE51FD0D9602
57D11EC369312BBC44D3F18017CE
59B23A6D5D18CA0BC8C01A788015
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
¸ýã□ùg4×ÔðB□ÿo×Mî]ù±ï´nÿŠ¦óå—"çŒ[ÃüñL|Z9†□l1FÎGß′g,□Ä
ŒrP□¤ßXªš□&ÂÕ¸Ë1°‟ìø□˜-.5™ÜC‟à/…□û«0T\¼³□1éY²:m]□ÊÈÀ□
x€ð
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

```
Character 4, ASCII 10, to character '□', ASCII 2.
Character 87, ASCII 12, to character '□', ASCII 2.
```

Example 7

In the 4-page Example 8 starting on the next page, the ‘plaintext input’ is now varied. All keys except for the last segment in the Post-Processor, as well as maintaining the same ciphertext output, are held identical to Examples 1 through 3.

What this illustrates is that it will be easy for any attacker to create a key for any legitimate plaintext for any single given ciphertext. With other cipher engines and MOO’s, if they were handed this task, it is most likely not possible.

When combined with the previous examples that illustrate the ease in creating multiple keys for any plaintext/ciphertext pair, it would hopefully provide solid evidence to the attacker that even beginning to attempt to attack this proposed MOO will never be worth their while.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor 228, 93, AES 751, Post-Processor 948, 92, Checksum Sub-Key 416 and Overhead Placement Sub-Key = 425

Vernam Pre-Processor Input Data, keys and Output:

```
Text  Characters:  T h i s   i s   t h e   f i r s t
ASCII Input Data: 54686973206973207468652066697273 74
Segment @112,536: ED873B05A3C59DD0B18EA8B81F18CF360D
Segment @ 84,239: 7E33E29E80498E0E141DA45CB9C3327084
Segment @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6756
Prep Data Output: 932637BCBD9F6C0857A363A8CEF08252AB

   u s e   o f   N o n - P s e u d o   R a n d o m
20757365206F66204E6F6E2D50736575646F2052616E646F6D20
554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C
8D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE3401
0241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0
FAEAF3B64A15A2C8F4B4C1719A3513D4417A1542E67F841A4C4D

 N u m b e r s   d u r i n g   e n c r y p t i o n
4E756D626572732064757269696E6720656E6372797074696F6E20
2ADCA5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738
5E6A8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8
EFEF655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DB
D52C20EE2563848050E2DC6961F4D8C28A2D4341F409FCC4E00B

 f o r   A N Y   c r y p t o s y s t e m . - A A A
666F7220414E592063727970746F73797374656D2E1E414141
ACF88D139CDEF4D2C47C1B980831B7D270F7AA6C8938844408
CA01D65B712C36D763688FE5334838492554AC679EAD9807DA
CF0422C45DC5E1C1F68A27787A436F83EB7875C4743A53F8AA
CF920BACF1797AE432ECCA7535559361CDAF16A24DB10EFA39
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
932637BCBD9F6C0857A363A8CEF08252ABFAEAF3B64A15A2C8F4B

-----------------------------------------------------
4C1719A3513D4417A1542E67F841A4C4DD52C20EE2563848050E2

----------------------------------------------3------
DC6961F4D8C28A2D4341F409FCC4E00BCF920BACF1797AEE432EC

--2-4----------------1
CA175535559361CDAF16A324DB10EFA39
```

AES Engine Keys, using seed 751, and directions:
Engine #1 Key 1,013, Used in Encrypt Orientation =
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Engine #2 Key 959, Used in Encrypt Orientation =
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Engine #3 Key 579, Used in Encrypt Orientation =
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

AES Engines (each) <Input Stream> | <Output Stream>:

```
932637BCBD9F6C0857A363A8CEF08252ABFAEAF3B64A15A2C8F4B4C1719A3513 |
  EF36FC9957BA7037AA23D46C952A0F887860EB7B516FEB5DD0DB5726AC1372E3

D4417A1542E67F841A4C4DD52C20EE2563848050E2DC6961F4D8C28A2D4341F4 |
  8505E49CDCFB1876B6F96946BBB643BF1A7159BF0AC4FADAE31316499D5E4B38

09FCC4E00BCF920BACF1797AEE432ECCA175535559361CDAF16A324DB10EFA39 |
  A90B66462DB42771417BE8DE5BEEA84327B1D48200CAAE616A134DC714D912B3
```

```
The Pre-Processor and AES seeds, 228 and 751
respectively, along with the 3 AES engine orientation
                                     123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
EF36FC9957BA7037AA23D46C952A0F8087860EB7B516FEB25FDD0

------25-----------------------------3
DB5726EEAC1372E38505E49CDCFB1876B6F964946BBB643BF1A71

59BF0AC4FADAE31316499D5E4B38A90B66462DB42771417BE8DE5

BEEA84327B1D48200CAAE616A134DC714D912B3
```

```
The Checksum is then calculated and the checksums and
checksum sub-key number (416) are converted to the
            123456789ABCDEFGHIJK
hex string <6A1B7826B469518EC1A0> and inserted in the
data stream:
```

```
--------6------------G----------------K--------------
EF36FC99857BA7037AA23ED46C952A0F80878600EB7B516FEB25F

---------------E---2-H----------------------7--------
DD0DB5726EEAC13172EA3C8505E49CDCFB1876B6F9642946BBB64

9---C---J------------3-------8-------------A---1----5
B3BF91A7A159BF0AC4FAD1AE313166499D5E4B38A904B666462D7

---------D--------------4---B---F----I
B427714175BE8DE5BEEA8432B7B16D488200C1AAE616A134DC714

D912B3
```

```
Using the hex digits on either end of the data stream
plus 2 bits from the Post-Processor seed, assembled
as the new Overhead Key Number, the lower 4 bits of
the current Post-Processor seed (4) plus the Overhead
key number (425), used before this point, are
                             1234
converted to the hex digits <1A94> and inserted in
the data stream:

-------------------------------------------2------
EF36FC99857BA7037AA23ED46C952A0F80878600EB7B5A16FEB25

---------------3----------------------------------
FDD0DB5726EEAC139172EA3C8505E49CDCFB1876B6F9642946BBB

----------------4-------------------------------1
64B3BF91A7A159BF04AC4FAD1AE313166499D5E4B38A904B66614

62D7B427714175BE8DE5BEEA8432B7B16D488200C1AAE616A134D

C714D912B3

Vernam Post-Processor Input Data, keys and Output:

Data  from   AES: EF36FC99857BA7037AA23ED46C952A0F808
Segment @  6,197: 028BF3E11432A31FAEA9B6C27391969CBC7
Segment @ 96,182: 0B5DA6856539976104047739 53D51570BAB
Seg Calc@ 11,292: DA399BDA0AA1106BC78A7FAF434ACEAD824
Post-Data Output: 3CD93227FED18316178580800F9B674E040

78600EB7B5A16FEB25FDD0DB5726EEAC139172EA3C8505E49CDCF
3F99898B1084D11BCC1D2B18899D5453D66E7E1307D1A241117CE
9552A5D9D8354076DE7E4617E9A62682997A71EB915E47F8814E7
1C42AFEF713F7403BCF58475F93CD7EC121B311F144BBAC9D21CD
CEE98D0A0C2F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2B

B1876B6F9642946BBB64B3BF91A7A159BF04AC4FAD1AE31316649
BE66FAC82F385584D10BEBCD1814748A19A740DB49EF1D6487E17
B664369A9138D1CD33E84235087C84B4FE8101DAF3ACA8AC69C13
103C0278494236D8AE953AD4559807ABCC9E8AD3FB809A22A60F4
A9B9A545610026FAF7122093D45756CC94BC679DECD9CCF95E4B9

9D5E4B38A904B6661462D7B427714175BE8DE5BEEA8432B7B16D4
BE95D73CE1C830934B2A52F58BC678E9635A5195FD7FF88DF3495
FA581B3FBFB270D4E238A900097CF1E81D706D67DAE5D81B2B693
93FD5EE304B3495C14FF35BB3F37EACD6F9F751F0763563C0EA26
4A6ED9D8F3CDBF7DA98F19FA9AFC22B9AF38AC53CA7D441D67EF4

88200C1AAE616A134DC714D912B3
07DE010F2FE4B5E135A153CD618C
2DFE1EB017BDABF3CE51FD0D9602
F025D524F8457741D31285779832
5225C6816E7D034065253F6E7D0F
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
”Ù2'þÑƒ□□…€€□›gN□□î˜Ð Âø¨X¶³š□â□¹□éäÀÛä□©Mï+©¹¥Ea□&ú÷
□ “ÔWVÌ<¼g□iÙÌù^K”¦í□□<Û÷Ú˜ñŸ©¯Â+šóŠÅ<§ÔAÖ~ôR%Æ□n}□@e
%?n}□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 18, ASCII 12, to character '□', ASCII 2.
Character 50, ASCII 0, to character '□', ASCII 2.

Example 8

In the next section, decryption will be explained and illustrated. Since even the legitimate recipient’s system has no idea what random numbers were selected to encrypt the first block of text, a ‘search’ algorithm is executed and will always succeed as long as the block is not corrupted or damaged in any way.

In Example 9a on the next 4 pages, the ‘acid test’ of this design is demonstrated. The input to the Post-Processor is taken and only 50% (55) of the characters are changed and that becomes the new ciphertext output. All Pre-Processor segments, all AES and all Post-Processor segments are then randomly recreated (no ‘calculated’ values in any of the Post-Processor segments). Normally, as you well know, if there was only one segment in this version of the Vernam, 55 of the values in this key would be zero since that many characters are identical in order to obtain the expected ciphertext output. Very easily, this would not classify as being a possibly random key. But because there are three segments, they are still free to be completely random, and are random as illustrated.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor 228, 93, AES 751, Post-Processor 948, 92, Checksum Sub-Key 416 and Overhead Placement Sub-Key 425.

Vernam Pre-Processor Input Data and Output Stream:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
ASCII Input Data: 546869732069732061207465737420 6F66
ReRandom@112,536: 2217233CD3C0AAE2524DC6445A23DC8E71
ReRandom@ 84,239: C2A776FD8C3A006C2F761FF173363DE707
ReRandom@ 48,603: 60C64E3697E2E4FB5001C44FB3241F9D77
Prep Data Output: D41E7284E8713D554C1A699FE945DE9B67

   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
6D2620B1F79F7BB48D13268415441A860DA80B7F4B20C5BF30D7
6252E31C8890F41EE4362DEBC7DC6FB29B0790167CE21124F4C3
7ED2512CE7B78A971862DE2DE1A8A4F2E5024E856D8E80C14BF1
51D2FAE4B8EC6C491029BC375E1092B40ADDA1833D3E352AE78C

 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
8A92E55187B9E98564189F05469292CC3D8CEA1860DCEDD98944
B3AAA80B370BA5E3822CE23D6AA9CC11B3F984021E957DCA89DC
27DCA32649EDB76EBE577D050DFB3074EEC746A7B86056A4DE6A
7DC4BD058A2B9E6578266E5E53B91EDD09DD469D954CA5C3B79D

 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D626572207573616765 2E1E4141
68A26493487A4C0F8C542F8F35D2FEAD72B89781558114E8B9
EFC47C9CFE6BB1287004024DF2D28294B14E4709DF41EDAE59
3AE14198AA782216F46D6766D3A1E61D9E549AD871C0DA061C
D3A70BF6720DB05C28733FC976C4E80428D12B379E2E3D01BD
```

The next ciphertext block's randomly selected table and offset seeds, 798, 69 are converted to the hex

```
        1234
string <31E5> and inserted in the data stream:
```

```
-----------------------------------------------------
D41E7284E8713D554C1A699FE945DE9B6751D2FAE4B8EC6C49102

-----------------------------------------------------
9BC375E1092B40ADDA1833D3E352AE78C7DC4BD058A2B9E657826

----------------------------------------------3------
6E5E53B91EDD09DD469D954CA5C3B79DD3A70BF6720DB0E5C2873

--2-4----------------1
3F1C5976C4E80428D12B3379E2E3D01BD
```

The AES keys specified here were randomly recreated during encryption and temporarily replaced the original keys in the specified locations.

