FORMAT-CONTROLLING ENCRYPTION

USING

DATATYPE-PRESERVING ENCRYPTION

Ulf T. Mattsson,
Chief Technology Officer,
Protegrity Corp.

ABSTRACT

Datatype-Preserving Encryption (DTP) enables encryption of values within a certain character set into ciphertext restricted to the same set, while still keeping data length. This is in contrast to conventional block cipher modes which produce binary data, i e each encrypted character may have an arbitrary value, possibly outside the original character set, often accompanied with a length expansion caused by padding.

Format-Controlling Encryption (FCE) is an extension to DTP, for which data length still is kept, but the output character range is allowed to be larger, though not covering the range of all possible values (i e binary data). With FCE it's possible to handle certain DTP limitations, like limited key rotation and integrity support.

INTRODUCTION

When encrypting a requirement is sometimes that the encrypted output should have the same data type as the cleartext. The data may be stored in legacy systems, for which modifications to the data format would be cumbersome. Applications using the existing data would have to be modified, an often costly, and if the source code isn't available even impossible, process. The target in such cases is to have application-transparent encryption.

Transparency may be handled with certain tricks like hiding the encryption inside database views/triggers or at the file system layer. But there will always be situations where this is not possible or appropriate. The data store may simply not support such tricks, or the encryption will become too local when end-to-end encryption is the target.

Format requirements for the encrypted output often arise within the area of Personally Identifiable Information (PII) [5], especially numeric PII. Since there are a growing number of regulations concerning PII, there is also a growing need for encryption within this area.
One major regulation is the Payment Card Industry Data Security Standard (PCI-DSS) [2], which requires credit card number (CCN) protection. When using encryption as the protection option, it must be based on *"strong cryptography"*. This is in the PCI glossary [3] defined as *"…cryptography that is extremely resilient to cryptanalysis. That is, given the cryptographic method (algorithm or protocol), the cryptographic key or protected data is not exposed…"*. For secret-key encryption, this means at least 80 bits as effective key size according to [4]. Regular key rotation must also be applied for PCI-DSS.

One idea of how to achieve transparent encryption was presented already back in 1997 by Harry Smith and Michael Brightwell in *"Using Datatype-Preserving Encryption to Enhance Data Warehouse Security"*

[1]. Their presentation is the base for Datatype-Preserving Encryption (DTP), as specified in this document.

In their presentation Smith and Brightwell describe how to encrypt data within a certain character set into ciphertext restricted to the same set, using some standard block cipher, claiming a security level as strong as the block cipher. But they also mention some issues with this encryption:

- possible misinterpretation of encrypted data, i e interpreting encrypted data as cleartext
- consistent encrypted output, i e always the same output for the same input

Some other issues may be added to this list:

- no support for key generation identifier, which may be necessary to support key rotation, as for instance required by PCI-DSS
- no integrity support, a modification to the encrypted value will not be detected

To handle these DTP shortcomings, Format-Controlling Encryption (FCE) may be used. FCE will always keep the data length just as DTP does, but FCE will also have different input and output character sets. This will prevent from misinterpretation of encrypted data as cleartext. If also extending the output set with additional characters, compression of the input data is possible. This enables storage of additional data, like an integrity value or key generation identifier, with unchanged data length.

The use of FCE implies it's possible to have a different output character set. When this isn't possible, the use of DTP is still mandated with the limitations this has.

OVERVIEW

DTP is based on [1], which encrypts using a combination of different pre-scrambling steps followed by the actual encryption. Pre-scrambling is performed to get randomized output for similar input. Without any pre-scrambling the output would be the same up to the first differentiating character, and this character would keep the relative difference. This may be seen in this sample encryption without pre-scrambling:

```
Cleartext           Ciphertext
----------------    ----------------
1111111111111111    3038556804191683
1111111111111112    3038556804191684
1111111111111113    3038556804191685
1111111111111114    3038556804191686
1111111111111115    3038556804191687

1111111111111112    3038556804191684
1111111111111121    3038556804191699
1111111111111211    3038556804191729
1111111111112111    3038556804192124
1111111111121111    3038556804100115
1111111111211111    3038556804217439
1111111112111111    3038556805000660
1111111121111111    3038556813314858
1111111211111111    3038556933988458
1111112111111111    3038557273849845
1111121111111111    3038566888710029
1111211111111111    3038693962325175
1112111111111111    3039117820592062
1121111111111111    3045848497050994
1211111111111111    3121031030187252
2111111111111111    4505000362594489
```

The 13 steps for encryption as specified in [1] are:

Initial tasks:

1. *Choose an encryption key with enough bits for the encryption algorithm key, encryption algorithm initial value and any basic processing stages.*
2. *Choose a suitable alphabet to support the datatype of the data to be encrypted.*
3. *Shuffle the alphabet according to a scheme based on the key.*

For each encrypted field:

4. *Scan the input buffer for characters which are not included in the chosen alphabet. Move all invalid characters unchanged to their corresponding positions in the ciphertext output buffer.*
5. *Move the index values of all valid characters to adjacent positions in a work buffer.*
6. *Add position-sensitive offsets according to a key-dependent scheme.*
7. *Shuffle the work buffer positions according to a data-dependent scheme.*
8. *"Ripple" the work buffer by calculating a key-based starter number and modularly adding pairwise from left to right then from right to left.*
9. *Set the cipher-feedback initial value using the chosen key.*
10. *Calculate the modular sum of the first work buffer position and the lowestorder DES output byte. Store this value in a second work buffer.*
11. *Obtain a new DES initial value by moving the DES output to the input, shifted one byte to the left, and shifting the work buffer value into the lowestorder position.*
12. *Repeat steps 9 through 11 using successive work buffer index values until all of the data is transformed.*
13. *Replace the transformed index values by their corresponding character equivalents and store them in the open ciphertext positions.*

Here steps 3, 6, 7 and 8 concern pre-scrambling of the data, and the actual encryption is performed in steps 9, 10, 11 and 12. Remaining steps principally concern alphabet conversions.

The pre-scrambling as specified in [1] may be considered a bit limited.

Steps 3, 6 and the starter numbers in 8 are only based on the key value. Parameters are typically generated *"based on a portion of the key"*, or *"by hashing the encryption key"*. This means they will be fixed for a certain key, and hence provide limited scrambling. It also assumes the key value is available in cleartext, which may not be the case if for instance using a Hardware Security Module (HSM).
To handle this an option would be to use values based on encryption with the key instead. This would solve key availability, but the cipher input would still be limited. The input may include the alphabet and data sizes, but these would still be constant if encrypting a 16-digit CCN with numeric characters.

Step 7 is supposed to shuffle based on some *"permutation-invariant property of the index values, such as a sum or exclusive-or, of all values".* But if for instance using the sum over all values, this would still provide limited scrambling. A 16-digit CCN has a maximum sum of 16x9 = 144; hence there will be 145 different permutations. Though certainly better than the constant values for the other steps, it would still be a bit limited. Using exclusive-or would be even more limiting; numeric index values 0 – 9 affect 4 bits; there will only be 16 permutations from this. The step 7 problem is that shuffling based on a "*permutation-invariant property"* can't consider the order of the characters.

Step 8 will provide scrambling output dependent on the character order. But for certain combinations the scrambling will be limited. If denoting the last three characters of a ripple buffer as `{x y z}`, these will ripple other characters based on the formula `3x + 2y + z`. Keeping this sum constant means the other characters will only be affected by a constant. This may be seen in the table below using `8` as the sum; the first 6 digits present a consistent pattern, though rippling is applied:

Original	Left-right	Right-left
111111 210	123456 899	764172 689
111111 121	123456 790	764172 690
111111 113	123456 781	764172 691
111111 105	123456 772	764172 692

In total this means patterns may be seen, even if applying all pre-scrambling steps as specified in [1]. For instance, using alphabet shuffling based on alphabet size, offset values based on data size, data shuffling based on sum, and starter values based on data size, encryption steps as below may be seen. Here a Knuth shuffle [11] with the sum as the seed for the random number generator has been used.

Original data	Shuffle alphabet	Position offsets	Shuffled data
1111111111111112	3333333333333331	7359292332902802	3927892020232953
1111111111111121	3333333333333313	7359292332902884	2389324897092235
1111111111111211	3333333333333133	7359292332902604	3927694020232953
1111111111112111	3333333333331333	7359292332900804	3927894020230953
1111111111121111	3333333333313333	7359292332982804	2389324097892235
1111111111211111	3333333333133333	7359292332702804	3727894020232953
1111111112111111	3333333331333333	7359292330902804	3927894020032953
1111111121111111	3333333313333333	7359292312902804	3927894020212953
1111111211111111	3333333133333333	7359292132902804	3927894020232951
1111112111111111	3333331333333333	7359290332902804	3907894020232953
1111121111111111	3333313333333333	7359272332902804	3927894020232753
1111211111111111	3333133333333333	7359092332902804	3927894000232953
1112111111111111	3331333333333333	7357292332902804	3927874020232953
1121111111111111	3313333333333333	7339292332902804	3927894020232933
1211111111111111	3133333333333333	7159292332902804	1927894020232953
2111111111111111	1333333333333333	5359292332902804	3925894020232953

Rippled data L-R	Rippled data R-L	Encrypted data
7685324466813270	8157297393798536	5466899531973352
6976915329980250	7125909419013316	3145077243628877
7685104466813270	4713877393798536	7898982838706069
7685326688033270	0379419379118536	8963880825852232
6976915541980250	3781565051013316	4177213594480738
7463104466813270	8171877393798536	5475705811611207
7685326688813270	6935075935798536	0227293794208742
7685326688013270	8157297157998536	5466899002583401
7685326688035490	6935075935774956	0227293794279043
7663104466813270	0371877393798536	8962717177224037
7685326688035270	2591631591330536	6345311177086056
7685326666813270	2591631593798536	6345311170786559
7685304466813270	6935077393798536	0227292319115280
7685326688035470	4713853713552736	7898950850628806
5463104466813270	6171877393798536	0116394877775763
7683104466813270	2591877393798536	6345272074192100

Some patterns may be seen in the output, for instance `6345311177086056`, `6345311170786559` and `6345272074192100`. It's not clear if this would be possible to use in a real-world attack, though. Adding more input to the data shuffling in step 7, like using both sum *and* exclusive-or, would also mean increased output randomization; the number of possible permutations would be increased.

DTP, however, uses a somewhat different type of pre-scrambling. Rippling is still performed, but the steps 3, 6 and the step 8 starter numbers, have been exchanged with a single step that produces offset values based on encryption of all characters. The target is to have pre-scrambling dependent on the whole data value, including the order of the characters. Step 7 may also be used in DTP, but is regarded optional.

DTP pre-scrambling is accomplished by splitting the input data into two halves. First the left half is enciphered. The cipher output is then used as position-dependent offset values for the right half. Then the resulting right half is enciphered, and the output is applied to the left half. Finally the rippling is performed, using 0 as the starter number.

