CHARACTER SET PRESERVING ENCRYPTION MODE

Proposed Mode:	Character Set Preserving Encryption Mode (CSPEM).
Submitted By:	Gary S. Sarasin
Phone:	541-345-4334
Fax:	541-345-6818
Address:	Prime Factor, Inc.
	4725 Village Plaza Loop
	Suite 100
	Eugene Oregon 97401
E-Mail:	gary.sarasin@primefactors.com
Date:	11/18/2011
Inventor/Owner:	n/a

This proposal is a generalization of FIPS 74 Section 8 to allow for any symmetric key block cipher algorithm.

MODE SPECIFICATION

The purpose of this mode is to provide a method for mapping characters within a set onto itself without expansion. For example an account number consisting of 16 digits 0-9 and producing cipher output of 16 digits 0-9. Or encrypting an alphabetic name consisting of characters A-Z and a-z and producing cipher output consisting of A-Z a-z and of the same length.

CONTENTS

1. INTRODUCION

In certain applications it is desirable that only valid plaintext characters appear as cipher. For example, special control characters are often used to designate headers, synchronization bits, and the beginning or ending of ciphertext. If control characters can also appear randomly as cipher, then it is difficult to distinguish between intended plaintext control characters and cipher. One solution is to stuff redundant characters into the transmitted data (to indicate control characters) thereby adding additional overhead. Also, in situations where cipher characters are to be printed, no unprintable characters can be permitted in cipher. A character is defined to be valid if it is not used as a control character and invalid if it may be used for control. For example, a character which indicates a carriage return is invalid. A problem arises since presently defined block encryption modes of operation map K-bit characters onto K-bit characters. If the number of members in a valid plaintext character set is not a power of two, then invalid characters will appear in cipher. A modification is proposed which permits the encryption of a character set of arbitrary size onto itself. Therefore, valid characters are always encrypted to valid characters. The modification is discussed as it applies to specific examples as well as to the general problem.

1.1 Example I (Digits)

In this subsection we will consider a solution for the problem of enciphering digits onto digits. Later subsections will apply the solution to other cases.

Consider Triple DES (or any encryption algorithm with a 64 bit output) as used in the Cipher Feedback (CFB) mode. K bits of the 64-bit Triple DES output are exclusive OR'ed with a K-bit plaintext character to form cipher. Suppose that one desires to encipher the digits, 0 through 9. Four-bit characters are required to represent the 10 digits, the first 10 character representations correspond to the digits, and the remaining 6 are invalid. (See table 1.1.) Even if only valid plaintext characters are enciphered, Triple DES in the CFB mode will produce cipher characters which may be invalid.

1.1.1 Solution

Let FO denote the bits of the final block encryption output (see note 1 below). Instead of exclusive

OR'ing the first four bits of FO with the four-bit plaintext character, add the two values modulo 10 (base 10). The modulo 10 sum of the digits A and B is the remainder of A + B divided by 10. X is congruent to Y modulo 10 (X = Y modulo 10) if and only if X - Y = 10m for some integer m. Thus A + B is congruent to a valid cipher character. For example, suppose that FO = 1101 and that one wished to encipher 0011. Since 0011 + 1101 = 10000 = 0110 modulo 10, 0110 is the resultant cipher. The input register to the Triple DES function will contain exactly 16 valid cipher characters, so 10^{16} distinct input register combinations are possible.

1.1.2 Decryption

The decryption algorithm is similar to the encryption algorithm except that the first FO character is subtracted modulo 10 from cipher to form plaintext. Using the values from the previous example, 0110 - 1101 = -0111 = 0011 modulo 10. The algorithms are inverses of each other because the FO generated by the decrypting device is the same as the FO generated by the encrypting device.

Let P be a valid plaintext character and G be the first character of FO. Let C be the corresponding cipher character.

```
C    =  (P+G)modulo 10.
C    =  P+G-10m.
P    =  C-G+10m.
P    =  (C - G) modulo 10.
```

Decryption is unique even though the first character of FO may not be an integer modulo 10 (i.e., a base 10 digit).

Since G is not necessarily a valid character, there is a bias on cipher which depends on the plaintext. If the plaintext is flat (randomly generated), for example, the cipher is also flat, but if several plaintext zeros are encrypted there is a bias towards zero through five in the cipher. This problem can be overcome by selecting G from FO in a manner which virtually assures that G s evenly distributed over the digits. Subsection 1.5 provides a solution which will render an insignificant bias in most applications.

One might consider encrypting the digits as follows: Exclusive OR(XOR) the first FO character with the plaintext character and then use the result modulo 10. The trouble with this solution is that it does not decrypt correctly. Suppose that FO = 0101 and that plaintext is 1000. 0101 XOR 1000 = 1101 = 0011 modulo 10. Therefore 0011 would be taken as cipher. But 0011 XOR 0101 = 0110 != 1000 modulo 10. Decryption would not produce the correct plaintext.