AES Engine Keys, using seed 751, and directions:

Engine #1 Key 1,013, Used in Encrypt Orientation =
```
7A7ACF071D944C8A07B9F412EA911502CE315607F62F6E8806FE5668BE612E72
```

Engine #2 Key 959, Used in Encrypt Orientation =
```
0C6FE8BA5870EC30663C9B519398B070652525546A3D69C3993556830B5F28FE
```

Engine #3 Key 579, Used in Encrypt Orientation =
```
7AA75A79C1EC9838C212D41A538D5A759884B8F7C762FCF9DED58EDCC70CEC03
```

AES Engines (each) <Input Stream> | <Output Stream>:
```
D41E7284E8713D554C1A699FE945DE9B6751D2FAE4B8EC6C491029BC375E1092 |
  D9DF8D85843931FF4F9F24E0E5E4A04208951E7E66C6F97C9BAF9808F2F579E2

B40ADDA1833D3E352AE78C7DC4BD058A2B9E6578266E5E53B91EDD09DD469D95 |
  7228553807D021FB106A529A699141688381B89AF0B231191BBC26C20B71ED0A

4CA5C3B79DD3A70BF6720DB0E5C28733F1C5976C4E80428D12B3379E2E3D01BD |
  DDB1A17ED0089432ADEE23DD7FA7CEB2017DB22A17C3D0087EE9C896A11E1481
```

```
The Pre-Processor and AES seeds, 228 and 751
respectively, along with the 3 AES engine orientation
                                            123456
bits, are converted to a hex string <0E42EF> and
inserted in the data stream:
```

```
-------------------------------1---------------4-6---
D9DF8D85843931FF4F9F24E0E5E4A040208951E7E66C6F927FC9B

------25-----------------------------3
AF9808EEF2F579E27228553807D021FB106A5429A699141688381

B89AF0B231191BBC26C20B71ED0ADDB1A17ED0089432ADEE23DD7

FA7CEB2017DB22A17C3D0087EE9C896A11E1481
```

```
The Checksum is then calculated and the checksums and
checksum sub-key number (416) are converted to the
              123456789ABCDEFGHIJK
hex string <6C4322A6DFD4F93CB1A0> and inserted in the
data stream:
```

```
--------6------------G----------------K--------------
D9DF8D852843931FF4F9FC24E0E5E4A0402089051E7E66C6F927F

---------------E---2-H----------------------7--------
C9BAF9808EEF2F5979EC2B7228553807D021FB106A54A29A69914

9---C---J------------3-------8-------------A---1----5
D1684838A1B89AF0B2311491BBC266C20B71ED0ADDBF1A167ED02

---------D--------------4---B---F----I
089432ADEFE23DD7FA7CEB20317DDB223A17C13D0087EE9C896A1

1E1481
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (4) plus the Overhead key number (425), used before this point, are

```
                                1234
converted to the hex digits <1A94> and inserted in
the data stream:
```

```
--------------2---------------------------------
D9DF8D852843931AFF4F9FC24E0E5E4A0402089051E7E66C6F927

--------3---------------------------------------
FC9BAF98098EEF2F5979EC2B7228553807D021FB106A54A29A699

---------------4--------------------------------
14D1684838A1B89A4F0B2311491BBC266C20B71ED0ADDBF1A167E

-------1
D020894132ADEFE23DD7FA7CEB20317DDB223A17C13D0087EE9C8

96A11E1481
```

Note: Pairs of hex digits that are identical in the input and output streams are highlighted in red. Notice that the 3 hex key numbers between the identical input/output digit pairs all Xor to zero, as they should. All hex digit pairs in all segments were selected randomly, forming these key segments.

Vernam Post-Processor Input Data and Output Stream:

```
Data   From  AES: D9DF8D852843931AFF4F9FC24E0E5E4A040
ReRandom@  6,197: FD2F5215613FC87193F31293C62C0CBBE3B
ReRandom@ 96,182: 4E211DA0C824189FD6EC56FAB7C987D77B2
ReRandom@ 11,292: B3734FB565ACD0EE4FDE5E7F71E58B32989
Post Data Output: D9A28D85E4F4931AF58E85D44E0E5E14040

2089051E7E66C6F927FC9BAF98098EEF2F5979EC2B7228553807D
85394F421A26883316C3EC5E8004331454435FF2DDABEAD3EA06A
5F313D9A65BE80F6F8BD203A32B021061A9E163A3188A9D1D6291
DA05B23C4D2BD8C5EE7ECC64B8E412124DED49C8EC23B30FEC2FB
2084C5FA4CD516F927FC9BAF92598EEF2C6979EC2B72D858E807D

021FB106A54A29A69914D1684838A1B89A4F0B2311491BBC266C2
707E1FFF60FA70F4F8453CF8205BE2307F1B0DB963BE945FD7C5E
378ACA474F76CE9A7755EB0ED89D155992C24FC9B39E375AD111B
D9CED56B2F3AB6231C985C606FC6E15CEDD91B5DE05CA305EFBB1
9C25B1D5A5FC21EB0A9C5AFEDF38B78D9A4F520E21351BBCCF036

0B71ED0ADDBF1A167ED020894132ADEFE23DD7FA7CEB20317DDB2
4BFDD4381F6EE66C841C149C7C027D937FD3F56B56A5451A4A03D
AB2D911DCE057866C9E061576746B72706CCB5E7C29F7C7556709
43486A95D16D18BA4DF855C5CB41E079D91F4088FC4A396F1C7F1
A8E9C2BADDB99CA67ED40087913787224 23DD7FE149B20317DD77

23A17C13D0087EE9C896A11E1481
2636FD5FAA5CEB922BD61581709C
A80F970ECCE9A4318BD72229070B
8D676A9C09B54FA34E0137A87797
20FF7CDEBF087EE92696A11E1481
```

After swapping the first digit pair with pair #61 and the last digit pair with pair #64, each of the resulting 2-digit hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
š¢□…äô"□õŽ…ÔN^□□□□L_¤ÍQo'□É°ù%˜îòÆ—žÂ·-…Ž€}œ%±Õ¥ü!ë□œ
Zþß8·□ÙOR□!5□¼Ï□jŽœ+-Û™Êgí@□y□xr$#Ý□áI²□□Ýw  ÿ|Þ¿□~é&–
¡-□
```

The following character conversions were executed to allow a single line, 1-character-per-position, display:

Character 53, ASCII 10, to character '□', ASCII 2.

Example 9a

In Example 9b on the next 4 pages, <u>100%</u> of the Post-Processor input is set to be the output.

The seed and number selections for this data: 228,93,751,0,425,416,948,92,798,69

Sub-Key Seeds used: Pre-Processor 492, 38, AES 730, Post-Processor 997, 47, Checksum Sub-Key 20 and Overhead Placement Sub-Key 338.

Vernam Pre-Processor Input Data and Output Stream:

```
Text  Characters:  T h i s   i s   a   t e s t   o f
Prep ASCII Input: 5468697320697320612074657374206F66
Segment @ 75,109: 083BF0D44DE4C90D976A506CDABB4BF12A
Segment @ 46,864: 1F631C91C231D158FB21707114E4362DED
Segment @ 97,277: 316D3E6A7E2D49A9E2DEE21B15E62871D8
Prep Data Output: 725DBB5CD19122DCEFB5B663A8CD75C279
```

```
   t h e   T i t a n i u m   C r y p t o g r a p h i
2074686520546974616E69756D2043727970746F677261706869
707EA088531E27FB1F88C1E386335A8B78E4F4FDA320491A36E7
709256F9F642C18914BA234D4DC52618B66EBBE3F121B115600E
F61A13EB49EE776725AC7DC9DA3B0ABC7A31E43630DEAEA1D48D
B3E60B22D1E2090C43E7E81AFEB9FA80D52969BB001F16439D96
```

```
 c   S y s t e m   E n c r y p t i o n   S e c t i o
632053797374656D20456E6372797074696F6E2053656374696F
B2E9EB9F4730444F4841150DBC1742BA57A1CAECA15E042C552A
870192CE7EB00149F3E200D7EC4FEFA5F132B15B1D0860FB7495
937E953654EA860A29BB63C1586FC56162D352CA356C8F3CA316
A0D239C3031A570CBE4A0670F81AD7D7B5CDF1A1DFEDA9029C5D
```

```
 n   R a n d o m   N u m b e r   u s a g e . - A A
6E2052616E646F6D204E756D6265722075736167652E1E4141
29620858138D0B0044B7919B2A2915CDA2DBF681BA2E643BBB
3E0D7729EFF9F621E6B426B01E9EF3E00AAB97D3CEC26C4FAF
1A38BC065AB4D457D729F1D5262B625D3834C8DFD04A281E3B
63779116C8A4461B5564339370F9F650E537C8EAC1883E2B6E
```

The next ciphertext block's randomly selected table and offset seeds, 872, 103 are converted to the hex

```
       1234
string <B687> and inserted in the data stream:
```

```
-----------------------------------------------------

725DBB5CD19122DCEFB5B663A8CD75C279B3E60B22D1E2090C43E

-----------------------------------------------------

7E81AFEB9FA80D52969BB001F16439D96A0D239C3031A570CBE4A

-----------2-------4----1----------------------------

0670F81AD7D67B5CDF17A1DFBEDA9029C5D63779116C8A4461B55

-----------------------3

64339370F9F650E537C8EAC81883E2B6E
```

AES Engine Keys, using seed 927, and directions:

Engine #1 Key 676, Used in Decrypt Orientation =
64CF78B19C8964E26B3A40C4A13B8F1D0215485A54448776B73C831C35AAC010

Engine #2 Key 253, Used in Encrypt Orientation
9971F2C111D3AA42CF17A82CA906A0311590D6549C6C33AB8BFF9864EC7F7206

Engine #3 Key 218, Used in Decrypt Orientation =
FACC5FFDD03F6CEA1499306877150A2D6AB409DF8EDFFFD46C4FD71C662A28C1

AES Engines (each) <Input Stream> | <Output Stream>:

```
725DBB5CD19122DCEFB5B663A8CD75C279B3E60B22D1E2090C43E7E81AFEB9FA |
  DF3BD28B2C57D9BCAB00E36C1A7BD90CAABE6DB2E2968391511FFAC3B4B1A37E

80D52969BB001F16439D96A0D239C3031A570CBE4A0670F81AD7D67B5CDF17A1 |
  A0D86DA49A2656A4C97B8846C60FEC267F0B603B1F9BEFA4D583E5B9DCF82A50

DFBEDA9029C5D63779116C8A4461B5564339370F9F650E537C8EAC81883E2B6E |
  F05881E71A17C881BA2B34998D99F0BCD11D2E535097F51964DEABE3A3A5B0A6
```

The Pre-Processor and AES seeds, 459 and 927 respectively, along with the 3 AES engine orientation bits, are converted to a hex string <1CB39F> and inserted in the data stream:

```
                                         123456
bits, are converted to a hex string <1CB39F> and
```

```
-----------------------------------------------------
DF3BD28B2C57D9BCAB00E36C1A7BD90CAABE6DB2E2968391511FF

------------6--------------------3-------------------
AC3B4B1A37EAF0D86DA49A2656A4C97B8B846C60FEC267F0B603B

-----------------------------------------------------
1F9BEFA4D583E5B9DCF82A50F05881E71A17C881BA2B34998D99F

----------1--5---------------2---4
0BCD11D2E51359097F51964DEABE3CA3A35B0A6
```

The Checksum is then calculated and the checksums and checksum sub-key number (523) are converted to the Hex string <4DB122264A9CE569B20B> and inserted in the data stream:

```
            123456789ABCDEFGHIJK
Hex string <4DB122264A9CE569B20B> and inserted in the
```

```
--3-------H-8--------------------------E-------------
DFB3BD28B2BC657D9BCAB00E36C1A7BD90CAABE56DB2E29683915

-C----------------4--------------B-----7-----1-----K-
1C1FFAC3B4B1A37EAF10D86DA49A2656A94C97B28B8464C60FEBC

--A--5--------------------------------9-----6--------
26A7F20B603B1F9BEFA4D583E5B9DCF82A50F045881E271A17C88

--------------2G--F----------------------I--------J-D
1BA2B34998D99FD90B6CD11D2E51359097F51964D2EABE3CA30AE

35B0A6
```

Using the hex digits on either end of the data stream plus 2 bits from the Post-Processor seed, assembled as the new Overhead Key Number, the lower 4 bits of the current Post-Processor seed (11) plus the Overhead key number (515), used before this point,

```
                                          1234
are converted to the hex digits <203B> and inserted
in the data stream:
```

```
-----------------------------------------------------
DFB3BD28B2BC657D9BCAB00E36C1A7BD90CAABE56DB2E29683915
-----------------------------------------------------
1C1FFAC3B4B1A37EAF10D86DA49A2656A94C97B28B8464C60FEBC
-------------------1--------------3------------------
26A7F20B603B1F9BEFA24D583E5B9DCF823A50F045881E271A17C
--------------------2----4
881BA2B34998D99FD90B06CD1B1D2E51359097F51964D2EABE3CA
30AE35B0A6
```

Note: The Post-Processor Input was copied to the desired Post-Processor Output. Then all hex digit pairs in all segments were selected randomly, forming these key segments so as to form the desired output. Notice that all 3 segments are still able to exist in a random state even though all vertical sets of 3 key segment numbers all Xor to zero.