In total DTP encryption with pre-scrambling may look like:

```
Original data      Cipher left half   Cipher right half
----------------   ----------------   ----------------
1111111111111112   1111111175085456   9659453075085456
1111111111111121   1111111175085465   0120078375085465
1111111111111211   1111111175085555   0400877375085555
1111111111112111   1111111175086455   9864022275086455
1111111111121111   1111111175095455   5824450275095455
1111111111211111   1111111175185455   9745238675185455
1111111112111111   1111111176085455   4734861576085455
1111111121111111   1111111185085455   9239497085085455
1111111211111111   1111111298972374   0073288498972374
1111112111111111   1111112120547302   5150886620547302
1111121111111111   1111121102749558   6174209002749558
1111211111111111   1111211164545455   2160865164545455
1112111111111111   1112111125845793   9728232525845793
1121111111111111   1121111107866556   1603473707866556
1211111111111111   1211111144850806   4334833844850806
2111111111111111   2111111114031565   9530738314031565

Rippled data L-R   Rippled data R-L   Encrypted data
----------------   ----------------   ----------------
9509381183316051   3499079879632661   1459946433649857
0133308183316061   7763077980743771   5906559126014409
0444296961194949   1173978237656239   9342837959299762
9737791305539383   2363690966163413   0682029252042910
5359388072216050   4961291336421550   2883982482212076
9605708416750494   5600588065927734   3608423411567804
4148623851194838   5106820794323918   3182642444990687
9143763316649383   6762926309373413   4384369006493911
0070208219857071   7770088087805881   5910971502547096
5611973911607002   8376569676599222   6791789383696374
6748009991821619   0473555678797609   8059726692381241
2399738959483727   5301252450179697   3389439501312968
9686813805372981   4591576355070891   2433783773693442
1770414118628384   5470065109153524   3448798356298641
4704258604277551   7366205711758161   5587292422822164
9477475893367283   2392514912960313   0613613614121230
```

The encrypted output for DTP doesn’t present the patterns as seen with the [1] pre-scrambling above.

The data shuffling in [1], step 7, is an optional step for DTP, and not used above. Normally shuffling isn’t used; each operation will in the end steal performance. An exception is however if varying-length characters are used, as described in the section *DTP Encryption with Varying-length Character Sets*.

SPECIFICATION

The DTP encryption steps as described in each sub-section of this specification are:

1. Select algorithm and key
2. Select input and output alphabets
3. Put index values of characters to be encrypted into a work buffer
4. Add position-sensitive offsets according to a data-dependent scheme
5. Shuffle the work buffer according to a data-dependent scheme
6. Ripple data by modularly adding pairwise from left to right then from right to left
7. Select initial value for internal cipher
8. For each index value, encrypt internal buffer, set ciphertext using a modular addition over index value and encrypted output, and update internal buffer using plaintext feedback
9. Output character representation of the encrypted index values

DTP is based on [1], and most steps have corresponding steps in [1]. If comparing to [1] steps 1 and 2 are the same, step 3 corresponds to [1].4 and [1].5, step 4 is a DTP specific step somewhat similar to [1].6, step 5 is [1].7, steps 6 and 7 are [1].8 and [1].9 respectively, step 8 corresponds to [1].10, [1].11 and [1].12, and step 9 is finally the same as [1].13. The internal processing for some steps, principally the pre-scrambling steps, is however a bit different.

As from [1], decryption means *"performing the inverse of each transformation in the reverse order"*.

1. SELECT ALGORITHM AND KEY

DTP is normally based on standard block ciphers. It may be regarded as a complementary mode of operation for these ciphers, as compared to standard modes like CFB (Cipher Feedback) and OFB (Output Feedback) [6]. However, since only the encrypt function of the block cipher is used, it's also possible to use a one-way cipher with a secret key for DTP.

Similar to CFB and OFB, DTP is a stream cipher mode. The cipher in question generates a stream of pseudorandom data used in the encryption process. But where others produce ciphertext by XOR-ing this data with some plaintext, DTP produces the ciphertext using a modular addition. The feedback for the cipher input is also different; DTP uses plaintext feedback.

Selecting algorithm for DTP means selecting the internal cipher algorithm. This must be a strong cipher, as requested by PCI-DSS. In [1] the algorithm used is DES, with a claim that *"recovering the string, i1 i2 i3 i4 i5 … in, from the transformed string, z1 z2 z3 z4 z5 … zn, without knowledge of the key, K, is as difficult as breaking the DES algorithm itself."* However, DES has been redrawn [7] since then, and isn't supported for DTP encryption.

One of the following alternatives should be used for the DTP internal cipher:

- AES [8] with 256-bit key.
- AES with 128-bit or 192-bit key
- Triple DES (TDEA) [7] with three different keys
- One of the HMAC-SHA ciphers [9] [10] with a 256-bit key

The first option, i e AES-256, is the preferred one for DTP. The others may be used for performance reasons, or when there is limited algorithm support.

2. SELECT INPUT AND OUTPUT ALPHABETS

DTP needs an input alphabet, i e a set of all different characters the plaintext may have. If having numeric characters only, like when encrypting CCN data, this set would be the characters `{0 1 2 3 4 5 6 7 8 9}`.

All characters to be encrypted must be part of the chosen input alphabet. To have consistent output, the alphabet size and the order of individual alphabet characters must be the same. DTP encrypts characters based on their alphabet position; moving characters within the alphabet will give different output.

The output alphabet is the set of all different characters the ciphertext may have. For DTP this is the same as the input alphabet, but potentially an output set could have only some or no characters in common with the input set. The only requirement is that the size of the output alphabet must be at least the size of the input alphabet when encrypting.

Using the Base26 output set `{a b c d e f g h i j k l m n o p q r s t u v w x y z}` for the numeric input set `{0 1 2 3 4 5 6 7 8 9}`, would be an example of no common characters for the input and output sets. Such completely different sets prevents from re-encrypting the same data twice. A larger output set may also enable storage of metadata within the output as discussed in *Using DTP with Format-Controlling Encryption*. This metadata may be used for such as initial value, integrity checksum and/or key generation identifier. The larger the output set, the more data may be stored.

When using different input and output sets it must be possible to store and process different characters than the original ones, which may not be the case. If so the only option is to use the same set for both input and output, which means checks for encrypted data and metadata storage isn't supported.

3. PUT INDEX VALUES INTO A WORK BUFFER

When performing DTP encryption, the plaintext characters to be encrypted will be put into a separate work buffer, converted according to their zero-based index in the input alphabet. Having an M-byte input set A with values $\{A_1\ A_2\ \ldots\ A_M\}$, an n-byte plaintext to be encrypted may be represented as:

$$\{P_1\ P_2\ P_3\ \ldots\ P_n\} \qquad P_k \in A$$

All of these n bytes will be converted according to their zero-based index in A. After this conversion the plaintext may be represented as:

$$\{p_1\ p_2\ p_3\ \ldots\ p_n\} \qquad 0 <= p_k < M$$

For an input set `{0 1 2 3 4 5 6 7 8 9}` and the string `1122334455667788`, index values are in hex:

```
INDEX VALUE DATA: 0x01010202030304040505060607070808
```

A character not part of the input set will not be encrypted. If not part of the output set it will be left unmodified, not affecting the encrypted value. If part of the output set an error will be raised to prevent from interpreting the character as an encrypted one.

As an example, suppose the input set is the numeric character set `{0 1 2 3 4 5 6 7 8 9}` and the output set is `{a b c d e f g h i j k l m n o p q r s t u v w x y z}`. If the string `1234567890` is encrypted into `abcdefghij`, then `1-2-3-4-5!6-7-8-9-0` will be encrypted into `a-b-c-d-e!f-g-h-i-j`. The characters `-` and `!` are not part of the input/output sets, and hence left unmodified and not affecting the encrypted values. The input value `123456789a` will however raise an error, since `a` is not part of the input set, but part of the output set.

4. ADD POSITION-SENSITIVE OFFSETS TO INDEX VALUES

In step 6 of [1] it's specified that each index value in the work buffer is supposed to have a modular addition based on a key-dependent scheme. It's not specified how the offset values are created, but in an example the offsets *"are generated based on a portion of the key being used to encrypt the data"*. Hence they are the same for a certain key.

DTP pre-scrambling includes a similar position-sensitive modular addition, but for DTP the offset values are based on applying the internal cipher over the complete data value. This makes the scrambling not only key-dependent as in [1], but also data-dependent. This process is referred to as *cipher scrambling*.

Suppose there is an M-byte input set. After index conversion in *3. Put index values into a work buffer*, an n-byte plaintext may be represented as:

$$\{p_1\ p_2\ p_3\ \ldots\ p_n\} \qquad 0 <= p_k < M$$

This value is separated into one left and one right part. If the internal cipher block size is `B` bytes, the right part consists of the last `L` index values, where for a single-byte set `L <= B` and `L <= n/2`, and for a double-byte character set `2*L <= B` and `L <= n/2`. The left part consists of all remaining index values.

$$\{p_1\ p_2\ p_3\ \ldots\ p_{n-L}\}\ +\ \{p_{n-L+1}\ p_{n-L+2}\ \ldots\ p_n\}$$

The left part is inserted into a cipher buffer, and padded up to the length of the block size (if not being a multiple of the block size already) with bytes of value `0xf1`. For a double-byte character set the index values are inserted as 2-byte integers in network order. The cipher buffer constructed is then encrypted in CBC mode [6] using an IV with all bytes set to zero. The last output block from this operation will be:

$$\{e_1\ e_2\ e_3\ \ldots\ e_B\} \qquad 0 <= e_k < 256$$

This block is then modularly added to the right part by reading 1 or 2 bytes from the block until all index values for the right part have been updated. This may be described as (here $e_{2k-1}e_{2k}$ should be interpreted as a 2-byte integer in network order):

$$p'_{n-L+k} = (p_{n-L+k} + e_k) \bmod M \qquad 0 < M <= 256;\ 0 < k <= L$$

$$p'_{n-L+k} = (p_{n-L+k} + e_{2k-1}e_{2k}) \bmod M \qquad M > 256;\ 0 < k <= L$$

After this operation, the updated n-byte plaintext may be represented as:

$$\{p'_1\ p'_2\ p'_3\ \ldots\ p'_n\} \qquad 0 <= p'_k < M$$

Here the last `L` index values have been updated, and the others are unchanged. This value is once again separated into one left and one right part. This time the left part consists of the first `L` index values, where `L` is chosen as above, and the right part consists of all remaining index values:

$$\{p'_1\ p'_2\ \ldots\ p'_L\}\ +\ \{p'_{L+1}\ p'_{L+2}\ p'_{L+3}\ \ldots\ p'_n\}$$