1.2 Example II (Alphanumerics)

The USA Standard Code for Information Interchange (ASCII), with b7 as the high-order bit and b1 as the low-order bit, appears in table 1.2. Suppose one desires to encipher the 96 characters whose binary representations range from 0100000 to 1111111. These 96 characters may be mapped into the integers modulo 96 by subtracting 0100000 from their ASCII representations. Let symbolize this mapping. Then

```
SP = 0100000  <--->  0000000  = 0,
!  = 0100001  <--->  0000001  = 1,
.
.
.
DEL = 1111111  <--->  1011111  = 95.
```

If we wish to encipher the character, n, and the first character of FO is }, then cipher is formed using the following equations.

```
n = 1101110  <--->  1001110.
} =  1111101.
cipher  <--->  (1001110 + 1111101) modulo 96 = 0001011.
cipher = (0001011 + 0100000) = 0101011 = +.
```

One must remember to translate the plaintext to an integer modulo 96 before addition and then to translate the result back to a valid character after addition. Nine characters may be held in the 64-bit input register. The number of possible input register settings is, $96^9 = 6.92 \text{ X } 10^{17}$. Note that since the length of a character (7 bits) does not evenly divide the length of the input register (64 bits) the first bit of the input register is always fixed to zero.

1.3 Example III (General Solution)

The proposed method may be used as a general solution.

For the purposes of describing this process any approved symmetric block cipher can be used. The following designations will be used:

> Let LC designate the number of possible cipher output characters from a given symmetric encryption algorithm.
>
> Let LB designate the number of output bits from a given symmetric encryption algorithm.
>
> For example, Triple DES will produce 64 bits of output with 16 bytes of possible cipher characters. Thus LC = 16 and LB = 64. AES will produce 128 bits of output with 32 bytes of possible cipher characters. Thus LC = 32 and LB = 128.

Suppose one has an N character alphabet. Let K be such that $2^{K-1} < N < 2^K$. Then one must be satisfied that $N^{[LB/K]}$ (the number of possible input register combinations) is sufficiently large where [X] is the greatest integer < X. For security reasons, with a symmtric algorithm where LB is 64, it is recommended that $N^{[64/K]}$ be at least $2^{48} = 10^{14.4}$.

If the characters are contiguous, then a simple translation will map them onto the integers modulo N before addition is performed; and after addition, the inverse will map back to valid characters (as previously discuss- in 1.2). If the characters are not contiguous, then conversion tables can be used to make the transformations to and from the integers modulo N. Consider the USA Standard Code (ASCII) presented in table 1.2. Suppose that the only valid characters are: A, B, C, F, H, I, M, N, O, P, U, V, and Z. In this case N=13 and K=4. The number of possible inputs at each encryption is, for example when LC=16, $13^{16} = 6.65 \times 10^{17}$.

If the set of possible characters is not too large, then for each possible character the conversion table will list its modulo N value, if it is valid, or an invalid indicator, if it is invalid. This table could be used to determine whether or not a character is valid as well as to map it to its corresponding modulo N value. (See table 1.3.)

If the character set is too large other possibilities exist. A conversion table could be made which just covers the range from the first to the last valid character. In this case characters which are found to be less than 1000001 and greater than 1011010 are invalid. For the others, subtract 1000000 and use the result as an index to the table. (See table 1.4.)

Another possibility is to store the binary representations and modulo N values for only valid characters. Searching, hashing, or some other method must be used to find the correct location of the character being looked up. (See table 1.5.)

Once the modulo N sum of the plaintext and K bits of FO have been found, another table the inverse of table 1.5) is required to convert back to the binary representation. This table need only have one entry for each integer modulo N. The integer modulo N is incremented and the result is used as an index to find the corresponding cipher character.

1.4 Solution for Plaintext Bias

When the ciphertext bias produced by the use of invalid characters from FO is unacceptable, only valid characters should be selected from FO. Consider the example where the digits are to be mapped onto themselves. The first four bits of FO will be valid with probability 10/16 (for LC = 16). If the first four bits form a valid character they may be used for the addition to plaintext. If they are invalid consider the second four bits. If the second four bits form a valid character they may be added to the plain text to form cipher. Repeat this procedure until either a valid cipher is formed or until all LC four-bit characters of FO have been examined and each one is found to be invalid. The latter event, called the default condition, will occur with probability $(6/16)^{16} = .000000153$ when LC = 16. In this case the value to be added to plaintext can be arbitrarily selected as 1001 (9). A new Fo is generated for each character to be enciphered.

If the bits of FO are statistically random then, as long as the default condition is not employed, the cipher should also be random. The default condition is definitely nonrandom, but since it should only occur with probability .000000153 (when LC=16) the ciphertext will be near random. In fact, frequency counts would have to be done on very large amounts of data before the slight bias would be detectable. Using a Chi-square test would require data from more than 10^{13} encryptions before one could expect to detect nonrandomness. Of course, if the plaintext is flat random, no bias will be found on cipher.

In general if one has a character set of N members and K is such that $2^{K-1} < N < 2^{K}$, then one must be satisfied that $((2^{K}-N)/2^{K})^{[LB/K]}$ where [X] is the greatest integer < X, is sufficiently small.