Vernam Post-Processor Input Data and Output Stream:

```
Data   From  AES: DFB3BD28B2BC657D9BCAB00E36C1A7BD90C
ReRandom@ 92,116: 00E90C42ED924DF9E98AD6634E9820B2DF7
ReRandom@ 97,017: F03D0C4CF2C4907459E4BFE9FD90A3BF42E
ReRandom@110,494: F0D4000E1F56DD8DB06E698AB308830D9D9
Post Data Output: DFB3BD28B2BC657D9BCAB00E36C1A7BD90C

AABE56DB2E296839151C1FFAC3B4B1A37EAF10D86DA49A2656A94
79FC686CFDD1D4BC0190EC706BE3E81470F5982CA0184FDDC2916
615324D894BA8C4DAC49C3337153111CAFA5571FD2762B11912E0
18AF4CB4696B58F1ADD92F431AB0F908DF50CF33726E64CC53BF6
AABE56DB2E296839151C1FFAC3B4B1A37EAF10D86DA49A2656A94

C97B28B8464C60FEBC26A7F20B603B1F9BEFA24D583E5B9DCF823
3920E47F6C76A3F256865D92B9D04CD12A619C47FA75441BBD651
9B9AE7F423EF5DBE9576F7C14985DF5A0D7EDDE9A0E99CB9418BF
A2BA038B4F99FE4CC3F0AA53F055938B271F41AE5A9CD8A2FCEEE
C97B28B8464C60FEBC26A7F20B603B1F9BEFA24D583E5B9DCF823

A50F045881E271A17C881BA2B34998D99FD90B06CD1B1D2E51359
D83DC04AB09B74247AACC81E7404CDE694587CF06A4FBCB54D4BA
3099C12A1CE230C86D0675ACC3472BE51F40697A9EEF373533F88
E8A40160AC7944EC17AABDB2B743E6038B18158AF4A08B807EB32
A50F045881E271A17C881BA2B34998D99FD90B06CD1B1D2E51359

097F51964D2EABE3CA30AE35B0A6
8CF4EC01FA1FAE0BC0C043FDAAF7
293D8EFB84C2D77A03FF7FA54ACF
A5C962FA7EDD7971C33F3C58E038
097F51964D2EABE3CA30AE35B0A6
```

After swapping the first digit pair with pair #3 and the last digit pair with pair #99, each of the resulting 2-digit

hex pairs is then converted to an ASCII character and would normally be written to the ciphertext output file. The ciphertext block:

```
½³ß(²¼e}›Ê°6Á§½□Ê«åm²â–ƒ‘QÁÿ¬;K□7êñ□†ÚI¢ej”É{(¸FL`þ¼&
§ò`;›ï¢MX>[□Ï,:PðE^-'□□È□°+4™□™ý□°1Ñ±Òå□Y□¦Q—M.«ãÊ0®5
°□
```

Example 9b #1

Below is just the Post-Processor output of the second of a total of 26 example files produced illustrating another example of the input text equal to the output text (the same as in 9b #1) and the associated randomly produced three segments (not the same as 9b #1).

Vernam Post-Processor Input Data and Output Stream:

```
Data   From  AES: DFB3BD28B2BC657D9BCAB00E36C1A7BD90C
ReRandom@ 92,116: 07B07FCCABD45E6A93FC8B02AF1E387AC18
ReRandom@ 97,017: 52F11ACB286C66D21B08FB6EEC61422EF59
ReRandom@110,494: 5541650783B838B888F4706C437F7A54341
Post Data Output: DFB3BD28B2BC657D9BCAB00E36C1A7BD90C
```

```
AABE56DB2E296839151C1FFAC3B4B1A37EAF10D86DA49A2656A94
60C1B9AD961C7E4491953B165DD8AD0789B9EA946CF071DFB5A15
1FA0F37E8D86C2DA1B0660B6CA13C088556B74D7475B7AB1CBF33
7F614AD31B9ABC9E8A935BA097CB6D8FDCD29E432BAB0B6E7E526
AABE56DB2E296839151C1FFAC3B4B1A37EAF10D86DA49A2656A94
```

```
C97B28B8464C60FEBC26A7F20B603B1F9BEFA24D583E5B9DCF823
41A7099948066008FB2B37AD56E265433BAE210D976016C35A288
8C297B2E3C2E64C06BFD8F7ECC08FBDBD3F1A594BE7DB71196A17
CD8E72B7742804C890D6B8D39AEA9E98E85F8499291DA1D2CC89F
C97B28B8464C60FEBC26A7F20B603B1F9BEFA24D583E5B9DCF823
```

```
A50F045881E271A17C881BA2B34998D99FD90B06CD1B1D2E51359
4C605B1D40322D39C4A49DF64D579EAEA7CD8143DB8B47E5FA49F
B9E5531C8F9F5E5FA496D3DADA0E7DC7EFF9B4A226FC281D7F9A8
F5850801CFAD736660324E2C9759E369483435E1FD776FF885D37
A50F045881E271A17C881BA2B34998D99FD90B06CD1B1D2E51359
```

```
097F51964D2EABE3CA30AE35B0A6
75B202C473144BD8ED5BBF2512A2
36E5389684E0A9555DF09EE2157E
C7B537798A1BB6489248AE65E58F
097F51964D2EABE3CA30AE35B0A6
```

Example 9b #2

---

10. Decryption – Starting with the unknown

In order to determine which randomly selected Post-Processor seed number was selected to encrypt the subject text, the decrypt engine needs to try all possible seeds until 5 specific data tests pass. This also prepares it for being capable of handling numbers generated by a Non-Deterministic Random Number Generator with no seed available.

Example 10, beginning on the next page, provides details of the process of the 'search' for the random selections while decrypting the ciphertext output of Example 1. For the specific example provided, it had to go through a total of 79,568 attempts, all of them failing, before attempt #75,569 was found to be successful. With room enough for only one attempt detail per page in this document, only three attempts will be provided for obvious reasons. As usual, the text file produced by the demonstration system is available that contains details of 50 failed attempts and the last successful attempt.

In spite of the significant amount of work that may be obvious per attempt, a computational shortcut, executed in non-demonstration or normal mode, results in successful synchronization in a mean time of about 0.6 second executing about 58,000 attempts in that time on the 'inefficient' demonstration system.

Attempt #1 - Line 1 = Ciphertext ASCII. Inputting Post-Processor seed 845 into a pseudo process produces blocks 894, 696 and 128 using offsets 0, 67 and 16 respectively, produces [<block#> * 111) + <offset>] these lines. Prior to processing, the Start and Finish hex digit pairs, displayed just above the 'SS' and 'FF' respectively, were swapped back to their possible original positions.

```
Raw   Input: 94D93227FED18316178580800F9B674E040CEE98D0A0C2
             SS--------------------------------------------
Good  Input: A6D93227FED18316178580800F9B674E040CEE98D0A0C2
Seg@ 99,234: 36A151DF0AF614877BAB29F47051E5F512C1CEC43BBEB1
Seg@ 77,323: CC44632476EEA6CF5202B7F8059C0FBF80FCBD8ADFDC66
Seg@ 14,224: B76DCB5C174E65178A4AAFC3348885D3873F9062096D25
Post Output: EB51CB8095875449B466B14F4EDE08D7110E0DB43DAF30

F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
-----------------------------------------------------------
F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
D32BD6EFC97D8918C75AADBBEFED117B271761661A634A83803CDEAAA05
FA92008A1EBA0E09E6D6A0E5DA9E98490C2FB3CA4B6961EE9995D4EBDF4
8058CC6BC96B02BC3D1A184E4BE12D132FE06BDD7E273BC6FF31B5CFBDA
514942B8AD36994F082F01F99A527FC51171F49E0484A90EA3F9BFA8384

7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
-------------------------------------SS--------------------
7122093D45756CC3CBC670FECD9CCF95E4B9494ED9D8F3CDBF7DA98F19F
D3999138A6543CA7E7B6B52240B725709BE6D67123A7BF96A4656AEBF2F
C58B0E094E22A6C4BEBC0D6C97F1FA7C1903D94689198AEDA40BFD9B9C3
E5BC75A86066FA61A20DB71CD141B9DC3031B77E8B9C84734DF74252E72
828CE3A4CD650CC130C17FACCB9BA945566DF107F8FA42C5F2E47CAD901

A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7D9D
----------------------------------------------FF--------FF
A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D03409D253F6E7D65
27D6AE6A9F367B7A430906140744FB821F1E484512D0D4FE109A6E8E85
147F17F7D0859605477ECD366E1FEE089AF4A6818A5145C3A88A1FE18C
7C0FAF732839E9E0D2A4A8E44BF2D74B7A7945384696A91AB744E4DC58
E609D4C5FD798E5AEA74B787F4D73693DA552A92A31478BA2A6BFBCE34

                                                1234
Test #1, Extracted 4 digits (used table #313): '12CA', the
4th digit 'A' should match the lower 4 bits of #845 (34Dh)
= 'D' - it does not - proceeding to the next attempt.

---------------------------------------------------2-------
EB51CB8095875449B466B14F4EDE08D7110E0DB43DAF30514942B8AD369

-----------------------------------------------------------
94F082F01F99A527FC51171F49E0484A90EA3F9BFA8384828CE3A4CD650

--1-------3------------------------------------------------
CC130C17FACCB9BA945566DF107F8FA42C5F2E47CAD901E609D4C5FD798

--4
E5AEA74B787F4D73693DA552A92A31478BA2A6BFBCE34
```

The search algorithm then continues, incrementing the ‘offset seed’ from 1 to 110 running each through the pseudo process for the offsets while using ‘845’ as the randomly selected block seed start. The next attempt, detailed on the following page, is #112 where it randomly selected block start ‘642’ and starts the ‘offset seed’ back to 0.

```
Attempt #112 - Line 1 = Ciphertext ASCII.  Inputting Post-
Processor seed 642 into a pseudo process produces blocks
888, 59 and 767 using offsets 0, 67 and 16 respectively,
produces [<block#> * 111) + <offset>] these lines.  Prior
to processing, the Start and Finish hex digit pairs,
displayed just above the 'SS' and 'FF' respectively, were
swapped back to their possible original positions.

Raw   Input: 94D93227FED18316178580800F9B674E040CEE98D0A0C2
             SS--------------------------------------------
Good  Input: 65D93227FED18316178580800F9B674E040CEE98D0A0C2
Seg@ 98,568: E734FD8866BB9E429D22848059D4FE9F8ACEA42F9AA646
Seg@  6,616: 4BA31F0E4DD908F108BF584197CE31E852D7AFDF57471D
Seg@ 85,153: 7A7FDD30E6E3FB38E0FBE62269D14F6565CCD3BAFC8180
Post Output: B3310D913350EE9D62E3BA63A850E75CB9D936D2E1C019


F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
--------------------------------------------------------
F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
0CFDCD22CA0A115A570966A4DB8DE544E3041537BF8C6AB228080BCA267
9669AA6996FE6ED66593EAF1AA439EC69198BC5371A1E83DAC443DC2D7D
8B41CC95A912208A3D3F1B5001EE5DDAAF3D9BDCBA2AEFD33DB07245313
E97DF368467C43E41B1C83EC94E0FDBCC8087F575FAED4F9FC9D446B3A6

7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
--------------------------------------------------------
7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
4D8753175ACB19735DF031D6D0F23E9C321F6A05782BCCB99B38B7F5D04
690EF1BE9F813DB658E7398DB713F11150DC5AEF29DE51E71EFC5400B68
30D4AE21E8C4BA4CBF40AC2A317E3B7866551A06D05B3F24D2550683F8E
657F05B568FBF24A7191D48F9B033B60E02F6082587651B7E8EC4CF987D

A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7D9D
----FF--------------------------------------SS-------FF
A9AF9D2B9AF38AC53CA7D441D67EF45225C6816E7D034094253F6E7DC2
FC3D319F2417EEC9C6148D65C6D63CBAE17AB1759CC0BFF27A6C27B772
C63F4647B602872E47C328DEC0FDCA42F28CECAF6E10BD6AF5922F6A30
9940DF7BAFA7F2AB25724BB5070EEF9C1BBA77A366F4B662F3309C1ACF
0AED3588A741118998023A4FD75BED362D8AAB17E927F46E59F1FABA4F

                                                   1234
Test #1, Extracted 4 digits (used table #1,006): 'B451',
the 4th digit '1' should match the lower 4 bits of #642
(282h) = '2' - it does not - proceeding to the next
attempt.

-------------------1------------------------------------
B3310D913350EE9D62E3BA63A850E75CB9D936D2E1C019E97DF368467C4

--2---------------------3-------------------------------
3E41B1C83EC94E0FDBCC8087F575FAED4F9FC9D446B3A6657F05B568FBF

-------------------------------------------------------4

24A7191D48F9B033B60E02F6082587651B7E8EC4CF987D0AED3588A7411

18998023A4FD75BED362D8AAB17E927F46E59F1FABA4F
```

The search algorithm, once again, increments the ‘offset seed’ from 1 to 110 running each through the pseudo process for the offsets while using ‘642’ as the randomly selected block seed start. The next attempt, detailed on the following page, is #223 where it randomly selects block start ‘464’ and resets the ‘offset seed’ to 0.