The right part is inserted into the cipher buffer, if necessary padded with bytes of value `0xf2`, and enciphered with a zeroized IV as before. The last output block from this operation will be:

$$\{e'_1\ e'_2\ e'_3\ \ldots\ e'_B\} \qquad 0 <= e'_k < 256$$

This block is then modularly added to the left part by reading 1 or 2 bytes from the block until all index values for the left part have been updated. This may be described as (here $e'_{2k-1}e'_{2k}$ should be interpreted as a 2-byte integer in network order):

$$h_k = (p'_k + e'_k) \bmod M \qquad 0 < M <= 256;\ 0 < k <= L$$

$$h_k = (p'_k + e'_{2k-1}e'_{2k}) \bmod M \qquad M > 256;\ 0 < k <= L$$

The resulting value will be this, for which the first `L` index values will depend on the whole data value:

$$\{h_1\ h_2\ h_3\ \dots\ h_n\} \qquad 0 <= h_k < M$$

For a numeric set `{0 1 2 3 4 5 6 7 8 9}` and AES-256 encryption, cipher scrambling would be like:

```
KEY: 0x0123456789abcdef111213141516171821222324252627283132333435363738
ORIGINAL DATA: 1122334455667788

INDEX VALUE DATA: 0x01010202030304040505060607070808 (after step 3)

CIPHER INPUT L:   0x0101020203030404f1f1f1f1f1f1f1f1 (left half + pad)
CIPHER OUTPUT L:  0x00c01f49d0c2c050188d8fdfadcdf846 (encrypted left half + pad)
RIGHT UPDATED:    0x01010202030304040507070905010008 (right half updated)

CIPHER INPUT R:   0x0507070905010008f2f2f2f2f2f2f2f2 (right half + pad)
CIPHER OUTPUT R:  0x4664f67128ad3b450df3f31c7d2df7ba (encrypted right half + pad)
CIPHER SCRAMBLED: 0x01010805030603030507070905010008 (left half updated)
```

This step is optional for DTP if using the random option in *7. Select initial value*.

5. SHUFFLE THE WORK BUFFER

In step 7 of [1] it's specified that the index values in the work buffer are supposed to be shuffled according to a data-dependent scheme. About this it's said that the *"shuffling method varies according to a permutation-invariant property of the index values, such as a sum or exclusive-or, of all values"*.

DTP shuffling is based on the Knuth shuffle method [11], executed from the back, i e the first index value shuffled is the last value in the work buffer, with sum and exclusive-or (XOR) of all values as input.

Suppose there is an M-byte input set. After cipher scrambling in *4. Add position-sensitive offsets to index values*, an n-byte plaintext may be represented as:

$$\{h_1\ h_2\ h_3\ \dots\ h_n\} \qquad 0 <= h_k < M$$

The shuffling used in DTP may then be described as

```
for i = n to 2
  j = rand(i)
  swap h_i and h_j
```

The shuffle includes a call to `rand(i)`, which is supposed to produce a random value `1 <= j <= i`. For DTP the `rand(i)` function is based on the output from the internal cipher as follows.

First an initial value (IV) with the same length as the block size is produced as:

- first 4 bytes are the sum of all index values as a 4-byte integer in network order
- following byte is the XOR value taken over all bytes in the work buffer
- remaining bytes all have the value `0xfe`

With a sum of `72`, and an XOR value of `0x08` the IVs for different algorithms will be in hex encoding:

```
AES:       0x0000004808fefefefefefefefefefefe
TDEA:      0x0000004808fefefe
HMAC-SHA1: 0x0000004808fefefefefefefefefefefefefefefe
```

The IV is then processed by the cipher algorithm and key as chosen in *1. Select algorithm and key*, producing some cipher output as:

$\{e_1\ e_2\ e_3 \ldots e_B\}$ $\qquad 0 <= e_k < 256$

The first return value for `rand(i)` is then taken by reading 1 or 2 bytes from the cipher output, followed by a modular division of the value. 1 byte is read when asking for a value `j <= 32`. This may be described as (here e_1e_2 should be interpreted as a 2-byte integer in network order):

$j = e_1 \bmod i + 1$ $\qquad 1 < i <= 32$

$j = e_1e_2 \bmod i + 1$ $\qquad i > 32$

Modular division will not provide a uniformly distributed result, since some values will be more likely than others, unless `i` has the format 2^x. This is however neglected to some extent, since true randomness is not the real target here. Using a more stringent random number generation would hurt performance disproportionately, especially when pre-computation of the permutation isn't possible.

The second return value for `rand(i)` will be the following 1 or 2 bytes:

$j = e_2 \bmod i + 1$ $\qquad 1 < i <= 32$

$j = e_3e_4 \bmod i + 1$ $\qquad i > 32$

This process, reading 1 or 2 bytes and then move the read position 1 or 2 bytes forward, will continue until the complete cipher block has been used. Then this block is reencrypted, producing a new cipher block to be used. This process continues until the shuffling has been finished.

For a numeric set `{0 1 2 3 4 5 6 7 8 9}` and AES-256 encryption, shuffling would be like:

```
KEY: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38
ORIGINAL DATA: 1122334455667788

INDEX VALUE DATA: 0x01010202030304040505060607070808 (after step 3)
CIPHER SCRAMBLED: 0x01010805030603030507070905010008 (after step 4)

INITIAL VALUE: 0x0000004808fefefefefefefefefefefe (shuffle IV)
CIPHER OUTPUT: 0x661a425236f7716c7b6a53646fba3297 (encrypted IV)

SHUFFLE VALUE 16: 0x01010805030608030507070905010003
SHUFFLE VALUE 15: 0x01010805030608030507070005010903
SHUFFLE VALUE 14: 0x01010805030608030507010005070903
```

```
SHUFFLE VALUE 13: 0x01010805050608030507010003070903
SHUFFLE VALUE 12: 0x01010805050600030507010803070903
SHUFFLE VALUE 11: 0x01010805050100030507060803070903
SHUFFLE VALUE 10: 0x01010807050100030505060803070903
SHUFFLE VALUE  9: 0x05010807050100030105060803070903
SHUFFLE VALUE  8: 0x05010803050100070105060803070903
SHUFFLE VALUE  7: 0x05000803050101070105060803070903
SHUFFLE VALUE  6: 0x05000803050101070105060803070903
SHUFFLE VALUE  5: 0x05000803050101070105060803070903
SHUFFLE VALUE  4: 0x05000803050101070105060803070903
SHUFFLE VALUE  3: 0x08000503050101070105060803070903
SHUFFLE VALUE  2: 0x00080503050101070105060803070903

SHUFFLED DATA: 0x00080503050101070105060803070903 (shuffled work buffer)
```

This step is optional for DTP encryption, and normally not used. An exception is when having varying-length characters, as described in the section *DTP Encryption with Varying-length Character Sets*; in this case shuffling is strongly recommended. When using multiple input alphabets, as described in the section *DTP Encryption with Multiple Character Sets*, shuffling isn't used.

6. RIPPLE DATA BY MODULARLY ADDING PAIRWISE

DTP performs rippling, i e pairwise modular addition of the index values, first left-to-right, then right-to-left, corresponding to step 8 of [1]. The starter value is however always set to zero for DTP, not *"obtained by hashing the encryption key"* as for [1].

Suppose there is an M-byte input set. After cipher scrambling in *4. Add position-sensitive offsets to index values*, and optionally with shuffling in *5. Shuffle the work buffer* applied, an n-byte plaintext may be represented as:

$$\{h_1\ h_2\ h_3\ \dots\ h_n\} \qquad 0 <= h_k < M$$

The rippling left-to-right may then be described as

$$r'_1 = h_1 \qquad 0 <= r'_k < M$$
$$r'_2 = (r'_1 + h_2) \bmod M$$
$$r'_3 = (r'_2 + h_3) \bmod M$$
$$\dots$$
$$r'_n = (r'_{n-1} + h_n) \bmod M$$

The following right-to-left rippling may be described as

$$r_n = r'_n \qquad 0 <= r_k < M$$
$$r_{n-1} = (r_n + r'_{n-1}) \bmod M$$
$$r_{n-2} = (r_{n-1} + r'_{n-2}) \bmod M$$
$$\dots$$
$$r_1 = (r_2 + r'_1) \bmod M$$

The rippled result will be these values, to be used as input for the encryption:

$$\{r_1\ r_2\ r_3\ \dots\ r_n\} \qquad 0 <= r_k < M$$

For a numeric set `{0 1 2 3 4 5 6 7 8 9}` and AES-256 encryption, rippling would be:

```
KEY: 0x0123456789abcdef1112131415161718212223242526272831323334353637
38
ORIGINAL DATA: 1122334455667788

INDEX VALUE DATA: 0x01010202030304040505060607070808 (after step 3)
CIPHER SCRAMBLED: 0x01010805030603030507070905010008 (after step 4)

RIPPLED DATA L-R: 0x01020005080407000502090803040402 (first rippled value)
RIPPLED DATA R-L: 0x04030101060804070702000103000602 (final rippled value)
```

This step is optional for DTP if using the random option in *7. Select initial value*.

7. SELECT INITIAL VALUE

DTP encryption needs an initial value (IV) for the encryption performed in *8. Encrypt using a modular sum*. In [1] this IV is "*constructed based on a portion of the encryption key"*. This implies the key value is available in cleartext, which may not be the case if for instance using a key inside an HSM.

For DTP the IV is selected by one of these options:

- the IV is a completely random value, unique for each encryption; this is the preferred option
- the IV is a value dependent on data length, as specified below

Since a DTP requirement is to keep data length, the first option implies the IV is a unique value stored outside of the encrypted field, like a database table primary key. With this option it must be ensured that the IV can't be modified without also modifying the encrypted value; otherwise the value can't be correctly decrypted.
This may be hard under some circumstances, such as when performing column-level encryption in a database. Some suitable external IV may simply also not be available. There could also be requirements for having consistent encrypted output, e g to support join operations over encrypted values in a database. In these cases a fix IV is necessary, which the second option means.

When using a random IV, the pre-scrambling performed in steps 4 and 6 isn't necessary and may be skipped. A random IV will provide different output for the same input, and no scrambling preparation is necessary.

When using the second option, the DTP IV is created as:

- first 4 bytes are the data length as a 4-byte integer in network order
- remaining bytes all have the value `0xff`

With a data length of 16, the DTP IV will for different algorithms be set to:

```
AES:       0x00000010ffffffffffffffffffffffff
TDEA:      0x00000010ffffffff
HMAC-SHA1: 0x00000010ffffffffffffffffffffffffffffffff
```

Using the FCE option, as described in *Using DTP with Format-Controlling Encryption*, it may be possible to have a random IV even without external storage. The IV will however not be complete with FCE; it will be restricted up to the FCE compression rate.