Table 1.1 Digit to Character Conversion Table

	0	<--->	0000	
	1	<--->	0001	
	2	<--->	0010	
valid	3	<--->	0011	valid
digits	4	<--->	0100	characters
	5	<--->	0101	
	6	<--->	0110	
	7	<--->	0111	
	8	<--->	1000	
	9	<--->	1001	
	10	<--->	1010	
	11	<--->	1011	
invalid	12	<--->	1100	invalid
digits	13	<--->	1101	characters
	14	<--->	1110	
	15	<--->	1111	

Tables 1.2 and 1.3 are not available at this time.

Table 1.4 Valid Character Range

Entry	Binary		Modulo N Value
1	1000001(A)	<-->	0 (valid character 0 modulo 13)
2	1000010(B)		1
3	1000011(C)		2
4	1000100(D)		17
.	.		.
25	1011001(Y)		17
26	1011010(Z)		12

Table 1.5 Valid Characters Only

Entry	Binary	Modulo N Value
1	1000001(A)	0
2	1000010(B)	1
3	1000011(C)	2
4	1000110(F)	3
5	1001000(H)	4
.	.	.
13	1011010(Z)	12

SUMMARY OF PROPERTIES

Security Function:	Encryption
Keying Material:	1 key
Memory Requirements:	Minimal
Message Length Requirements:	Arbitrary Length
Ciphertext Expansion:	None

TEST VECTORS

Algorithm:	112 bit Triple DES
Character Set:	0123456789 (length is 10 characters, either ASCII or EBCDIC)
IV:	F9467D313F80EF51 (length is 8 binary bytes)
Key:	0123456789ABCDEF0FEDCBA9876543210 (length is 16 binary bytes)
Clear Text:	1234123412341234 (16 characters, either ASCII or EBCDIC)
Cipher Text:	9357050596492460 (16 characters, either ASCII or EBCDIC)

Algorithm:	168 bit Triple DES
Character Set:	0123456789 (length is 10 characters, either ASCII or EBCDIC)
IV:	F9467D313F80EF51 (length is 8 binary bytes)
Key:	F5013C75F565266C66DE767FEB28DABC6146C083032A95B1 (length is 24 binary bytes)
Clear Text:	1234123412341234 (16 characters, either ASCII or EBCDIC)
Cipher Text:	1662080857783336 (16 characters, either ASCII or EBCDIC)

Algorithm:	128 bit AES
Character Set:	0123456789 (length is 10 characters, either ASCII or EBCDIC)
IV:	F9467D313F80EF51C55AF95F2CEB1853 (length is 16 binary bytes)
Key:	0123456789ABCDEF0FEDCBA9876543210 (length is 16 binary bytes)
Clear Text:	1234123412341234 (16 characters, either ASCII or EBCDIC)
Cipher Text:	6373530456852566 (16 characters, either ASCII or EBCDIC)

Algorithm:	192 bit AES
Character Set:	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrs tuvwxyz0123456789 (length is 62 characters either ASCII or EBCDIC)
IV:	F9467D313F80EF51C55AF95F2CEB1853 (length is 16 binary bytes)
Key:	6F1D0CD5D4368DE296593D112E567EEA8298F0197C024C30 (length is 24 binary bytes)

Clear Text:	TestThisString4Me2AndWeWillSee (30 characters, either ASCII or EBCDIC)
Cipher Text:	WfHmQ4osZTr8VyFe4vC8dblf5oA9nP (30 characters, either ASCII or EBCDIC)

Algorithm:	256 bit AES
Character Set:	0123456789 (length is 10 characters, either ASCII or EBCDIC)
IV:	F9467D313F80EF51C55AF95F2CEB1853 (length is 16 binary bytes)
Key:	6236F0CE491882BB53003E4D890DACA8
	E317C91552A3B0D353C1C716B318C908 (length is 32 binary bytes)
Clear Text:	1234567890987654321 (19 characters, either ASCII or EBCDIC)
Cipher Text:	3496560310727509672 (19 characters, either ASCII or EBCDIC)

Algorithm:	256 bit AES
Character Set:	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrs
	tuvwxyz0123456789 (length is 62 characters either ASCII or EBCDIC)
IV:	F9467D313F80EF51C55AF95F2CEB1853 (length is 16 binary bytes)
Key:	6236F0CE491882BB53003E4D890DACA8
	E317C91552A3B0D353C1C716B318C908 (length is 32 binary bytes)
Clear Text:	TestThisString4Me2AndWeWillSee (30 characters, either ASCII or EBCDIC)
Cipher Text:	H70TzQrvLX7Mar9sJcTUCnMbhQb4oF (30 characters, either ASCII or EBCDIC)
	(Value at position 3 is a numeric 0).

PERFORMANCE ESTIMATES

Performance is determined by the selected symmetric algorithm. In general this method will perform one block cipher for each encrypted character of output.

Intellectual Property Statements/Aggreements/Disclosures

None.