Attempt #223 - Line 1 = Ciphertext ASCII. Inputting Post-Processor seed 220 into a pseudo process produces blocks 405, 687 and 482 using offsets 0, 67 and 16 respectively, produces [<block#> * 111) + <offset>] these lines. Prior to processing, the Start and Finish hex digit pairs, displayed just above the 'SS' and 'FF' respectively, were swapped back to their possible original positions.

```
Raw   Input: 94D93227FED18316178580800F9B674E040CEE98D0A0C2
             SS--------------------------------------------
Good  Input: B6D93227FED18316178580800F9B674E040CEE98D0A0C2
Seg@ 91,797: 2985C5FBA0501726FDE0B2B95EADBB90410EBC6097FE55
Seg@ 95,416: DF1DD5446140AF2A923D411916FE5CED69D89146BB10A1
Seg@ 29,098: BF2C953EC324E3F20F4C86F8700ECEB1D66A51A9ABF3DF
Post Output: FF6DB7A6FCE5D8E87714F5D837C64E82FAB0921757BDE9

F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
--FF--SS---------------------------------------------------
F89D5894B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
9A1A1F2F92E1FF32E0329BF8079721815433FA28095708D5BB512C75F98
2DFE9E35FA3D31CF752F72013B0C8DDD639FEEAC629D48E32C35DC529CE
BF631AB88655AFFDECC9A565195216319A5E8762BF0B6C29413CEF9F29E
F01AC3365D137DE26D6D5875C1096189B85BDE09FF6895BA93391F9EB67

7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
-----------------------------------------------------------
7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
40CA72EBE13B8C9C5B9D29A33047080844F16416E4A3EB971429537AEA2
8F89F169BBEF36BEA26F3FDC046F22325D165CE56DE976D1BEC9D0F5522
DEDB6ECC34170DBE8464EBB15970CAFF0198B5F993C0F537F7DAAA07930
60BAE4732BB6DB5FB6508D30A0C42F50FCC6C764C3529BBCE247800732F

A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7D9D
--------------------------------------------------------FF
A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7DA8
6724FCBA58E539C05603DE14CB89D7C917AE91603DB454D6DC10D3E262
8DC8D7DBEB557661640364F9BFCCB92010C453152EB3E097680439F074
F8C7127C0DDA24A7ACE0A894BA7BCC89F2C84636D987E9F3F57C764A3A
BB84FB362499E1C3A247C63818405632D064052DB7831DD76457F22584

                                                1234
Test #1, Extracted 4 digits (used table #316): '8F96', the
4th digit '6' should match the lower 4 bits of #464 (1D0h)
= '0' - it does not - proceeding to the next attempt.

--4----------1------2--------------------------------------
FF6DB7A6FCE5D8E87714F5D837C64E82FAB0921757BDE9F01AC3365D137

-----------------------------------------3
DE26D6D5875C1096189B85BDE09FF6895BA93391F9EB6760BAE4732BB6D

B5FB6508D30A0C42F50FCC6C764C3529BBCE247800732FBB84FB362499E

1C3A247C63818405632D064052DB7831DD76457F22584
```

The search algorithm then increments the ‘offset seed’ from 1 to 110 running each through the pseudo process for the offsets while using ‘220’ as the randomly selected block seed start. This search process continued for 77,345 more failed attempts. Attempt #77,349, the one that it finally found to be successful, is detailed starting on the next page.

Attempt #79,569 - Line 1 = Ciphertext ASCII. Inputting Post-Processor seed 948 into a pseudo process produces blocks 55, 866 and 101 using offsets 92, 56 and 81 respectively, produces [<block#> * 111) + <offset>] these lines. Prior to processing, the Start and Finish hex digit pairs, displayed just above the 'SS' and 'FF' respectively, were swapped back to their possible original positions.

```
Raw   Input: 94D93227FED18316178580800F9B674E040CEE98D0A0C2
             SS--------------------------------------------
Good  Input: 3CD93227FED18316178580800F9B674E040CEE98D0A0C2
Seg@  6,197: 028BF3E11432A31FAEA9B6C27391969CBC73F99898B108
Seg@ 96,182: 0B5DA685653997610404773953D51570BAB9552A5D9D83
Seg@ 11,292: C99A2770F5BD0F087D7341B3FAC23E3E128AF42E4A0B35
Post Output: FC9540337A67B860C05B00C8D51DDA9C104CB6045F877C

F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
-----------------------------------------------------------
F8A858B6B39A1CE214B914E9E4C0DBE415A94DEF2BA9B9A545610026FAF
4D11BCC1D2B18899D5453D66E7E1307D1A241117CEBE66FAC82F385584D
54076DE7E4617E9A62682997A71EB915E47F8814E7B664369A9138D1CD3
5F1CA868EB4AEB7F4D71C829C6E1264800AC9BA5F2FD77E303AB05B1720
BEA221F86E00019EEEE5C83162DE74C4EB5E4F49F05CCC8A14740513C11

7122093D45756CC3CBC670FECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
---------------SS----FF------------------------------------
7122093D45756CC94BC679DECD9CCF95E4B94A6ED9D8F3CDBF7DA98F19F
10BEBCD1814748A19A740DB49EF1D6487E17BE95D73CE1C830934B2A52F
3E84235087C84B4FE8101DAF3ACA8AC69C13FA581B3FBFB270D4E238A90
E0D44F5129D05A08A0A8609C631BBAA670267867240D410A482C748057E
BFCCD9ED6A2A352F990A09590ABC29BD769B76C431D6ECBDB716741DB5E

A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7D9D
--------------------------------------------------------FF
A9AFC22B9AF38AC53CA7D441D67EF45225C6816E7D034065253F6E7D0F
58BC678E9635A5195FD7FF88DF349507DE010F2FE4B5E135A153CD618C
0097CF1E81D706D67DAE5D81B2B6932DFE1EB017BDABF3CE51FD0D9602
B327C0862D3D3253A099410242E9EF5C46E22F5A54E2F7E936D496EAD4
42A3AA3DA02C1B59BE47374AF9151D24433B110C70FFA577E345386055
```

1234

Test #1, Extracted 4 digits (used table #380): <1A94>, the 4th digit '4' should match the lower 4 bits of #948 (3B4h) = '4' - it does - testing proceeds to the next check.

```
-----------------------------------------------------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F86E000

-------------------------4-------------------1-------------
19EEEE5C83162DE74C4EB5E4F49F05CCC8A14740513C11BFCCD9ED6A2A3

----3-------------------------------------------2
52F990A09590ABC29BD769B76C431D6ECBDB716741DB5E42A3AA3DA02C1

B59BE47374AF9151D24433B110C70FFA577E345386055
```

Test #2, The first 3 digits after conversion (425) should be below 1,024 - it is - testing proceeds to the next check.

```
                               123456789ABCDEFGHIJK
Test #3, Extracted 20 digits: <6DAAB706CD5DA34071A0>, the
last 3 digits '1A0' convert to '416', it should be below
1,024 - it is - testing proceeds to the next check.
```

The Pre-Processor Offset, '93', was extracted and reassembled from the 7th hex digit 'riding' on these two checksums.

```
--------6------------G----------------K--------------------
FC9540337A67B860C05B00C8D51DDA9C104CB6045F877CBEA221F86E000

---------E---2-H----------------------7--------9---C---J---
19EEEE5C83162DE74C4EB5E4F9F05CCC8A14740513C1BFCCD9ED6A2A352

---------3-------8-------------A---1----5---------D--------
F90A09590ABC29BD769B76C431D6ECBDB716741DB5E423AA3DA02C1B59B

------4---B---F----I
E47374AF9151D24433B110C70FFA577E345386055
```

Test #4, 3 Checksums: '5,942,128', '5,070,243' and '1,031' checked for being within range - they are - testing proceeds to the next check.

Test #5, 3 calculated checksums: '5,942,128', '5,070,243' and '1,031' checked for being equal to the above - they do match - synchronization attained - decryption begins.

```
                     123456
Extracted 6 digits: <0E42EF>.  The first 3 digits, ‘0E4’ =
228.  The lower 14 bits were converted to the Pre-
Processor's Seed (#228).  The last 3 digits, '2EF' = 751.
```

The lower 14 bits were converted to AES Cipher's Seed (#751). The 3-bit AES Operational Orientation number, '0', was reassembled from the upper 2 bits of these 2 numbers.

```
-------------------------------1---------------4-6---------
FC954033A67B860C05B0C8D51DDA9C104CB645F877CBEA221F86E00019E

25-----------------------------3
EEE5C8162E4C4EB5E4F9F05CCC8A1474513C1BFCD9E6A2352F90A09590B

C29BD79B76C431D6ECBB71741D5E423AA3D02C1B59BE47374F911D2433B

10C70FFA577E345386055
```

The 3 AES Cipher Engines using the specified keys executed in the specified orientations to decrypt this data:

```
AES In  = FC954033A67B860C05B0C8D51DDA9C14CB645F877CBEA2186
E00019EE5C8162E4C4EB5E4F9F05CCC8A147513C1BFCD9E6A2352F90A09
590BC29BD79B76C431D6ECBB71741D5E423AA3D02C1B59BE47374F911D2
433B10C70FFA577E345386055
```

Seed = 751,

Key for Engine #1 (#1,013 Encrypt):
C89B8EE20B0EBDF91E15314A65CC9E885B1D5D7A3B9BBEFC00AEA04690F14070

Key for Engine #2 (#959 Encrypt):
1DA89553F435FA3E2702A160BE886D214F9448CFC5F80DED6081A6CFFD297163

Key for Engine #3 (#579 Encrypt):
16BAB20F57D21DFD4A0E6E384D1F268D13C39B4EB08BA63DF0E29DACED4B81E3

```
AES Out = 932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9
CDBB5C629A76635D35C65417FE06381054904F8791EF5336592CD14D2C0
637DEA88D38D215F18D718F6DFE744C7DD2BEDDE534CEA971D0C6165823
5F9238CBA812A38068151FA39

Extracted 4 digits from the 3 concatenated AES outputs:
 1234
<31E5>, the first 3 digits convert to next block Post-
processor seed '798', Engine offset seed # = '0 (0)'.

-----------------------------------------------------------
932637BCBD9F6C0842EB72EDDBEDD04EB9FAEBE8B64A2EAD9CDBB5C629A

-----------------------------------------------------------
76635D35C65417FE06381054904F8791EF5336592CD14D2C0637DEA88D3

----------------------------------3--------2-4-------------
8D215F18D718F6DFE744C7DD2BEDDE534CEA971D0C61658235F9238CBA8

---1
12A38068151FA39

Preprocessor Seeds (228, 93) > pseudo(228, 93) = (1,013,
101) * 111 = 112,536, pseudo(1,013, 101) = (758, 96) * 111
= 84,239, pseudo(758, 96) = (437, 0) * 111 = 48,603

Pre-Processor  Data  Input: 932637BCBD9F6C0842EB72EDDBEDD04
Key Segment start @112,536: ED873B05A3C59DD0B18EA8B81F18CF3
Key Segment start @ 84,239: 7E33E29E80498E0E141DA45CB9C3327
Key Segment start @ 48,603: 54FA8754BE7A0CF686580A6C0E420D6
Pre-Processor ASCII Output: 5468697320697320612074657374206

EB9FAEBE8B64A2EAD9CDBB5C629A76635D35C65417FE06381054904F879
60D554B6EA317B26D4A78CBAE1D79A1E49AE9D6F39E883A40906D9C2ADC
0848D9540653AE2A1AB6A855DCA5528855DBDD70867216185DE34015E6A
7560241AE15472A0809A8955C8BE6CF17667114CEE92E4A253B78F0EFEF
F66207468652054697461 6E69756D204372797074 6F677261706869632 0

1EF5336592CD14D2C0637DEA88D38D215F18D718F6DFE744C7DD2BEDDE5
A5E0D8A31D5236B0C23A3AAEE6B16C99E6B2A5730A741738ACF88D139CD
8D3D0E6AA00FBC38ED4310AE772045E8784A7F4C53E75FC8CA01D65B712
655196D84AFDBE1F817925936936CD3FAFC05E42CC38C6DBCF0422C45DC
53797374656D20456E6372797074696F6E2053656374696F6E2052616E6

34CA971D0C668235F9238CBA812A8068151
EF4D2C47C1B980831B7D270F7AA6C893884
C36D763688FE5334838492554AC679EAD98
5E1C1F68A27787A436F83EB7875C4743A53
46F6D204E756D6265722075736167652E1E

Converting hex pairs in the 'Pre-Processor Data Output'
line above to single ASCII characters, the resulting output
between '>>>' and '<<<':

        >>>This is a test of the Titanium Cryptographic
System Encryption Section Random Number usage.<<<
```

Example 10

Note: The ‘search’ process selects block starts at random so as to not yield what number it found by any analysis of the execution time of this search. Sometimes it will find it in less than 500 attempts, other times it will be greater than 100,000, most of the time something in-between, all determined randomly.

11. Multiple Block attacks

Eventually, if enough plaintext is encrypted, there will be some Vernam Key segment overlapping with what was used to encrypt other blocks. It is this unknown factor that will guarantee failure for any potential attacker. If they attempt to attack this system and the ciphertext file is large enough, they will find they have run out of room to store the segments they have created.