8. ENCRYPT USING A MODULAR SUM OVER INDEX VALUE AND INTERNAL CIPHER

DTP encryption is performed as specified in [1], using the IV as set in *7. Select initial value.*

For each index value, DTP performs a modular add operation with the last 1 or 2 encrypted bytes of the cipher block. Next cipher input is then produced by shifting the cipher output 1 or 2 bytes to the left, and inserting the last index value that was encrypted. This is a sort of plaintext feedback.

Suppose there is an M-byte input set and an N-byte output set, with N >= M. After rippling, an n-byte plaintext may be represented as:

$$\{r_1\ r_2\ r_3\ \ldots\ r_n\} \qquad 0 \le r_k < M$$

If M > 256, each r_k value would here be double-byte values.

Assume the block cipher used is TDEA. Since this works on 8-byte blocks, the IV would be like:

$$\{i_1\ i_2\ i_3\ i_4\ i_5\ i_6\ i_7\ i_8\} \qquad 0 \le i_k < 255$$

Applying the internal cipher to this IV will result in the output:

$$\{o_{11}\ o_{12}\ o_{13}\ o_{14}\ o_{15}\ o_{16}\ o_{17}\ o_{18}\} \qquad 0 \le o_{1k} < 255$$

The first DTP-encrypted index value c_1 will then be produced as (here $o_{17}o_{18}$ should be interpreted as a 2-byte integer in network order):

$$c_1 = (r_1 + o_{18}) \bmod N \qquad 0 < N \le 256$$

$$c_1 = (r_1 + o_{17}o_{18}) \bmod N \qquad N > 256$$

Next cipher input will be (here $r_{11}r_{12}$ is the rippled index value as a 2-byte integer in network order):

$$\{o_{12}\ o_{13}\ o_{14}\ o_{15}\ o_{16}\ o_{17}\ o_{18}\ r_1\} \qquad 0 < M \le 256$$

$$\{o_{13}\ o_{14}\ o_{15}\ o_{16}\ o_{17}\ o_{18}\ r_{11}\ r_{12}\} \qquad 256 < M$$

Encrypting this input block will result in the output block:

$$\{o_{21}\ o_{22}\ o_{23}\ o_{24}\ o_{25}\ o_{26}\ o_{27}\ o_{28}\} \qquad 0 \le o_{2k} < 255$$

The second encrypted index value c_2 will then be produced as:

$$c_2 = (r_2 + o_{28}) \bmod N \qquad 0 < N \le 256$$

$$c_2 = (r_2 + o_{27}o_{28}) \bmod N \qquad N > 256$$

and the next input block for TDEA will be:

$$\{o_{22}\ o_{23}\ o_{24}\ o_{25}\ o_{26}\ o_{27}\ o_{28}\ r_2\} \qquad 0 < M \le 256$$

$$\{o_{23}\ o_{24}\ o_{25}\ o_{26}\ o_{27}\ o_{28}\ r_{21}\ r_{22}\} \qquad 256 < M$$

This encryption process continues until r_n is encrypted. The result will be the values:

$$\{c_1\ c_2\ c_3\ \ldots\ c_n\} \qquad\qquad 0 <= c_k < N$$

For a numeric set {0 1 2 3 4 5 6 7 8 9} and AES-256 encryption, DTP encryption would be like:

```
KEY: 0x0123456789abcdef111213141516171821222324252627283132333435363738
ORIGINAL DATA: 1122334455667788

INDEX VALUE DATA: 0x01010202030304040505060607070808 (after step 3)
CIPHER SCRAMBLED: 0x01010805030603030507070905010008 (after step 4)
RIPPLED DATA R-L: 0x04030101060804070702000103000602 (after step 6)

ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

  1     00000010ffffffffffffffffffffffff   04  3a  02
        8f6307734f2bf814ca0c34b609ef3c3a
  2     6307734f2bf814ca0c34b609ef3c3a04   03  63  02
        7f513c44b78e077abe8642fa3e0ee063
  3     513c44b78e077abe8642fa3e0ee06303   01  57  08
        10c60493ff20c9af57115211c50be857
  4     c60493ff20c9af57115211c50be85701   01  b8  05
        537f88b01588fc1de873e20133c847b8
  5     7f88b01588fc1de873e20133c847b801   06  d4  08
        44a0ef375bda4a0c993fb4fcd6ce4fd4
  6     a0ef375bda4a0c993fb4fcd6ce4fd406   08  51  09
        ebf609d5d9d536941444cfd6cd5bcc51
  7     f609d5d9d536941444cfd6cd5bcc5108   04  b0  00
        fdc2747d9da91dd711d1794bb23248b0
  8     c2747d9da91dd711d1794bb23248b004   07  65  08
        fdee05e508f6eecca476ba61fb470f65
  9     ee05e508f6eecca476ba61fb470f6507   07  e1  02
        4b749f270510d9e96f79f63d049935e1
 10     749f270510d9e96f79f63d049935e107   02  3e  04
        c4d949adb89c545b701d2f0b0f3c653e
 11     d949adb89c545b701d2f0b0f3c653e02   00  bb  07
        12b5cdac616722c3a4918659790c5ebb
 12     b5cdac616722c3a4918659790c5ebb00   01  81  00
        6457d064b94f918721b187fa8d444181
 13     57d064b94f918721b187fa8d44418101   03  1a  09
        62d16a84f6257580465d0c187affeb1a
 14     d16a84f6257580465d0c187affeb1a03   00  71  03
        0c785062aca4049c92cd05b6b77a6671
 15     785062aca4049c92cd05b6b77a667100   06  a0  06
        41e30a70df15a0622d5e1c1dcb2205a0
 16     e30a70df15a0622d5e1c1dcb2205a006   02  4a  06
        19e414b080d91da82fbeadf1a0fa3e4a
```

For some examples when performing DTP encryption with different algorithms and output sets, see Appendix A.

For some examples of DTP encryption output when having similar input, see Appendix C.

9. REPLACE ENCRYPTED INDEX VALUES

Final step for DTP encryption is to convert the encrypted index values into their corresponding values in the output set. This is normally the same as the input set, but if using the FCE option, as described in *Using DTP with Format-Controlling Encryption,* it may also be different.

Suppose there is an N-byte output set B with values $\{B_1\ B_2\ \dots\ B_N\}$. After encryption, an n-byte value may be represented as:

$$\{c_1\ c_2\ c_3\ \dots\ c_n\} \qquad 0 <= c_k < N$$

These values will then be converted according to their zero-based index in B, using the B character at the index. The final encrypted result will be:

$$\{C_1\ C_2\ C_3\ \dots\ C_n\} \qquad C_k \in B$$

For a numeric set {0 1 2 3 4 5 6 7 8 9} and AES-256 encryption, output set conversion would be like:

```
KEY: 0x0123456789abcdef111213141516171821222324252627283132333435363738
ORIGINAL DATA: 1122334455667788

INDEX VALUE DATA: 0x01010202030304040505060607070808 (after step 3)
CIPHER SCRAMBLED: 0x01010805030603030507070905010008 (after step 4)
RIPPLED DATA R-L: 0x04030101060804070702000103000602 (after step 6)
ENCRYPTED DATA:   0x02020805080900080204070009030606 (after step 8)

OUTPUT DATA: 2285890824709366 (encrypted value in output set)
```

VARIATIONS AND UTILIZATION

DTP MULTI-BYTE ENCRYPTION

The main DTP encryption operation, as specified in *8. Encrypt using a modular sum* and [1], performs encryption byte-by-byte; there will be as many internal cipher operations as work buffer length. For instance, if encrypting a 16-byte credit card number, 16 AES encryptions are needed instead of 1 if using CBC mode (without padding). This will hurt performance; basic DTP is a rather slow construction.

If comparing with CFB mode, it's possible to run CFB in 8-bit or byte-oriented mode, in which case its performance would be similar to DTP. But when using CFB, most often multiple bytes are encrypted for each internal cipher operation. This is also possible for DTP, and is called *multi-byte DTP*.

For multi-byte DTP the work buffer is handled in blocks of data, where each block has q bytes. If the input length isn't a multiple of q, the last block will have less then q bytes, all the others will have q bytes. If using 3-byte DTP, an input length of 9 would have 3 blocks, and an input length of 16 would have 6:

$$\{r_1\ r_2\ r_3 \quad r_4\ r_5\ r_6 \quad r_7\ r_8\ r_9\}$$

$$\{r_1\ r_2\ r_3 \quad r_4\ r_5\ r_6 \quad r_7\ r_8\ r_9 \quad r_{10}\ r_{11}\ r_{12} \quad r_{13}\ r_{14}\ r_{15} \quad r_{16}\}$$

Each q-byte block is encrypted with q separate modular additions. In this the plaintext is read from the left and the cipher output is read from the right, i e backwards. As an example, for 3-byte DTP with an M-byte input set and an N-byte output set, the first 3 rippled bytes would be encrypted as:

$$c_1 = (r_1 + o_{18}) \bmod N \qquad 0 < M <= 256$$
$$c_2 = (r_2 + o_{17}) \bmod N$$
$$c_3 = (r_3 + o_{16}) \bmod N$$

To produce the next input block, the previous output block is shifted q bytes to the left and appended with the last q plaintext bytes used. Next input block would for the 3-byte example hence be:

$$\{o_{14}\ o_{15}\ o_{16}\ o_{17}\ o_{18}\ r_1\ r_2\ r_3\} \qquad 0 < M <= 256$$

For a numeric set {0 1 2 3 4 5 6 7 8 9} and AES-256 encryption, 3-byte DTP encryption would be like:

```
 KEY: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38
 ORIGINAL DATA: 1122334455667788
 RIPPLED DATA R-L: 0x04030101060804070702000103000602

ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

  1     00000010ffffffffffffffffffffffff   04  3a  02
        8f6307734f2bf814ca0c34b609ef3c3a   03  3c  03
                                           01  ef  00
  2     734f2bf814ca0c34b609ef3c3a040301   01  bb  08
        c0bc489674f55297e78c20bc5bbacdbb   06  cd  01
                                           08  ba  04
  3     9674f55297e78c20bc5bbacdbb010608   04  b8  08
        3978d5fee586cb46d88b52c72bba07b8   07  07  04
                                           07  ba  03
  4     fee586cb46d88b52c72bba07b8040707   02  23  07
        9ce673dc70de1f9616fb99b1ea763b23   00  3b  09
                                           01  76  09
  5     dc70de1f9616fb99b1ea763b23020001   03  4a  07
        910939b7c2a6349d6e0ebdbc5151744a   00  74  06
                                           06  51  07
  6     b7c2a6349d6e0ebdbc5151744a030006   02  19  07
        1c0e73aaa98e8301aed26dde800f8419

 OUTPUT DATA: 2308148437997677 (encrypted value in output set)
```

For some additional examples when performing 3-byte DTP encryption, see Appendix B.
For some examples of 3-byte DTP encrypted values with similar input, see Appendix D.

Using 8-byte DTP encryption for TDEA, similar to 64-bit CFB mode, would provide the input block:

$$\{r_1\ r_2\ r_3\ r_4\ r_5\ r_6\ r_7\ r_8\} \qquad 0 < M <= 256$$

Here the value range of r_k may be limited, thereby providing limited cipher input. For this reason the size of n mustn't exceed B/2, where B is the block length for the internal cipher.
When not using a random IV, differences for n bytes of rippled index values may be seen based on the modular addition. For this reason, n mustn't exceed L/2, where L is the data length.

DTP ENCRYPTION WITH MULTIPLE CHARACTER SETS

DTP encryption, as described in the *Specification* section, assumes all characters to be encrypted are using the same alphabet. This is most often true. In some cases, however, there may be character-specific rules for the input data, making it necessary to use multiple alphabets.

For instance, suppose the value `A-111-B-222-C-333-D` is the target for DTP encryption. One option would be to use the alphabet `{0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}`, which covers all characters. In this case any encrypted character may have any value in this alphabet; the output could for instance be `W-T52-1-OQS-X-EE4-7`. But what if `111`, `222` and `333` must be encrypted into numbers, and `A`, `B`, `C` and `D` into letters, i e the output should be like `G-526-A-722-O-916-K`? Then multiple alphabets would be required.

One option would be to put the data into two work buffers; one buffer would contain `ABCD` and the other `111222333`. Then these would be encrypted using the alphabets `{0 1 2 3 4 5 6 7 8 9}` and `{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}` respectively. Though possible, this isn't an optimal situation. Work buffers may become very short with this type of separation. If having a value like `A-123-456-789`, there would even only be a single member in one of the buffers. This wouldn't provide an adequate encryption level.

A better option is to base the encryption over the whole value, though still keeping the different character rules. To accomplish this DTP uses a single work buffer also with multiple alphabets, but the conversion and modular addition operations are position-dependent. If having the value `A-111-B-222-C-333-D` and the requirements above, the work buffer would be created as:

```
ORIGINAL DATA: A-111-B-222-C-333-D

INDEX VALUE DATA: 0x00010101010202020203030303 (multi-alphabet, mixed work buffer)
```

Here the characters `ABCD` have been index converted based on `{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}` and `111222333` based on `{0 1 2 3 4 5 6 7 8 9}`. The `-` character isn't part of any of the two alphabets, and hence not part of the work buffer.

The index values are then cipher scrambled, rippled and encrypted. In these operations the modular addition will based on the alphabet size at each position. This will be like:

```
KEY: 0x0123456789abcdef111213141516171821222324252627283132333435363738

CIPHER INPUT L: 0x00010101010202f1f1f1f1f1f1f1f1f1
CIPHER OUTPUT L: 0x25bbca08582d5bbdd396819eb69ac321
RIGHT UPDATED: 0x00010101010202090705010116

CIPHER INPUT R: 0x02090705010116f2f2f2f2f2f2f2f2f2
CIPHER OUTPUT R: 0x822f84e31ecc136b54f741cf751fc668
CIPHER SCRAMBLED: 0x00080308050602090705010116

RIPPLED DATA L-R: 0x000801090e0002010803040501
RIPPLED DATA R-L: 0x06060807120404020b03000601

ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

  1     0000000dffffffffffffffffffffffff   06  3e  10
        a61e17a43f7ec8622fca4b5b323fcf3e
```

2	1e17a43f7ec8622fca4b5b323fcf3e06 3ad2cfb2826fafad4e522831f011e3f0	06	f0	06
3	d2cfb2826fafad4e522831f011e3f006 771c1d9b4e8697764beba39a83a2bc97	08	97	09
4	1c1d9b4e8697764beba39a83a2bc9708 0935343b10eb8f58801a474bac7096cd	07	cd	02
5	35343b10eb8f58801a474bac7096cd07 093806677ffc0b0d91af0bc6caa55e25	12	25	03
6	3806677ffc0b0d91af0bc6caa55e2512 f81e5add18fe8236406feab5cc458b69	04	69	09
7	1e5add18fe8236406feab5cc458b6904 08b2722d4398ad045ca4a2081558d0dd	04	dd	05
8	b2722d4398ad045ca4a2081558d0dd04 413ec5d105a4fdb355a6dca7b1a4bb43	02	43	09
9	3ec5d105a4fdb355a6dca7b1a4bb4302 e11ed1a091a1145d8a8af2188d400966	0b	66	09
10	1ed1a091a1145d8a8af2188d4009660b 4b0428e71c7761efc1e61486ef1e9c9e	03	9e	01
11	0428e71c7761efc1e61486ef1e9c9e03 921b50ff8170200c707b54cd3003ed7b	00	7b	03
12	1b50ff8170200c707b54cd3003ed7b00 ff72d45c040bf3615b0338ad58638ba5	06	a5	01
13	72d45c040bf3615b0338ad58638ba506 50f85053aaa6ebc74f11d3a50af22b58	01	58	0b

This encrypted index values are then converted using the alphabet for each position, resulting in

```
OUTPUT DATA: Q-692-D-959-J-131-L
```

Here only two alphabets have been used, but the same rules apply if using additional alphabets. Each character is handled based on the alphabet for that position. The number of alphabets may be as many as there are number of characters.

DTP ENCRYPTION WITH VARYING-LENGTH CHARACTER SETS

The encryption specified in [1] concerns single-byte characters. DTP encryption, as described in the *Specification* section, may also be used with double-byte characters, i e when all characters have exactly two bytes.

However, there may also be situations where the characters have varying length. As an example, if having data within the Swedish alphabet {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Å Ä Ö}, this may be stored as single-byte characters only. But it could also be stored as UTF-8 characters in which case ÅÄÖ would be double-byte characters and ABCDEFGHIJKLMNOPQRSTUVWXYZ single-byte.

One target for DTP is to keep the output length. To ensure this, a 1-byte character must be encrypted into 1 byte, a 2-byte character into 2 bytes, etc. For this reason it's necessary to separate the input/output set into different parts, where each part consists of the values with a specific length.

For the Swedish alphabet above and UTF-8 storage, this alphabet would be separated into the two subsets {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z} and {Å Ä Ö}. The alphabet to use for a certain character will then be selected based on the character's length; a 1-byte character will be index converted from the first alphabet, and a 2-byte character from the second.

Having two different alphabets means similar encryption may then be applied as for *DTP Encryption with Multiple Character Sets.* There is however one difference. To keep length, each 1-byte character will be mapped to another 1-byte character, each 2-byte to another 2-byte, etc. Hence it will be possible to identify the lengths for each character, which may expose the protected data.

This is a general problem with length-preserving encryption. Even if applying the strongest encryption possible, it's still easy to separate `John` from `Joe`; just look at the length. Length-preserving encryption must for this reason be applied with careful consideration, especially when protecting PII data; it mustn't be possible to perform identification based on data length only.

Using varying-length character sets like UTF-8 makes these considerations further complicated; there are additional plaintext relations in the output. In this case not only the data length is visible, but also the varying length of each character may be seen. This will further facilitate an identification task.

For this reason it's strongly recommended to apply shuffling when encrypting varying-length characters. This will hide the order of characters to some extent; how much will depend on the maximum sum and XOR distribution.

As an example, if having the Swedish region names `ÅNGERMANLAND` and `ÖSTERGÖTLAND` in UTF-8, the recommendation is to encrypt these with shuffling included. This will be like:

```
KEY: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38
ORIGINAL DATA: ÅNGERMANLAND

INDEX VALUE DATA: 0x000d0604110c000d0b000d03

CIPHER INPUT L: 0x000d0604110cf1f1f1f1f1f1f1f1f1f1
CIPHER OUTPUT L: 0x4a5c01d73055819c6eeab9f6a04b4e03
RIGHT UPDATED: 0x000d0604110c16010c07090a

CIPHER INPUT R: 0x16010c07090af2f2f2f2f2f2f2f2f2f2
CIPHER OUTPUT R: 0xba5f8f2c61f51e77f9c89aa15a46c816
CIPHER SCRAMBLED: 0x000413160a1716010c07090a

INITIAL VALUE: 0x0000008b03fefefefefefefefefefefe
CIPHER OUTPUT: 0x61ecf3ba6e092409affd9e8b9c310061
SHUFFLED DATA: 0x01090a070a00130c16161704

RIPPLED DATA L-R: 0x010a14010b02150703191600
RIPPLED DATA R-L: 0x1312080e0d02000518151600

ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

  1     0000000cffffffffffffffffffffffff   13  8e  05
        fb0e5d802f53ccbc58d99902fc7cbc8e
  2     0e5d802f53ccbc58d99902fc7cbc8e13   12  4b  0f
        ff9f1825395514396d1dcd927085594b
  3     9f1825395514396d1dcd927085594b12   08  bd  0f
        1d88f26df111395b365db2d0f1b229bd
  4     88f26df111395b365db2d0f1b229bd08   0e  46  06
        9270c31d79e9d39bcdb54a190a0e9746
  5     70c31d79e9d39bcdb54a190a0e97460e   0d  a5  16
        210e6d741f6236e4b445238044b84ca5
  6     0e6d741f6236e4b445238044b84ca50d   02  89  01
        7bca8c95be7df2e2be925b395df7c189
```

```
 7     ca8c95be7df2e2be925b395df7c18902   00  9d  01
       e35f5284c16e8373ebae829fd958f19d
 8     5f5284c16e8373ebae829fd958f19d00   05  98  01
       0d91b7d246b4caa79a9f023ab5595298
 9     91b7d246b4caa79a9f023ab559529805   18  ea  18
       98daec89c2b259faa82be35820c48dea
10     daec89c2b259faa82be35820c48dea18   15  c2  07
       7dc7311f54538a1d0a555cd7420eacc2
11     c7311f54538a1d0a555cd7420eacc215   16  59  07
       0db35c5cafe80cad3677d1a9c277c659
12     b35c5cafe80cad3677d1a9c277c65916   00  08  08
       bc0b83e2edb006e06cae565de0dc2f08

ENCRYPTED DATA: FPPGWÄBBYHHI


KEY: 0x0123456789abcdef1112131415161718212223242526272831323334353637338
ORIGINAL DATA: ÖSTERGÖTLAND

INDEX VALUE DATA: 0x02121304110602130b000d03

CIPHER INPUT L: 0x021213041106f1f1f1f1f1f1f1f1f1f1
CIPHER OUTPUT L: 0xe4fdf1a373f112da559c69412a43ddc4
RIGHT UPDATED: 0x021213041106020c1207180a

CIPHER INPUT R: 0x020c1207180af2f2f2f2f2f2f2f2f2f2
CIPHER OUTPUT R: 0xc54f9c035b40e09ece62c279161d03ef
CIPHER SCRAMBLED: 0x011313070412020c1207180a

INITIAL VALUE: 0x0000008d19fefefefefefefefefefefe
CIPHER OUTPUT: 0x8577f4714969f14548b8f622266bb039
SHUFFLED DATA: 0x13071202180c010a12040713

RIPPLED DATA L-R: 0x13001202000c010b03070e07
RIPPLED DATA R-L: 0x0c1313010e0e021005021507

ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

 1     0000000cffffffffffffffffffffffff   0c  8e  18
       fb0e5d802f53ccbc58d99902fc7cbc8e
 2     0e5d802f53ccbc58d99902fc7cbc8e0c   13  2a  09
       ddcd025c864cd5accf6225053be41e2a
 3     cd025c864cd5accf6225053be41e2a13   13  40  05
       f9250c00f46b327cc813960c1eb77c40
 4     250c00f46b327cc813960c1eb77c4013   01  5d  01
       d49e3723c059f6168f60d5e3360b2f5d
 5     9e3723c059f6168f60d5e3360b2f5d01   0e  3b  15
       17ff033dcb16e3f48de79c5a224b433b
 6     ff033dcb16e3f48de79c5a224b433b0e   0e  12  06
       47b159111501e2648f58ac760b60ef12
 7     b159111501e2648f58ac760b60ef120e   02  c0  02
       09ec21121d7749cb16d320c5898835c0
 8     ec21121d7749cb16d320c5898835c002   10  9f  13
       c43d3a0fc63d96cfdde555bf8bab359f
 9     3d3a0fc63d96cfdde555bf8bab359f10   05  e5  00
```

```
        8bc94d998fe99e425430d89badd409e5
10      c94d998fe99e425430d89badd409e505   02  04  06
        0f57941adc4527db8ca7b804c85dc104
11      57941adc4527db8ca7b804c85dc10402   15  c3  08
        fe8224aa249132ea9b781451a26f37c3
12      8224aa249132ea9b781451a26f37c315   07  06  0d
        423ee7f67ad364a4b04253ed99030e06

ENCRYPTED DATA: YJFÄVGÖTAGIN
```