If they carry out this attack anyway, they will be able to create 3-94 byte segments and 3-111 byte segments per block so attacked, a total of 615 bytes. After decrypting about 184 blocks, they will find they have no more room to create and store the segments. Although it will always be unknown (except by analyzing demonstration outputs), there most likely will be overlap usage of the Vernam key in this system within the first 184 blocks. So the attacker will realize they have failed if they consistently create segments that have no way of overlapping during their production. And even if they code the attack to create some segments that overlap, there is no mathematical ability to determine which segments, if any, should overlap.

There are serious questions that will be needed to be answered by any attack algorithm:

1. What would the criteria be for the detection of at least a partial key segment usage overlap during the decryption of a particular block with the segment used in one of the other previous blocks?
2. Which other block would have these segment overlaps?
3. What factors determine the values being overlapped?
4. How many locations are common between the usages?

In any case, there is no mathematical information within the ciphertext block to create the segment pointers to place the segments they have created. Yes, there is a seed start value; but it references the chain table in creating the pointers. There is no mathematical information within the ciphertext that could ever yield the content of the chain table since that table is not used to perform any encryption of the ciphertext.

12. Another capability – Group Segregation

This system also has the capability of segregating users into groups so that one group, if not authorized, cannot decrypt the text of another group even if the same key table is used by both users. This capability is also used to encrypt the ‘Last Login’ information in such a way that no user, not even the System Administrator, can forge a false entry. If the last login information has been tampered with in any way so that the system is unable to provide that information to the user, they have the ability to contact the System Administrator for any appropriate action that may be needed.

For the demonstration system, it has the designed capacity for one billion groups with 255 users per group. This system can be arranged to duplicate the organizational chart of any corporation in the world that has no more than about 250 users per group. The maximum number of users depends upon the number of ‘Group Leaders’ above each bottom-most group of users. Users within groups can also be authorized or not authorized to decrypt any text from any other user within the group. Group leaders would only be allowed to decrypt text produced by users at or below them in the organizational chart. Any Group Leader above two or more Groups would be able to Group-translate a ciphertext produced by one group for use by another group if needed.

One use could be to provide security for every voter in the United States. Every voter could be a ‘User’ with groups set to streets, towns, cities, counties and states. Various ‘group leaders’ (Town/City, County or State, for example) would be able to decrypt or ‘obtain’ the votes cast for all users beneath them with no authorization to see how any other vote in the system was cast.

13. Conclusion

The use of the 3-key-segment Vernam engine as a Pre and Post Processor for this MOO gives it the ability to very quickly be extremely flexible. They allow multiple key changes and/or multiple plaintext input changes and/or multiple ciphertext changes while holding other various elements of the encryption process constant. This also prevents any mathematical information that could ever be gleaned from any analysis of the plaintext/ciphertext pair. Why? Simply, the keys and/or the plaintext input can change and not cause a change in the output, or any other ‘hold some, vary other’ scenarios. This provides no mathematical information for any attack technology that relies on changes to perform the needed calculations.

Consider a total of 4 randomly set segments, 3 for the Pre-Processor and the first one for the Post-Processor. For each segment in the Pre-Processor, there are 94 factorial (1.0873 x 10^{146}) ways the numbers within each segment can be rearranged and still be considered a randomly created key. Even if you hold all of the other 3 segments constant, you have more possible keys than all keys possible for the 256-bit AES engine (1.1579 x 10^{77}). In Example 9, it has already been demonstrated that the last two segments can be randomly recreated to make the ciphertext not change. But you have two other Pre-Processor segments that can be rearranged plus the first Post-Processor segment with 111 numbers (111 factorial=1.76 x 10^{180}). Multiplying the possible segments for the Pre-Processor and Post-Processor, the result is 2.262 x 10^{618}. This does not even take into consideration creating who-knows-how-many other sets of 4 randomly created segments to rearrange. It also does not consider that, for each rearrangement of the 4 segments, the AES engines in the center have the ability to do whatever they want to with the data created by the Post-Processor with any key, also randomly created, not to mention the 8 possible engine orientations for each key set.

The reader can well imagine that with so many keys that could decrypt any given plaintext/ciphertext pair, that the attacker should well realize the absolutely total futility of even attempting to start to attack this MOO design, no matter what computational hardware or attack software was ever to be built or developed.

No other MOO offers the flexibility of this proposal, dealing with the issue of preventing cipher attacks in a whole new way. It permits massive numbers of sets of keys that successfully convert the ciphertext to the subject plaintext. It will allow the use of a Non-Deterministic Random Number Generator when the generator has been approved for use in cryptographic systems, allowing further improvement in its security, if needed.

Today's MOO's **are** completely secure; but hardware and attack technologies are not going to stand still. Who knows if some brainiac teenager will create an extremely sophisticated attack algorithm that, when executed on a then-existing Quantum Computer, would render the AES and possibly one or more of the current MOO's broken.

There was almost a 4 year gap from the time the previous DES algorithm was found to be beyond its useful lifetime and the formal adoption of the current AES algorithm as the new standard. Considering the information and data currently being protected, the security industry cannot afford another gap between acceptable standards. Now, while current MOO's are secure, is the time to analyze this new MOO. If approved and usage is started and then used throughout all areas currently using security, and if technology improves to the point where the current MOO's become broken, the information that was converted to this new MOO will still be protected, the goal that I believe the entire security industry should desire.

In spite of the almost impossible odds, someone may claim that it would still be possible to create the keys for one block. To illustrate, look at a lottery ticket where you need to select 7 numbers from 1 to 45 and consider the odds of winning. But with this MOO 'Lottery', you have to pick 94 numbers from 0 to 255, occasional duplication of numbers is allowed, and select these numbers correctly 3 times in a row. Then, in the same drawing, pick 111 numbers (same range) correctly 3 times in a row. On top of that, pick 64 numbers from 0 to 15, 3 times in a row, also all correctly. All this needs to be done in one 'pick'. Suppose, to give someone the benefit of the doubt, they do pick them all correctly. What would be the indication of their being correct, considering all the other incorrect picks they most likely made? They would have had to process an unknown number of other blocks where all the keys were reused in a random fashion in separate blocks, and know when attacking those blocks what other keys needed to be created.

It would be anyone's guess as to what computational, memory and disk resources would be needed to even begin attacking this MOO. The whole idea is to persuade all potential attackers to not even try to start, this seems to be an adequate beginning to that end. Who knows how many attackers are trying even now to break the AES and whether or not some 'brainiac' teenager may someday help them in a way no one expected.

Appendix A

Suppose you have a small function that selects random numbers from 1 to 1,000 and records each when selected, ending when eventually all numbers from 1 to 1,000 have been selected. What is the number of random accesses that were needed to have all numbers eventually selected? The following list of numbers shows the total number of accesses, from 6,036 to 12,415, required to randomly encounter all numbers from 1 to 1,000 inclusive through 50 sets of random accesses.

Try #1, it took 6,049 random accesses to encounter all numbers from 1 to 1,000.
Try #2, it took 6,135 random accesses to encounter all numbers from 1 to 1,000.
Try #3, it took 6,330 random accesses to encounter all numbers from 1 to 1,000.
Try #4, it took 8,629 random accesses to encounter all numbers from 1 to 1,000.
Try #5, it took **6,036** random accesses to encounter all numbers from 1 to 1,000.
Try #6, it took 6,899 random accesses to encounter all numbers from 1 to 1,000.
Try #7, it took 6,181 random accesses to encounter all numbers from 1 to 1,000.
Try #8, it took 9,766 random accesses to encounter all numbers from 1 to 1,000.
Try #9, it took 8,537 random accesses to encounter all numbers from 1 to 1,000.
Try #10, it took 7,929 random accesses to encounter all numbers from 1 to 1,000.
Try #11, it took 8,898 random accesses to encounter all numbers from 1 to 1,000.
Try #12, it took 6,820 random accesses to encounter all numbers from 1 to 1,000.
Try #13, it took 7,221 random accesses to encounter all numbers from 1 to 1,000.
Try #14, it took 6,547 random accesses to encounter all numbers from 1 to 1,000.
Try #15, it took 6,712 random accesses to encounter all numbers from 1 to 1,000.
Try #16, it took 7,443 random accesses to encounter all numbers from 1 to 1,000.
Try #17, it took 6,591 random accesses to encounter all numbers from 1 to 1,000.
Try #18, it took 6,683 random accesses to encounter all numbers from 1 to 1,000.
Try #19, it took 9,090 random accesses to encounter all numbers from 1 to 1,000.
Try #20, it took 8,644 random accesses to encounter all numbers from 1 to 1,000.
Try #21, it took 6,549 random accesses to encounter all numbers from 1 to 1,000.
Try #22, it took 6,542 random accesses to encounter all numbers from 1 to 1,000.
Try #23, it took 7,571 random accesses to encounter all numbers from 1 to 1,000.
Try #24, it took 6,717 random accesses to encounter all numbers from 1 to 1,000.
Try #25, it took 5,728 random accesses to encounter all numbers from 1 to 1,000.
Try #26, it took 6,882 random accesses to encounter all numbers from 1 to 1,000.
Try #27, it took 7,152 random accesses to encounter all numbers from 1 to 1,000.
Try #28, it took 8,531 random accesses to encounter all numbers from 1 to 1,000.
Try #29, it took 8,256 random accesses to encounter all numbers from 1 to 1,000.
Try #30, it took 5,892 random accesses to encounter all numbers from 1 to 1,000.
Try #31, it took 6,677 random accesses to encounter all numbers from 1 to 1,000.
Try #32, it took 7,577 random accesses to encounter all numbers from 1 to 1,000.
Try #33, it took 8,082 random accesses to encounter all numbers from 1 to 1,000.
Try #34, it took 6,836 random accesses to encounter all numbers from 1 to 1,000.
Try #35, it took 5,665 random accesses to encounter all numbers from 1 to 1,000.
Try #36, it took 8,039 random accesses to encounter all numbers from 1 to 1,000.
Try #37, it took 8,070 random accesses to encounter all numbers from 1 to 1,000.
Try #38, it took 6,353 random accesses to encounter all numbers from 1 to 1,000.
Try #39, it took 7,490 random accesses to encounter all numbers from 1 to 1,000.
Try #40, it took 6,263 random accesses to encounter all numbers from 1 to 1,000.
Try #41, it took 6,576 random accesses to encounter all numbers from 1 to 1,000.
Try #42, it took 6,200 random accesses to encounter all numbers from 1 to 1,000.
Try #43, it took 9,561 random accesses to encounter all numbers from 1 to 1,000.
Try #44, it took 5,885 random accesses to encounter all numbers from 1 to 1,000.
Try #45, it took 5,765 random accesses to encounter all numbers from 1 to 1,000.
Try #46, it took 6,418 random accesses to encounter all numbers from 1 to 1,000.
Try #47, it took 7,229 random accesses to encounter all numbers from 1 to 1,000.
Try #48, it took 8,930 random accesses to encounter all numbers from 1 to 1,000.
Try #49, it took 6,537 random accesses to encounter all numbers from 1 to 1,000.
Try #50, it took **12,415** random accesses to encounter all numbers from 1 to 1,000.

Appendix B

Three Pre-Processor Pseudo selection process examples:

```
Seed     0, Sg 1 =  51,253, Sg 2 =  75,557, Sg 3 = 101,482
Seed     1, Sg 1 = 100,648, Sg 2 =  38,927, Sg 3 =  12,016
Seed     2, Sg 1 =  21,505, Sg 2 =  49,472, Sg 3 = 100,816
Seed     3, Sg 1 =  85,441, Sg 2 =  98,645, Sg 3 = 105,478
Seed     4, Sg 1 =  59,578, Sg 2 =     743, Sg 3 =  89,050
    - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Sg 1 =  20,617, Sg 2 =   3,407, Sg 3 =  32,551
Seed 1,020, Sg 1 =  61,021, Sg 2 =  20,501, Sg 3 =  39,433
Seed 1,021, Sg 1 =  19,729, Sg 2 =  51,803, Sg 3 =  91,270
Seed 1,022, Sg 1 =  83,332, Sg 2 =  18,392, Sg 3 =  50,200
Seed 1,023, Sg 1 =   5,965, Sg 2 = 106,970, Sg 3 =  92,491

    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Seed     0, Sg 1 =  68,903, Sg 2 =  72,895, Sg 3 =  81,155
Seed     1, Sg 1 =  36,380, Sg 2 =   9,070, Sg 3 =  99,581
Seed     2, Sg 1 =  96,098, Sg 2 =  43,036, Sg 3 =  82,376
Seed     3, Sg 1 = 108,308, Sg 2 =   6,739, Sg 3 =  70,610
Seed     4, Sg 1 =  69,347, Sg 2 =  26,164, Sg 3 =   1,901
    - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Sg 1 =  86,996, Sg 2 =  40,594, Sg 3 =  56,735
Seed 1,020, Sg 1 =  73,343, Sg 2 =  32,047, Sg 3 =  79,823
Seed 1,021, Sg 1 = 101,870, Sg 2 =  85,549, Sg 3 =  43,637
Seed 1,022, Sg 1 =  98,651, Sg 2 =  79,111, Sg 3 =  69,389
Seed 1,023, Sg 1 = 103,424, Sg 2 =  33,046, Sg 3 =  76,493

    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Seed     0, Sg 1 =   4,898, Sg 2 =  96,442, Sg 3 = 112,142
Seed     1, Sg 1 =  84,818, Sg 2 = 111,538, Sg 3 =  40,991
Seed     2, Sg 1 =  62,729, Sg 2 =  49,822, Sg 3 =  15,017
Seed     3, Sg 1 =   5,231, Sg 2 =  43,162, Sg 3 =  88,832
Seed     4, Sg 1 =  29,651, Sg 2 =  17,854, Sg 3 =  27,227
    - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Sg 1 =     902, Sg 2 =  50,377, Sg 3 =  46,430
Seed 1,020, Sg 1 =   1,235, Sg 2 =  50,599, Sg 3 =   9,134
Seed 1,021, Sg 1 =  17,996, Sg 2 =   3,313, Sg 3 =  52,757
Seed 1,022, Sg 1 =  29,207, Sg 2 = 111,316, Sg 3 =   7,247
Seed 1,023, Sg 1 =  53,072, Sg 2 =  86,674, Sg 3 =  65,966
```

These examples are three of over 1 billion different possible initializations due to the involvement of 4 other randomly selected key numbers involved in the pseudo process that produced these numbers.