Though shuffling certainly may hide the character order, it's still easy to separate `ÅNGERMANLAND` from `ÖSTERGÖTLAND` by looking at the output. `FPPGWÄBBYHHI` has a single 2-byte character, `YJFÄVGÖTAGIN` has two; hence `FPPGWÄBBYHHI` must be `ÅNGERMANLAND`. But without shuffling, it would also be possible to separate words with equal number of 2-byte character by looking at the character positions.

The general recommendation is to not use DTP with varying-length character sets, unless the distribution of characters with different lengths doesn't expose the data.

USING DTP WITH FORMAT-CONTROLLING ENCRYPTION

Having different output and input sets prevents from reencrypting the same data twice. If the output set size also is larger than the input size, i e there are more characters, it may also enable storage of metadata within the output with unchanged total data length. This is the base for Format-Controlling Encryption (FCE). The metadata would be data related to the encryption process, for instance:

- IV
- Integrity value
- Key identifier

A problem with length and format-preserving encryption is that it doesn't allow storage of an integrity value. Having the same input/output sets means it's not possible to detect a modification in the encrypted value. Keeping length means an IV or key identifier can't be added either. With a larger output set, this will however be possible to some extent; how much will depend on the size difference.

Metadata storage is enabled by first representing the plaintext in the larger output set, thereby effectively compressing the value. DTP encryption will then be performed over the compressed value, now having the input set the same as the output set. Remaining space may then be used for metadata storage.

An n-byte value to be encrypted based on an M-size input set may be interpreted as an integer value in base M, having the value:

$$p_1 * M^{n-1} + p_2 * M^{n-2} + p_3 * M^{n-3} + \ldots + p_{n-1} * M + p_n$$

The maximum if this base M value would be $M^n - 1$. A representation of the value in base N, the output set size, would be like:

$$q_1 * N^{k-1} + q_2 * N^{k-2} + q_2 * N^{k-3} + \ldots + q_{k-1} * N + q_k \qquad k <= n$$

The maximum of this would be $N^k - 1$. To get k, the number of bytes necessary to represent the base M value in base N, the requirement is that $N^k - 1 >= M^n - 1$. For such a k any base M value of length n may be represented.

$$N^k >= M^n \quad ===> \quad k >= n * \log(M)/\log(N)$$

This means there will be

```
n - n * log(M)/log(N) = n * (1 - log(M)/log(N))
```

bytes left for metadata storage. For instance, for a 16-digit credit card number stored in Base64 with M = 10 and N = 64, there would be 16 * (1 – log(10)/log(64)) = 7 bytes left for the metadata. Since each Base64 character may hold 6 bits, in total 7 * 6 = 42 bits would be available for the metadata. In this case it would be possible to have an IV corresponding to > 5 random bytes with data length kept.

A random IV means the DTP output will be different for the same input. One may also consider a combined integrity and IV format. First a reduced 42-bit HMAC integrity value is calculated over the plaintext, and put into the free 7 bytes. These HMAC bits are then used as IV for the DTP encryption operation. This handling will give the same DTP output for the same input, but the DTP encryption process will be stronger; encryption of each digit will depend on the whole value.

These are examples, the metadata format is not part of the DTP specification.

REFERENCES

[1] Using Datatype-Preserving Encryption to Enhance Data Warehouse Security, Harry Smith and Michael Brightwell, 20th NISSC Proceedings, Oct 1997
http://csrc.nist.gov/nissc/1997/proceedings/

[2] Payment Card Industry (PCI) Data Security Standard v1.2, Oct 2008
https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml

[3] Payment Card Industry (PCI) Data Security Standard Glossary, Abbreviations and Acronyms
https://www.pcisecuritystandards.org/security_standards/glossary.shtml

[4] NIST SP 800-57, Recommendation for Key Management, Mar 2007
http://csrc.nist.gov/publications/PubsSPs.html

[5] NIST SP 800-122, Guide to Protecting the Confidentiality of Personally Identifiable Information, Jan 2009
http://csrc.nist.gov/publications/PubsSPs.html

[6] NIST SP 800-38a, Recommendation for Block Cipher Modes of Operation, Dec 2001
http://csrc.nist.gov/publications/PubsSPs.html

[7] NIST SP 800-67, Recommendation for the Triple Data Encryption Algorithm (TDEA), May 2008
http://csrc.nist.gov/publications/PubsSPs.html

[8] FIPS 197, Advanced Encryption Standard, Nov 2001
http://csrc.nist.gov/publications/PubsFIPS.html

[9] FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), Jul 2008
http://csrc.nist.gov/publications/PubsFIPS.html

[10] NIST SP 800-107, Recommendation for Applications Using Approved Hash Algorithms, Feb 2009
http://csrc.nist.gov/publications/PubsSPs.html

[11] The Art of Computer Programming volume 2: Seminumerical algorithms, Donald E. Knuth, 1969

APPENDIX A: DTP 1-BYTE ENCRYPTION/DECRYPTION

Following are examples when encrypting/decrypting with DTP in 1-byte mode. In all examples the plaintext is the numeric string value `1122334455667788`. Binary values are presented as hex values.

Input set is `{0 1 2 3 4 5 6 7 8 9}`.
Output set is either the same or the larger set `{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}`.

TDEA 1-BYTE DTP ENCRYPTION/DECRYPTION

```
Key: 0x0123456789abcdef11121314151617182122232425262728
Cipher-scrambled data: 0x04070405080205090401090701030500
Rippled data: 0x00060500000202070305060803070804
```

DTP TDEA 1-byte; same output set

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	r	o	c
1	00000010ffffffff	056ee40032852045	00	45	09
2	6ee4003285204500	5d114159b9a60549	06	49	09
3	114159b9a6054906	826d05c47cb37dcb	05	cb	08
4	6d05c47cb37dcb05	083efc219fa875a4	00	a4	04
5	3efc219fa875a400	d2459eee91d0a614	00	14	00
6	459eee91d0a61400	a70751fcd219c383	02	83	03
7	0751fcd219c38302	fb2b0f3c9638383b	02	3b	01
8	2b0f3c9638383b02	1b04a457c797291b	07	1b	04
9	04a457c797291b07	b319e721a2e2dafe	03	fe	07
10	19e721a2e2dafe03	0c16d1e5581d4401	05	01	06
11	16d1e5581d440105	dd2ccef835e4e229	06	29	07
12	2ccef835e4e22906	c82e75158781b670	08	70	00
13	2e75158781b67008	f36dd78bd8c177ae	03	ae	07
14	6dd78bd8c177ae03	f19d808998f40578	07	78	07
15	9d808998f4057807	5a8855d5ce646c76	08	76	06
16	8855d5ce646c7608	d61784ee6d065bf8	04	f8	02

```
Encrypted value: 9984031476707762
```

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	c	o	r
1	00000010ffffffff	056ee40032852045	09	45	00
2	6ee4003285204500	5d114159b9a60549	09	49	06
3	114159b9a6054906	826d05c47cb37dcb	08	cb	05
4	6d05c47cb37dcb05	083efc219fa875a4	04	a4	00
5	3efc219fa875a400	d2459eee91d0a614	00	14	00
6	459eee91d0a61400	a70751fcd219c383	03	83	02
7	0751fcd219c38302	fb2b0f3c9638383b	01	3b	02
8	2b0f3c9638383b02	1b04a457c797291b	04	1b	07
9	04a457c797291b07	b319e721a2e2dafe	07	fe	03
10	19e721a2e2dafe03	0c16d1e5581d4401	06	01	05
11	16d1e5581d440105	dd2ccef835e4e229	07	29	06
12	2ccef835e4e22906	c82e75158781b670	00	70	08

13	2e75158781b67008	f36dd78bd8c177ae	07	ae	03
14	6dd78bd8c177ae03	f19d808998f40578	07	78	07
15	9d808998f4057807	5a8855d5ce646c76	06	76	08
16	8855d5ce646c7608	d61784ee6d065bf8	02	f8	04

Decrypted value: 1122334455667788

DTP TDEA 1-byte; different output set

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	r	o	c
1	00000010ffffffff	056ee40032852045	00	45	11
2	6ee4003285204500	5d114159b9a60549	06	49	01
3	114159b9a6054906	826d05c47cb37dcb	05	cb	00
4	6d05c47cb37dcb05	083efc219fa875a4	00	a4	08
5	3efc219fa875a400	d2459eee91d0a614	00	14	14
6	459eee91d0a61400	a70751fcd219c383	02	83	03
7	0751fcd219c38302	fb2b0f3c9638383b	02	3b	09
8	2b0f3c9638383b02	1b04a457c797291b	07	1b	08
9	04a457c797291b07	b319e721a2e2dafe	03	fe	17
10	19e721a2e2dafe03	0c16d1e5581d4401	05	01	06
11	16d1e5581d440105	dd2ccef835e4e229	06	29	15
12	2ccef835e4e22906	c82e75158781b670	08	70	10
13	2e75158781b67008	f36dd78bd8c177ae	03	ae	15
14	6dd78bd8c177ae03	f19d808998f40578	07	78	17
15	9d808998f4057807	5a8855d5ce646c76	08	76	16
16	8855d5ce646c7608	d61784ee6d065bf8	04	f8	12