AES Pseudo selection process examples (1 of many):

```
Seed     0, Eng 1 =   483, Eng 2 =   180, Eng 3 =   116
Seed     1, Eng 1 =   325, Eng 2 =    69, Eng 3 =    10
Seed     2, Eng 1 =   780, Eng 2 =   786, Eng 3 =   472
Seed     3, Eng 1 =   547, Eng 2 =    87, Eng 3 =   471
Seed     4, Eng 1 =   723, Eng 2 =   344, Eng 3 =   749
    - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Eng 1 =   347, Eng 2 =   454, Eng 3 =    23
Seed 1,020, Eng 1 =   288, Eng 2 =   943, Eng 3 =   485
Seed 1,021, Eng 1 =   596, Eng 2 =   869, Eng 3 =   133
Seed 1,022, Eng 1 =   892, Eng 2 =   316, Eng 3 =   940
Seed 1,023, Eng 1 =   228, Eng 2 =   480, Eng 3 =   959

  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Seed     0, Eng 1 =   197, Eng 2 =   931, Eng 3 =   690
Seed     1, Eng 1 =   212, Eng 2 =   888, Eng 3 =   682
Seed     2, Eng 1 =   728, Eng 2 =   917, Eng 3 =   301
Seed     3, Eng 1 =   881, Eng 2 =   660, Eng 3 =   203
Seed     4, Eng 1 =   570, Eng 2 =   783, Eng 3 =   525
    - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Eng 1 =   287, Eng 2 =   624, Eng 3 =   152
Seed 1,020, Eng 1 =   936, Eng 2 =   975, Eng 3 =   732
Seed 1,021, Eng 1 =   579, Eng 2 =   865, Eng 3 =   872
Seed 1,022, Eng 1 =   136, Eng 2 =   327, Eng 3 =   355
Seed 1,023, Eng 1 =   248, Eng 2 =   620, Eng 3 =   187

  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Seed     0, Eng 1 =   347, Eng 2 =   468, Eng 3 =   510
Seed     1, Eng 1 =   353, Eng 2 =    94, Eng 3 =   181
Seed     2, Eng 1 =   787, Eng 2 =   812, Eng 3 =   669
Seed     3, Eng 1 =   897, Eng 2 =   855, Eng 3 =   794
Seed     4, Eng 1 =   228, Eng 2 =   205, Eng 3 =   904
    - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - -
Seed 1,019, Eng 1 =   483, Eng 2 =   166, Eng 3 =   636
Seed 1,020, Eng 1 =   524, Eng 2 =   830, Eng 3 =   198
Seed 1,021, Eng 1 =   796, Eng 2 =   553, Eng 3 =    84
Seed 1,022, Eng 1 =   704, Eng 2 =   971, Eng 3 =   797
Seed 1,023, Eng 1 =   723, Eng 2 =   497, Eng 3 =   390
```

These examples are three of over 1 billion different possible initializations due to the involvement of 4 other randomly selected key numbers involved in the pseudo process that produced these numbers.

Post-Processor Pseudo selection process examples:

```
 Seed     0, Sg 1 =   7,523, Sg 2 =  75,805, Sg 3 = 108,836
 Seed     1, Sg 1 =  98,987, Sg 2 =  57,490, Sg 3 =  35,798
 Seed     2, Sg 1 =  49,814, Sg 2 =  89,902, Sg 3 =  73,982
 Seed     3, Sg 1 =   5,525, Sg 2 =  57,268, Sg 3 =  82,751
 Seed     4, Sg 1 =  73,124, Sg 2 =  75,916, Sg 3 = 108,947
     - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - - - - - - - - - - - -
 Seed 1,019, Sg 1 =   7,967, Sg 2 =  84,019, Sg 3 =  44,789
 Seed 1,020, Sg 1 = 103,316, Sg 2 =  45,169, Sg 3 =  78,533
 Seed 1,021, Sg 1 =  68,129, Sg 2 =  38,065, Sg 3 =  54,113
 Seed 1,022, Sg 1 =  60,914, Sg 2 =   8,317, Sg 3 =  77,423
 Seed 1,023, Sg 1 =  57,029, Sg 2 =  88,903, Sg 3 =  12,488

     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

 Seed     0, Sg 1 =   7,443, Sg 2 =  75,714, Sg 3 = 108,874
 Seed     1, Sg 1 =  98,907, Sg 2 =  57,399, Sg 3 =  35,836
 Seed     2, Sg 1 =  49,734, Sg 2 =  89,811, Sg 3 =  74,020
 Seed     3, Sg 1 =   5,445, Sg 2 =  57,177, Sg 3 =  82,789
 Seed     4, Sg 1 =  73,044, Sg 2 =  75,825, Sg 3 = 108,985
     - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - - - - - - - - - - - -
 Seed 1,019, Sg 1 =   7,887, Sg 2 =  83,928, Sg 3 =  44,827
 Seed 1,020, Sg 1 = 103,236, Sg 2 =  45,078, Sg 3 =  78,571
 Seed 1,021, Sg 1 =  68,049, Sg 2 =  37,974, Sg 3 =  54,151
 Seed 1,022, Sg 1 =  60,834, Sg 2 =   8,226, Sg 3 =  77,461
 Seed 1,023, Sg 1 =  56,949, Sg 2 =  88,812, Sg 3 =  12,526

     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

 Seed     0, Sg 1 =   7,437, Sg 2 =  75,769, Sg 3 = 108,796
 Seed     1, Sg 1 =  98,901, Sg 2 =  57,454, Sg 3 =  35,758
 Seed     2, Sg 1 =  49,728, Sg 2 =  89,866, Sg 3 =  73,942
 Seed     3, Sg 1 =   5,439, Sg 2 =  57,232, Sg 3 =  82,711
 Seed     4, Sg 1 =  73,038, Sg 2 =  75,880, Sg 3 = 108,907
     - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - - - - - - - - - - - -
 Seed 1,019, Sg 1 =   7,881, Sg 2 =  83,983, Sg 3 =  44,749
 Seed 1,020, Sg 1 = 103,230, Sg 2 =  45,133, Sg 3 =  78,493
 Seed 1,021, Sg 1 =  68,043, Sg 2 =  38,029, Sg 3 =  54,073
 Seed 1,022, Sg 1 =  60,828, Sg 2 =   8,281, Sg 3 =  77,383
 Seed 1,023, Sg 1 =  56,943, Sg 2 =  88,867, Sg 3 =  12,448
```

These examples are three of 110 different possible initializations due to the involvement of the randomly selected offset number involved in the pseudo process that produced these numbers.

To obtain the entire output of all nine pseudo process examples, request Appendix B.

Appendix C

cT(0)=67	cT(363)=689	cT(525)=118	cT(326)=392
cT(67)=682	cT(689)=631	cT(118)=702	cT(392)=915
cT(682)=980	cT(631)=271	cT(702)=286	cT(915)=443
cT(980)=925	cT(271)=266	cT(286)=275	cT(443)=41
cT(925)=14	cT(266)=1,001	cT(275)=947	cT(41)=243
cT(14)=538	cT(1,001)=780	cT(947)=281	cT(243)=235
cT(538)=790	cT(780)=75	cT(281)=728	cT(235)=886
cT(790)=33	cT(75)=634	cT(728)=651	cT(886)=372
cT(33)=312	cT(634)=573	cT(651)=220	cT(372)=968
cT(312)=356	cT(573)=308	cT(220)=827	cT(968)=54
cT(356)=422	cT(308)=711	cT(827)=859	cT(54)=542
cT(422)=225	cT(711)=607	cT(859)=262	cT(542)=345
cT(225)=1,010	cT(607)=993	cT(262)=673	cT(345)=657
cT(1,010)=751	cT(993)=44	cT(673)=956	cT(657)=238
cT(751)=488	cT(44)=310	cT(956)=355	cT(238)=884
cT(488)=958	cT(310)=268	cT(355)=330	cT(884)=586
cT(958)=732	cT(268)=633	cT(330)=962	cT(586)=169
cT(732)=1,000	cT(633)=424	cT(962)=911	cT(169)=17
cT(1,000)=547	cT(424)=125	cT(911)=115	cT(17)=353
cT(547)=324	cT(125)=331	cT(115)=681	cT(353)=413
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
cT(660)=429	cT(253)=814	cT(300)=239	cT(1,013)=189
cT(429)=782	cT(814)=61	cT(239)=656	cT(189)=292
cT(782)=788	cT(61)=746	cT(656)=663	cT(292)=131
cT(788)=945	cT(746)=789	cT(663)=608	cT(131)=294
cT(945)=58	cT(789)=186	cT(608)=272	cT(294)=701
cT(58)=741	cT(186)=283	cT(272)=73	cT(701)=191
cT(741)=136	cT(283)=197	cT(73)=688	cT(191)=89
cT(136)=38	cT(197)=290	cT(688)=264	cT(89)=924
cT(38)=667	cT(290)=188	cT(264)=19	cT(924)=499
cT(667)=568	cT(188)=927	cT(19)=897	cT(499)=810
cT(568)=913	cT(927)=676	cT(897)=904	cT(810)=284
cT(913)=414	cT(676)=70	cT(904)=435	cT(284)=221
cT(414)=363	cT(70)=525	cT(435)=326	cT(221)=0

Request ‘Appendix C’ to obtain the entire key.

Appendix D

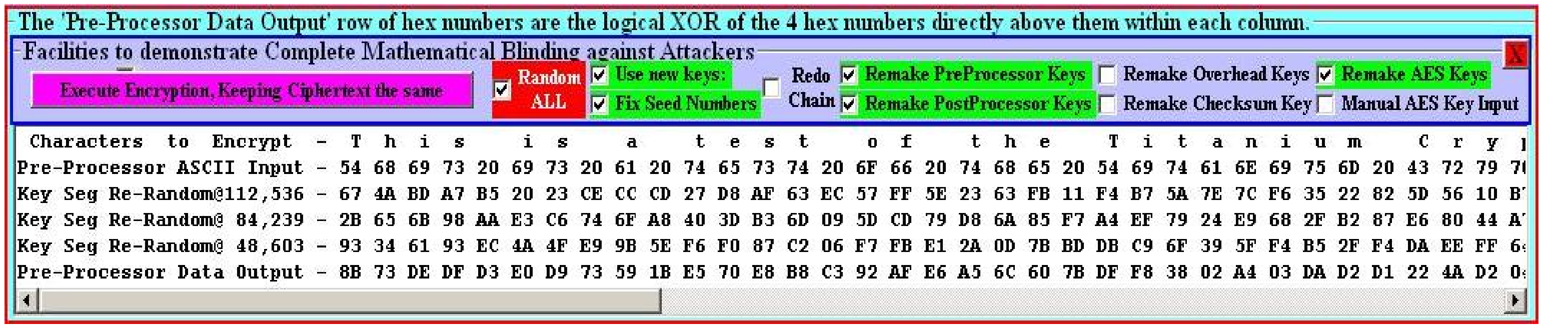
cT(0)=554	cT(491)=383	cT(693)=906	cT(217)=290
cT(554)=125	cT(383)=287	cT(906)=660	cT(290)=364
cT(125)=863	cT(287)=981	cT(660)=1,022	cT(364)=637
cT(863)=511	cT(981)=28	cT(1,022)=546	cT(637)=736
cT(511)=407	cT(28)=730	cT(546)=308	cT(736)=197
cT(407)=889	cT(730)=823	cT(308)=139	cT(197)=242
cT(889)=382	cT(823)=952	cT(139)=158	cT(242)=436
cT(382)=271	cT(952)=717	cT(158)=167	cT(436)=1,001
cT(271)=802	cT(717)=232	cT(167)=309	cT(1,001)=324
cT(802)=519	cT(232)=210	cT(309)=59	cT(324)=560
cT(519)=144	cT(210)=301	cT(59)=878	cT(560)=925
cT(144)=14	cT(301)=104	cT(878)=2	cT(925)=941
cT(14)=251	cT(104)=355	cT(2)=544	cT(941)=933
cT(251)=747	cT(355)=89	cT(544)=582	cT(933)=629
cT(747)=80	cT(89)=532	cT(582)=752	cT(629)=665
cT(80)=893	cT(532)=347	cT(752)=240	cT(665)=923
cT(893)=165	cT(347)=949	cT(240)=890	cT(923)=155
cT(165)=536	cT(949)=196	cT(890)=249	cT(155)=819
cT(536)=746	cT(196)=136	cT(249)=1,011	cT(819)=320
cT(746)=1,021	cT(136)=57	cT(1,011)=926	cT(320)=873
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
. . . . . .	. . . . . .	. . . . . .	. . . . . .
cT(738)=377	cT(195)=395	cT(645)=253	cT(571)=339
cT(377)=325	cT(395)=695	cT(253)=877	cT(339)=27
cT(325)=529	cT(695)=369	cT(877)=632	cT(27)=470
cT(529)=437	cT(369)=378	cT(632)=112	cT(470)=852
cT(437)=304	cT(378)=918	cT(112)=129	cT(852)=455
cT(304)=208	cT(918)=376	cT(129)=540	cT(455)=101
cT(208)=259	cT(376)=356	cT(540)=683	cT(101)=31
cT(259)=225	cT(356)=281	cT(683)=398	cT(31)=220
cT(225)=656	cT(281)=401	cT(398)=241	cT(220)=708
cT(656)=453	cT(401)=586	cT(241)=598	cT(708)=67
cT(453)=896	cT(586)=1,005	cT(598)=766	cT(67)=211
cT(896)=247	cT(1,005)=461	cT(766)=337	cT(211)=709
cT(247)=491	cT(461)=693	cT(337)=217	cT(709)=0

Request ‘Appendix D’ to obtain the entire key.

This page left blank intentionally.

Appendix E

These are screenshots of portions of the Demonstration System showing the first of two examples of when all keys are changed.

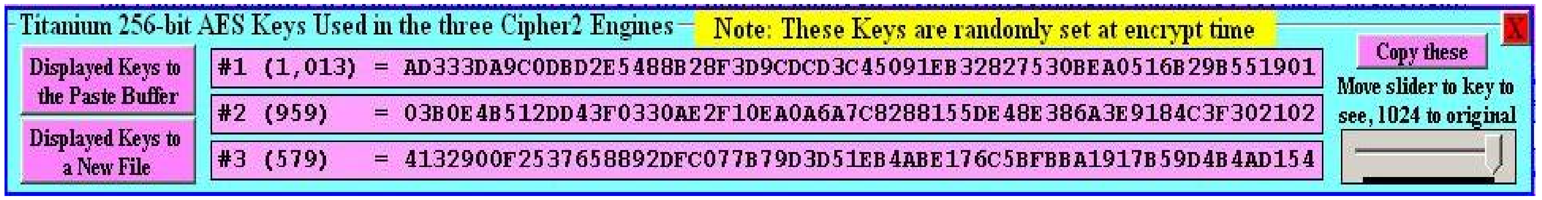


The 'Pre-Processor Data Output' row of hex numbers are the logical XOR of the 4 hex numbers directly above them within each column.

Facilities to demonstrate Complete Mathematical Blinding against Attackers

[✓] Random ALL
[✓] Use new keys:
[✓] Fix Seed Numbers
[ ] Redo Chain
[✓] Remake PreProcessor Keys
[✓] Remake PostProcessor Keys
[ ] Remake Overhead Keys
[ ] Remake Checksum Key
[✓] Remake AES Keys
[ ] Manual AES Key Input

```
 Characters  to  Encrypt  - T  h  i  s     i  s     a     t  e  s  t     o  f     t  h  e     T  i  t  a  n  i  u  m     C  r  y
Pre-Processor ASCII Input - 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6F 66 20 74 68 65 20 54 69 74 61 6E 69 75 6D 20 43 72 79
Key Seg Re-Random@112,536 - 67 4A BD A7 B5 20 23 CE CC CD 27 D8 AF 63 EC 57 FF 5E 23 63 FB 11 F4 B7 5A 7E 7C F6 35 22 82 5D 56 10
Key Seg Re-Random@ 84,239 - 2B 65 6B 98 AA E3 C6 74 6F A8 40 3D B3 6D 09 5D CD 79 D8 6A 85 F7 A4 EF 79 24 E9 68 2F B2 87 E6 80 44
Key Seg Re-Random@ 48,603 - 93 34 61 93 EC 4A 4F E9 9B 5E F6 F0 87 C2 06 F7 FB E1 2A 0D 7B BD DB C9 6F 39 5F F4 B5 2F F4 DA EE FF
Pre-Processor Data Output - 8B 73 DE DF D3 E0 D9 73 59 1B E5 70 E8 B8 C3 92 AF E6 A5 6C 60 7B DF F8 38 02 A4 03 DA D2 D1 22 4A D2
```

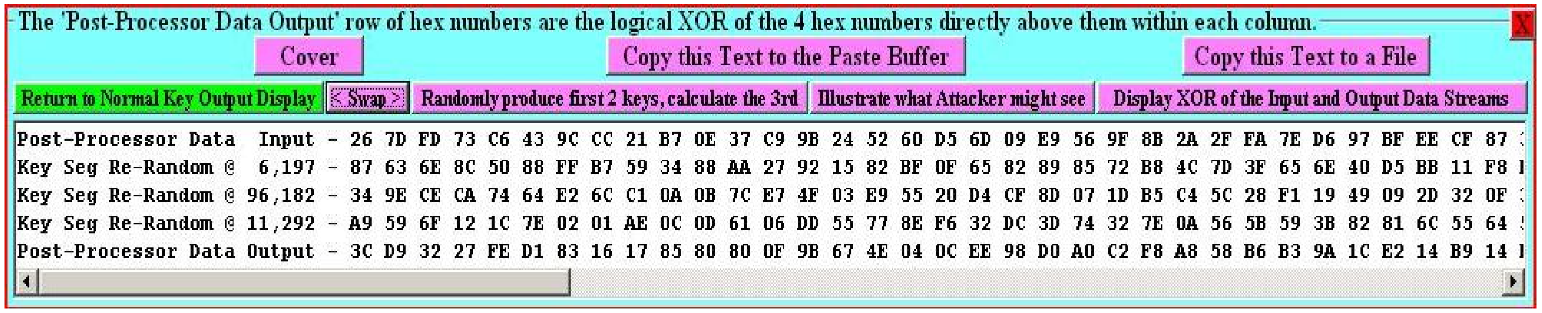


Titanium 256-bit AES Keys Used in the three Cipher2 Engines

Note: These Keys are randomly set at encrypt time

Displayed Keys to the Paste Buffer
Displayed Keys to a New File
Copy these
Move slider to key to see, 1024 to original

```
#1 (1,013) = AD333DA9C0DBD2E5488B28F3D9CDCD3C45091EB32827530BEA0516B29B551901
#2 (959)   = 03B0E4B512DD43F0330AE2F10EA0A6A7C828815 5DE48E386A3E9184C3F302102
#3 (579)   = 4132900F2537658892DFC077B79D3D51EB4ABE176C5BFBBA1917B59D4B4AD154
```

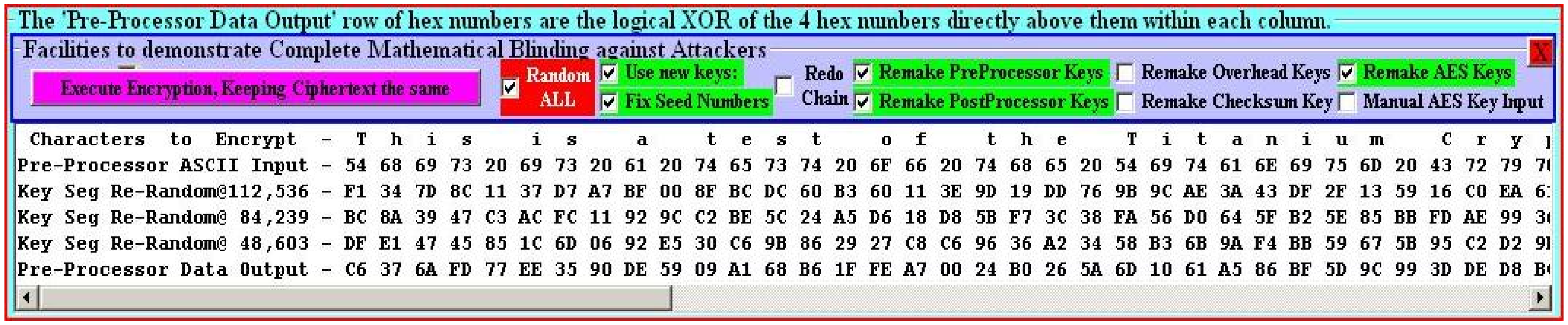


The 'Post-Processor Data Output' row of hex numbers are the logical XOR of the 4 hex numbers directly above them within each column.

Cover
Copy this Text to the Paste Buffer
Copy this Text to a File
Return to Normal Key Output Display
< Swap >
Randomly produce first 2 keys, calculate the 3rd
Illustrate what Attacker might see
Display XOR of the Input and Output Data Streams

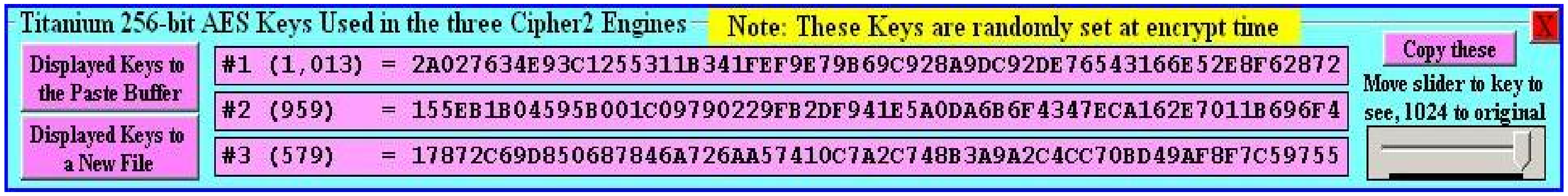
```
Post-Processor Data  Input - 26 7D FD 73 C6 43 9C CC 21 B7 0E 37 C9 9B 24 52 60 D5 6D 09 E9 56 9F 8B 2A 2F FA 7E D6 97 BF EE CF 87
Key Seg Re-Random @  6,197 - 87 63 6E 8C 50 88 FF B7 59 34 88 AA 27 92 15 82 BF 0F 65 82 89 85 72 B8 4C 7D 3F 65 6E 40 D5 BB 11 F8
Key Seg Re-Random @ 96,182 - 34 9E CE CA 74 64 E2 6C C1 0A 0B 7C E7 4F 03 E9 55 20 D4 CF 8D 07 1D B5 C4 5C 28 F1 19 49 09 2D 32 0F
Key Seg Re-Random @ 11,292 - A9 59 6F 12 1C 7E 02 01 AE 0C 0D 61 06 DD 55 77 8E F6 32 DC 3D 74 32 7E 0A 56 5B 59 3B 82 81 6C 55 64
Post-Processor Data Output - 3C D9 32 27 FE D1 83 16 17 85 80 80 0F 9B 67 4E 04 0C EE 98 D0 A0 C2 F8 A8 58 B6 B3 9A 1C E2 14 B9 14
```

Notice that the ‘Pre-Processor ASCII Input’, left image, and the ‘Post-Processor Data Output’, right image, are identical on both pages. <u>All</u> keys and segments used between the two are different.