Encrypted value: RBAIUDJIXGVQVXWS

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	c	o	r
1	00000010ffffffff	056ee40032852045	11	45	00
2	6ee4003285204500	5d114159b9a60549	01	49	06
3	114159b9a6054906	826d05c47cb37dcb	00	cb	05
4	6d05c47cb37dcb05	083efc219fa875a4	08	a4	00
5	3efc219fa875a400	d2459eee91d0a614	14	14	00
6	459eee91d0a61400	a70751fcd219c383	03	83	02
7	0751fcd219c38302	fb2b0f3c9638383b	09	3b	02
8	2b0f3c9638383b02	1b04a457c797291b	08	1b	07
9	04a457c797291b07	b319e721a2e2dafe	17	fe	03
10	19e721a2e2dafe03	0c16d1e5581d4401	06	01	05
11	16d1e5581d440105	dd2ccef835e4e229	15	29	06
12	2ccef835e4e22906	c82e75158781b670	10	70	08
13	2e75158781b67008	f36dd78bd8c177ae	15	ae	03
14	6dd78bd8c177ae03	f19d808998f40578	17	78	07
15	9d808998f4057807	5a8855d5ce646c76	16	76	08
16	8855d5ce646c7608	d61784ee6d065bf8	12	f8	04

Decrypted value: 1122334455667788

AES-256 1-BYTE DTP ENCRYPTION/DECRYPTION

Key: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38
Cipher-scrambled data: 0x01010805030603030507070905010008
Rippled data: 0x04030101060804070702000103000602

DTP AES-256 1-byte; same output set

ROUND	AES-256 INPUT/OUTPUT BLOCK	r	o	c
1	00000010ffffffffffffffffffffffff	04	3a	02
	8f6307734f2bf814ca0c34b609ef3c3a			
2	6307734f2bf814ca0c34b609ef3c3a04	03	63	02
	7f513c44b78e077abe8642fa3e0ee063			
3	513c44b78e077abe8642fa3e0ee06303	01	57	08
	10c60493ff20c9af57115211c50be857			
4	c60493ff20c9af57115211c50be85701	01	b8	05
	537f88b01588fc1de873e20133c847b8			
5	7f88b01588fc1de873e20133c847b801	06	d4	08
	44a0ef375bda4a0c993fb4fcd6ce4fd4			
6	a0ef375bda4a0c993fb4fcd6ce4fd406	08	51	09
	ebf609d5d9d536941444cfd6cd5bcc51			
7	f609d5d9d536941444cfd6cd5bcc5108	04	b0	00
	fdc2747d9da91dd711d1794bb23248b0			
8	c2747d9da91dd711d1794bb23248b004	07	65	08
	fdee05e508f6eecca476ba61fb470f65			
9	ee05e508f6eecca476ba61fb470f6507	07	e1	02
	4b749f270510d9e96f79f63d049935e1			
10	749f270510d9e96f79f63d049935e107	02	3e	04
	c4d949adb89c545b701d2f0b0f3c653e			
11	d949adb89c545b701d2f0b0f3c653e02	00	bb	07
	12b5cdac616722c3a4918659790c5ebb			
12	b5cdac616722c3a4918659790c5ebb00	01	81	00
	6457d064b94f918721b187fa8d444181			
13	57d064b94f918721b187fa8d44418101	03	1a	09
	62d16a84f6257580465d0c187affeb1a			
14	d16a84f6257580465d0c187affeb1a03	00	71	03
	0c785062aca4049c92cd05b6b77a6671			
15	785062aca4049c92cd05b6b77a667100	06	a0	06
	41e30a70df15a0622d5e1c1dcb2205a0			
16	e30a70df15a0622d5e1c1dcb2205a006	02	4a	06
	19e414b080d91da82fbeadf1a0fa3e4a			

Encrypted value: 2285890824709366

ROUND	AES-256 INPUT/OUTPUT BLOCK	c	o	r
1	00000010ffffffffffffffffffffffff	02	3a	04
	8f6307734f2bf814ca0c34b609ef3c3a			
2	6307734f2bf814ca0c34b609ef3c3a04	02	63	03
	7f513c44b78e077abe8642fa3e0ee063			
3	513c44b78e077abe8642fa3e0ee06303	08	57	01
	10c60493ff20c9af57115211c50be857			
4	c60493ff20c9af57115211c50be85701	05	b8	01

	537f88b01588fc1de873e20133c847b8			
5	7f88b01588fc1de873e20133c847b801	08	d4	06
	44a0ef375bda4a0c993fb4fcd6ce4fd4			
6	a0ef375bda4a0c993fb4fcd6ce4fd406	09	51	08
	ebf609d5d9d536941444cfd6cd5bcc51			
7	f609d5d9d536941444cfd6cd5bcc5108	00	b0	04
	fdc2747d9da91dd711d1794bb23248b0			
8	c2747d9da91dd711d1794bb23248b004	08	65	07
	fdee05e508f6eecca476ba61fb470f65			
9	ee05e508f6eecca476ba61fb470f6507	02	e1	07
	4b749f270510d9e96f79f63d049935e1			
10	749f270510d9e96f79f63d049935e107	04	3e	02
	c4d949adb89c545b701d2f0b0f3c653e			
11	d949adb89c545b701d2f0b0f3c653e02	07	bb	00
	12b5cdac616722c3a4918659790c5ebb			
12	b5cdac616722c3a4918659790c5ebb00	00	81	01
	6457d064b94f918721b187fa8d444181			
13	57d064b94f918721b187fa8d44418101	09	1a	03
	62d16a84f6257580465d0c187affeb1a			
14	d16a84f6257580465d0c187affeb1a03	03	71	00
	0c785062aca4049c92cd05b6b77a6671			
15	785062aca4049c92cd05b6b77a667100	06	a0	06
	41e30a70df15a0622d5e1c1dcb2205a0			
16	e30a70df15a0622d5e1c1dcb2205a006	06	4a	02
	19e414b080d91da82fbeadf1a0fa3e4a			

Decrypted value: 1122334455667788

DTP AES-256 1-byte; different output set

ROUND	AES-256 INPUT/OUTPUT BLOCK	r	o	c
1	00000010ffffffffffffffffffffffff	04	3a	0a
	8f6307734f2bf814ca0c34b609ef3c3a			
2	6307734f2bf814ca0c34b609ef3c3a04	03	63	18
	7f513c44b78e077abe8642fa3e0ee063			
3	513c44b78e077abe8642fa3e0ee06303	01	57	0a
	10c60493ff20c9af57115211c50be857			
4	c60493ff20c9af57115211c50be85701	01	b8	03
	537f88b01588fc1de873e20133c847b8			
5	7f88b01588fc1de873e20133c847b801	06	d4	0a
	44a0ef375bda4a0c993fb4fcd6ce4fd4			
6	a0ef375bda4a0c993fb4fcd6ce4fd406	08	51	0b
	ebf609d5d9d536941444cfd6cd5bcc51			
7	f609d5d9d536941444cfd6cd5bcc5108	04	b0	18
	fdc2747d9da91dd711d1794bb23248b0			
8	c2747d9da91dd711d1794bb23248b004	07	65	04
	fdee05e508f6eecca476ba61fb470f65			
9	ee05e508f6eecca476ba61fb470f6507	07	e1	18
	4b749f270510d9e96f79f63d049935e1			
10	749f270510d9e96f79f63d049935e107	02	3e	0c
	c4d949adb89c545b701d2f0b0f3c653e			
11	d949adb89c545b701d2f0b0f3c653e02	00	bb	05
	12b5cdac616722c3a4918659790c5ebb			
12	b5cdac616722c3a4918659790c5ebb00	01	81	00

```
        6457d064b94f918721b187fa8d444181
  13    57d064b94f918721b187fa8d44418101   03  1a  03
        62d16a84f6257580465d0c187affeb1a
  14    d16a84f6257580465d0c187affeb1a03   00  71  09
        0c785062aca4049c92cd05b6b77a6671
  15    785062aca4049c92cd05b6b77a667100   06  a0  0a
        41e30a70df15a0622d5e1c1dcb2205a0
  16    e30a70df15a0622d5e1c1dcb2205a006   02  4a  18
        19e414b080d91da82fbeadf1a0fa3e4a

Encrypted value: KYKDKLYEYMFADJKY

 ROUND  AES-256 INPUT/OUTPUT BLOCK          c   o   r

   1    00000010ffffffffffffffffffffffff   0a  3a  04
        8f6307734f2bf814ca0c34b609ef3c3a
   2    6307734f2bf814ca0c34b609ef3c3a04   18  63  03
        7f513c44b78e077abe8642fa3e0ee063
   3    513c44b78e077abe8642fa3e0ee06303   0a  57  01
        10c60493ff20c9af57115211c50be857
   4    c60493ff20c9af57115211c50be85701   03  b8  01
        537f88b01588fc1de873e20133c847b8
   5    7f88b01588fc1de873e20133c847b801   0a  d4  06
        44a0ef375bda4a0c993fb4fcd6ce4fd4
   6    a0ef375bda4a0c993fb4fcd6ce4fd406   0b  51  08
        ebf609d5d9d536941444cfd6cd5bcc51
   7    f609d5d9d536941444cfd6cd5bcc5108   18  b0  04
        fdc2747d9da91dd711d1794bb23248b0
   8    c2747d9da91dd711d1794bb23248b004   04  65  07
        fdee05e508f6eecca476ba61fb470f65
   9    ee05e508f6eecca476ba61fb470f6507   18  e1  07
        4b749f270510d9e96f79f63d049935e1
  10    749f270510d9e96f79f63d049935e107   0c  3e  02
        c4d949adb89c545b701d2f0b0f3c653e
  11    d949adb89c545b701d2f0b0f3c653e02   05  bb  00
        12b5cdac616722c3a4918659790c5ebb
  12    b5cdac616722c3a4918659790c5ebb00   00  81  01
        6457d064b94f918721b187fa8d444181
  13    57d064b94f918721b187fa8d44418101   03  1a  03
        62d16a84f6257580465d0c187affeb1a
  14    d16a84f6257580465d0c187affeb1a03   09  71  00
        0c785062aca4049c92cd05b6b77a6671
  15    785062aca4049c92cd05b6b77a667100   0a  a0  06
        41e30a70df15a0622d5e1c1dcb2205a0
  16    e30a70df15a0622d5e1c1dcb2205a006   18  4a  02
        19e414b080d91da82fbeadf1a0fa3e4a

Decrypted value: 1122334455667788
```

HMAC-SHA1 1-BYTE DTP ENCRYPTION/DECRYPTION

```
Key: 0x0123456789abcdef111213141516171821222324252627283132333435363738
Cipher-scrambled data: 0x06020106080701040307010808040004
Rippled data: 0x04080001060303020709040804020600
```