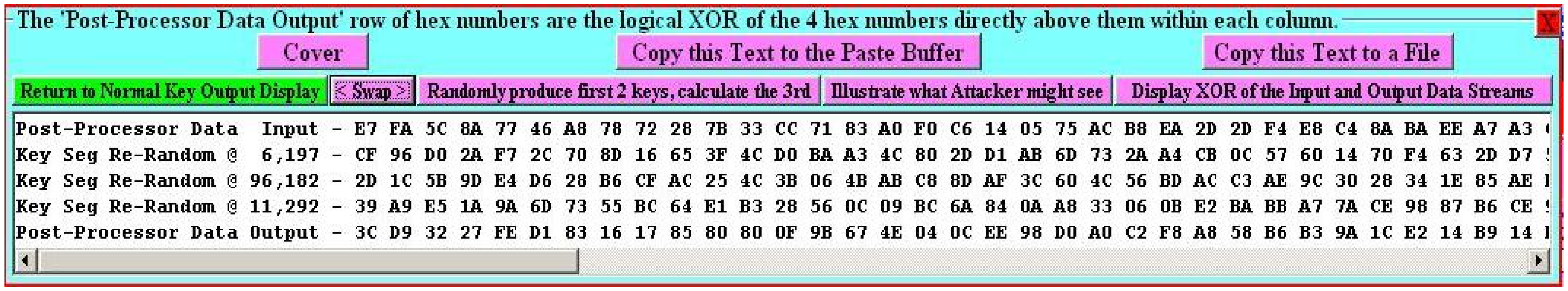

The 'Pre-Processor Data Output' row of hex numbers are the logical XOR of the 4 hex numbers directly above them within each column.
Facilities to demonstrate Complete Mathematical Blinding against Attackers
Execute Encryption, Keeping Ciphertext the same
Random ALL
Use new keys:
Fix Seed Numbers
Redo Chain
Remake PreProcessor Keys
Remake PostProcessor Keys
Remake Overhead Keys
Remake Checksum Key
Remake AES Keys
Manual AES Key Input

[✓] Random ALL · [✓] Use new keys: · [✓] Fix Seed Numbers · [ ] Redo Chain · [✓] Remake PreProcessor Keys · [✓] Remake PostProcessor Keys · [ ] Remake Overhead Keys · [ ] Remake Checksum Key · [✓] Remake AES Keys · [ ] Manual AES Key Input

```
 Characters  to  Encrypt  -  T  h  i  s     i  s     a     t  e  s  t     o  f     t  h  e     T  i  t  a  n  i  u  m     C  r  y
Pre-Processor ASCII Input - 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6F 66 20 74 68 65 20 54 69 74 61 6E 69 75 6D 20 43 72 79
Key Seg Re-Random@112,536 - F1 34 7D 8C 11 37 D7 A7 BF 00 8F BC DC 60 B3 60 11 3E 9D 19 DD 76 9B 9C AE 3A 43 DF 2F 13 59 16 C0 EA
Key Seg Re-Random@ 84,239 - BC 8A 39 47 C3 AC FC 11 92 9C C2 BE 5C 24 A5 D6 18 D8 5B F7 3C 38 FA 56 D0 64 5F B2 5E 85 BB FD AE 99
Key Seg Re-Random@ 48,603 - DF E1 47 45 85 1C 6D 06 92 E5 30 C6 9B 86 29 27 C8 C6 96 36 A2 34 58 B3 6B 9A F4 BB 59 67 5B 95 C2 D2
Pre-Processor Data Output - C6 37 6A FD 77 EE 35 90 DE 59 09 A1 68 B6 1F FE A7 00 24 B0 26 5A 6D 10 61 A5 86 BF 5D 9C 99 3D DE D8
```


Titanium 256-bit AES Keys Used in the three Cipher2 Engines
Note: These Keys are randomly set at encrypt time
Displayed Keys to the Paste Buffer
Displayed Keys to a New File
Copy these
Move slider to key to see, 1024 to original

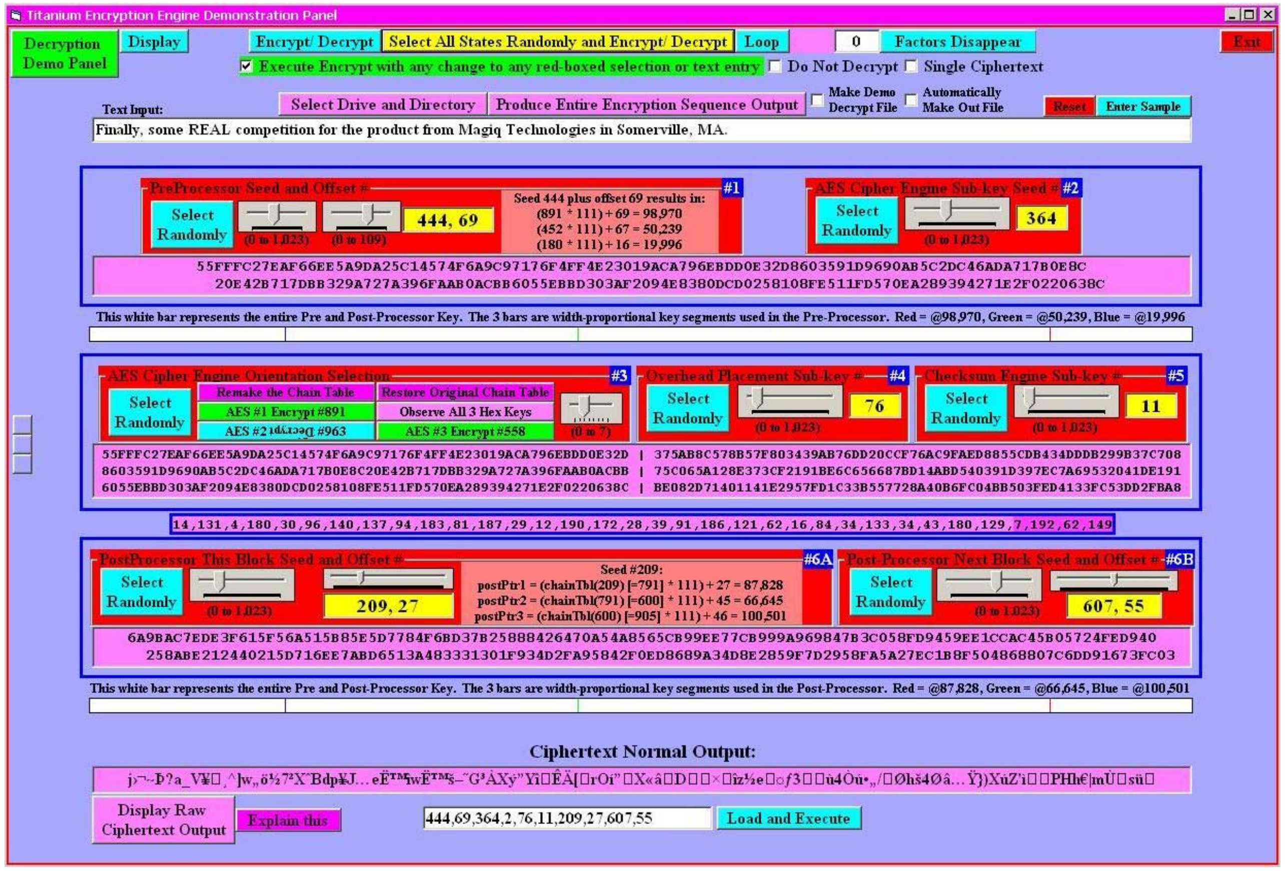
```
#1 (1,013) = 2A027634E93C1255311B341FEF9E79B69C928A9DC92DE76543166E52E8F62872
#2 (959)   = 155EB1B04595B001C09790229FB2DF941E5A0DA6B6F4347ECA162E7011B696F4
#3 (579)   = 17872C69D850687846A726AA57410C7A2C748B3A9A2C4CC70BD49AF8F7C59755
```


The 'Post-Processor Data Output' row of hex numbers are the logical XOR of the 4 hex numbers directly above them within each column.
Cover
Copy this Text to the Paste Buffer
Copy this Text to a File
Return to Normal Key Output Display
< Swap >
Randomly produce first 2 keys, calculate the 3rd
Illustrate what Attacker might see
Display XOR of the Input and Output Data Streams

```
Post-Processor Data  Input - E7 FA 5C 8A 77 46 A8 78 72 28 7B 33 CC 71 83 A0 F0 C6 14 05 75 AC B8 EA 2D 2D F4 E8 C4 8A BA EE A7 A3
Key Seg Re-Random @  6,197 - CF 96 D0 2A F7 2C 70 8D 16 65 3F 4C D0 BA A3 4C 80 2D D1 AB 6D 73 2A A4 CB 0C 57 60 14 70 F4 63 2D D7
Key Seg Re-Random @ 96,182 - 2D 1C 5B 9D E4 D6 28 B6 CF AC 25 4C 3B 06 4B AB C8 8D AF 3C 60 4C 56 BD AC C3 AE 9C 30 28 34 1E 85 AE
Key Seg Re-Random @ 11,292 - 39 A9 E5 1A 9A 6D 73 55 BC 64 E1 B3 28 56 0C 09 BC 6A 84 0A A8 33 06 0B E2 BA BB A7 7A CE 98 87 B6 CE
Post-Processor Data Output - 3C D9 32 27 FE D1 83 16 17 85 80 80 0F 9B 67 4E 04 0C EE 98 D0 A0 C2 F8 A8 58 B6 B3 9A 1C E2 14 B9 14
```

Appendix F

This is the Technology Demonstration Window illustrating some of the display and operational capabilities. Manual and/or individually random selection of seeds and numbers is possible along with an additional window (not shown) that can display multiple block encryption statistics for the engines. Actual size is about double.


Titanium Encryption Engine Demonstration Panel
Decryption Demo Panel
Display
Encrypt/ Decrypt
Select All States Randomly and Encrypt/ Decrypt
Loop
0
Factors Disappear
Exit
Execute Encrypt with any change to any red-boxed selection or text entry
Do Not Decrypt
Single Ciphertext
Select Drive and Directory
Produce Entire Encryption Sequence Output
Make Demo Decrypt File
Automatically Make Out File
Reset
Enter Sample
Text Input:
PreProcessor Seed and Offset #
#1
Select Randomly
(0 to 1,023)
(0 to 109)
444, 69
AES Cipher Engine Sub-key Seed #
#2
Select Randomly
(0 to 1,023)
364
AES Cipher Engine Orientation Selection
#3
Select Randomly
Remake the Chain Table
Restore Original Chain Table
AES #1 Encrypt #891
Observe All 3 Hex Keys
AES #2 Decrypt #963
AES #3 Encrypt #558
(0 to 7)
Overhead Placement Sub-key #
#4
Select Randomly
(0 to 1,023)
76
Checksum Engine Sub-key #
#5
Select Randomly
(0 to 1,023)
11
PostProcessor This Block Seed and Offset #
#6A
Select Randomly
(0 to 1,023)
209, 27
Post-Processor Next Block Seed and Offset #
#6B
Select Randomly
(0 to 1,023)
607, 55
Ciphertext Normal Output:
Display Raw Ciphertext Output
Explain this
Load and Execute

☑ Execute Encrypt with any change to any red-boxed selection or text entry ☐ Do Not Decrypt ☐ Single Ciphertext

☐ Make Demo Decrypt File ☐ Automatically Make Out File

Text Input:

Finally, some REAL competition for the product from Magiq Technologies in Somerville, MA.

Seed 444 plus offset 69 results in:
(891 * 111) + 69 = 98,970
(452 * 111) + 67 = 50,239
(180 * 111) + 16 = 19,996

```
55FFFC27EAF66EE5A9DA25C14574F6A9C97176F4FF4E23019ACA796EBDD0E32D8603591D9690AB5C2DC46ADA717B0E8C
20E42B717DBB329A727A396FAAB0ACBB6055EBBD303AF2094E8380DCD0258108FE511FD570EA289394271E2F0220638C
```

This white bar represents the entire Pre and Post-Processor Key. The 3 bars are width-proportional key segments used in the Pre-Processor. Red = @98,970, Green = @50,239, Blue = @19,996

```
55FFFC27EAF66EE5A9DA25C14574F6A9C97176F4FF4E23019ACA796EBDD0E32D  |  375AB8C578B57F803439AB76DD20CCF76AC9FAED8855CDB434DDDB299B37C708
8603591D9690AB5C2DC46ADA717B0E8C20E42B717DBB329A727A396FAAB0ACBB  |  75C065A128E373CF2191BE6C656687BD14ABD540391D397EC7A69532041DE191
6055EBBD303AF2094E8380DCD0258108FE511FD570EA289394271E2F0220638C  |  BE082D71401141E2957FD1C33B557728A40B6FC04BB503FED4133FC53DD2FBA8
```

```
14,131,4,180,30,96,140,137,94,183,81,187,29,12,190,172,28,39,91,186,121,62,16,84,34,133,34,43,180,129,7,192,62,149
```

Seed #209:
postPtr1 = (chainTbl(209) [=791] * 111) + 27 = 87,828
postPtr2 = (chainTbl(791) [=600] * 111) + 45 = 66,645
postPtr3 = (chainTbl(600) [=905] * 111) + 46 = 100,501

```
6A9BAC7EDE3F615F56A515B85E5D7784F6BD37B25888426470A54A8565CB99EE77CB999A969847B3C058FD9459EE1CCAC45B05724FED940
258ABE212440215D716EE7ABD6513A483331301F934D2FA95842F0ED8689A34D8E2859F7D2958FA5A27EC1B8F504868807C6DD91673FC03
```

This white bar represents the entire Pre and Post-Processor Key. The 3 bars are width-proportional key segments used in the Post-Processor. Red = @87,828, Green = @66,645, Blue = @100,501

Ciphertext Normal Output:

j›¬-Þ?a_V¥□¸^]w„ö½7²X˜Bdp¥J…eË™¶wË™š–˜G³ÀXý"Yì□ÊÄ[□rOí"□X«â□D□□×□îz½e□○ƒ3□□ù4Òù•„/□Øhš4Øâ…Ÿ))XúZ'ì□□PHh€|mÙ□sü□

444,69,364,2,76,11,209,27,607,55