DTP HMAC-SHA1 1-byte; same output set

```
 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK             r   o   c

   1     00000010ffffffffffffffffffffffffffffffff   04  a2  06
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2
   2     0efbae2dd2f3caf9f52e27324d4280f3d63fa204   08  ad  01
         ae13c79f04127e496669f3f6e9d95febb91e89ad
   3     13c79f04127e496669f3f6e9d95febb91e89ad08   00  8d  01
         999e898d364065238d05e182521c911953f7118d
   4     9e898d364065238d05e182521c911953f7118d00   01  ef  00
         e024fccb7b79adc00260b0435923d6a5c67a8eef
   5     24fccb7b79adc00260b0435923d6a5c67a8eef01   06  f0  06
         1af7c98cffdf704521a7bd5304c3b4b49f1b1df0
   6     f7c98cffdf704521a7bd5304c3b4b49f1b1df006   03  0e  07
         53a906f4f87693afe130c93252da898420eacf0e
   7     a906f4f87693afe130c93252da898420eacf0e03   03  d6  07
         5e6de16f79c1edb0df7a4729d0cefdbbf698cdd6
   8     6de16f79c1edb0df7a4729d0cefdbbf698cdd603   02  ce  08
         0b44ce5f8e9569e575639e0a5e3324e4b6d455ce
   9     44ce5f8e9569e575639e0a5e3324e4b6d455ce02   07  7a  09
         3030a8d22c0a2fc1826b5adea4c0be9ed373087a
  10     30a8d22c0a2fc1826b5adea4c0be9ed373087a07   09  38  05
         3e7ff882df221ed432b04a1e97bbae3be00d8c38
  11     7ff882df221ed432b04a1e97bbae3be00d8c3809   04  26  02
         addb3d5033d889a5ef5ade45232695c24c8d8026
  12     db3d5033d889a5ef5ade45232695c24c8d802604   08  93  05
         2fb1c84799da5ef5ea9c035fe6c2fa7f209cc293
  13     b1c84799da5ef5ea9c035fe6c2fa7f209cc29308   04  11  01
         24264ec6c8d1d05ca19000b14c9c4959cb5c9b11
  14     264ec6c8d1d05ca19000b14c9c4959cb5c9b1104   02  7c  06
         bbd0e1506730ca5cd88ee4d9ff1a070a457f1d7c
  15     d0e1506730ca5cd88ee4d9ff1a070a457f1d7c02   06  9c  02
         e3461378527b8e23782e14ef3d2949168f54699c
  16     461378527b8e23782e14ef3d2949168f54699c06   00  10  06
         68ca95f1dbe2b0afb420d8f8492b471a7f692510

Encrypted value: 6110677895251626

 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK             c   o   r

   1     00000010ffffffffffffffffffffffffffffffff   06  a2  04
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2
   2     0efbae2dd2f3caf9f52e27324d4280f3d63fa204   01  ad  08
         ae13c79f04127e496669f3f6e9d95febb91e89ad
   3     13c79f04127e496669f3f6e9d95febb91e89ad08   01  8d  00
         999e898d364065238d05e182521c911953f7118d
   4     9e898d364065238d05e182521c911953f7118d00   00  ef  01
```

	e024fccb7b79adc00260b0435923d6a5c67a8eef			
5	24fccb7b79adc00260b0435923d6a5c67a8eef01	06	f0	06
	1af7c98cffdf704521a7bd5304c3b4b49f1b1df0			
6	f7c98cffdf704521a7bd5304c3b4b49f1b1df006	07	0e	03
	53a906f4f87693afe130c93252da898420eacf0e			
7	a906f4f87693afe130c93252da898420eacf0e03	07	d6	03
	5e6de16f79c1edb0df7a4729d0cefdbbf698cdd6			
8	6de16f79c1edb0df7a4729d0cefdbbf698cdd603	08	ce	02
	0b44ce5f8e9569e575639e0a5e3324e4b6d455ce			
9	44ce5f8e9569e575639e0a5e3324e4b6d455ce02	09	7a	07
	3030a8d22c0a2fc1826b5adea4c0be9ed373087a			
10	30a8d22c0a2fc1826b5adea4c0be9ed373087a07	05	38	09
	3e7ff882df221ed432b04a1e97bbae3be00d8c38			
11	7ff882df221ed432b04a1e97bbae3be00d8c3809	02	26	04
	addb3d5033d889a5ef5ade45232695c24c8d8026			
12	db3d5033d889a5ef5ade45232695c24c8d802604	05	93	08
	2fb1c84799da5ef5ea9c035fe6c2fa7f209cc293			
13	b1c84799da5ef5ea9c035fe6c2fa7f209cc29308	01	11	04
	24264ec6c8d1d05ca19000b14c9c4959cb5c9b11			
14	264ec6c8d1d05ca19000b14c9c4959cb5c9b1104	06	7c	02
	bbd0e1506730ca5cd88ee4d9ff1a070a457f1d7c			
15	d0e1506730ca5cd88ee4d9ff1a070a457f1d7c02	02	9c	06
	e3461378527b8e23782e14ef3d2949168f54699c			
16	461378527b8e23782e14ef3d2949168f54699c06	06	10	00
	68ca95f1dbe2b0afb420d8f8492b471a7f692510			

Decrypted value: 1122334455667788

DTP HMAC-SHA1 1-byte; different output set

ROUND	HMAC-SHA1-32 INPUT/OUTPUT BLOCK	r	o	c
1	00000010ffffffffffffffffffffffffffffffff	04	a2	0a
	0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2			
2	0efbae2dd2f3caf9f52e27324d4280f3d63fa204	08	ad	19
	ae13c79f04127e496669f3f6e9d95febb91e89ad			
3	13c79f04127e496669f3f6e9d95febb91e89ad08	00	8d	0b
	999e898d364065238d05e182521c911953f7118d			
4	9e898d364065238d05e182521c911953f7118d00	01	ef	06
	e024fccb7b79adc00260b0435923d6a5c67a8eef			
5	24fccb7b79adc00260b0435923d6a5c67a8eef01	06	f0	0c
	1af7c98cffdf704521a7bd5304c3b4b49f1b1df0			
6	f7c98cffdf704521a7bd5304c3b4b49f1b1df006	03	0e	11
	53a906f4f87693afe130c93252da898420eacf0e			
7	a906f4f87693afe130c93252da898420eacf0e03	03	d6	09
	5e6de16f79c1edb0df7a4729d0cefdbbf698cdd6			
8	6de16f79c1edb0df7a4729d0cefdbbf698cdd603	02	ce	00
	0b44ce5f8e9569e575639e0a5e3324e4b6d455ce			
9	44ce5f8e9569e575639e0a5e3324e4b6d455ce02	07	7a	19
	3030a8d22c0a2fc1826b5adea4c0be9ed373087a			
10	30a8d22c0a2fc1826b5adea4c0be9ed373087a07	09	38	0d
	3e7ff882df221ed432b04a1e97bbae3be00d8c38			
11	7ff882df221ed432b04a1e97bbae3be00d8c3809	04	26	10
	addb3d5033d889a5ef5ade45232695c24c8d8026			

12	db3d5033d889a5ef5ade45232695c24c8d802604	08	93	19
	2fb1c84799da5ef5ea9c035fe6c2fa7f209cc293			
13	b1c84799da5ef5ea9c035fe6c2fa7f209cc29308	04	11	15
	24264ec6c8d1d05ca19000b14c9c4959cb5c9b11			
14	264ec6c8d1d05ca19000b14c9c4959cb5c9b1104	02	7c	16
	bbd0e1506730ca5cd88ee4d9ff1a070a457f1d7c			
15	d0e1506730ca5cd88ee4d9ff1a070a457f1d7c02	06	9c	06
	e3461378527b8e23782e14ef3d2949168f54699c			
16	461378527b8e23782e14ef3d2949168f54699c06	00	10	10
	68ca95f1dbe2b0afb420d8f8492b471a7f692510			

Encrypted value: KZLGMRJAZNQZVWGQ

ROUND	HMAC-SHA1-32 INPUT/OUTPUT BLOCK	c	o	r
1	00000010ffffffffffffffffffffffffffffffff	0a	a2	04
	0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2			
2	0efbae2dd2f3caf9f52e27324d4280f3d63fa204	19	ad	08
	ae13c79f04127e496669f3f6e9d95febb91e89ad			
3	13c79f04127e496669f3f6e9d95febb91e89ad08	0b	8d	00
	999e898d364065238d05e182521c911953f7118d			
4	9e898d364065238d05e182521c911953f7118d00	06	ef	01
	e024fccb7b79adc00260b0435923d6a5c67a8eef			
5	24fccb7b79adc00260b0435923d6a5c67a8eef01	0c	f0	06
	1af7c98cffdf704521a7bd5304c3b4b49f1b1df0			
6	f7c98cffdf704521a7bd5304c3b4b49f1b1df006	11	0e	03
	53a906f4f87693afe130c93252da898420eacf0e			
7	a906f4f87693afe130c93252da898420eacf0e03	09	d6	03
	5e6de16f79c1edb0df7a4729d0cefdbbf698cdd6			
8	6de16f79c1edb0df7a4729d0cefdbbf698cdd603	00	ce	02
	0b44ce5f8e9569e575639e0a5e3324e4b6d455ce			
9	44ce5f8e9569e575639e0a5e3324e4b6d455ce02	19	7a	07
	3030a8d22c0a2fc1826b5adea4c0be9ed373087a			
10	30a8d22c0a2fc1826b5adea4c0be9ed373087a07	0d	38	09
	3e7ff882df221ed432b04a1e97bbae3be00d8c38			
11	7ff882df221ed432b04a1e97bbae3be00d8c3809	10	26	04
	addb3d5033d889a5ef5ade45232695c24c8d8026			
12	db3d5033d889a5ef5ade45232695c24c8d802604	19	93	08
	2fb1c84799da5ef5ea9c035fe6c2fa7f209cc293			
13	b1c84799da5ef5ea9c035fe6c2fa7f209cc29308	15	11	04
	24264ec6c8d1d05ca19000b14c9c4959cb5c9b11			
14	264ec6c8d1d05ca19000b14c9c4959cb5c9b1104	16	7c	02
	bbd0e1506730ca5cd88ee4d9ff1a070a457f1d7c			
15	d0e1506730ca5cd88ee4d9ff1a070a457f1d7c02	06	9c	06
	e3461378527b8e23782e14ef3d2949168f54699c			
16	461378527b8e23782e14ef3d2949168f54699c06	10	10	00
	68ca95f1dbe2b0afb420d8f8492b471a7f692510			

Decrypted value: 1122334455667788

APPENDIX B: DTP 3-BYTE ENCRYPTION/DECRYPTION EXAMPLES

Following are examples when encrypting/decrypting with DTP in 3-byte mode. In all examples the plaintext is the numeric string value `1122334455667788`. Binary values are presented as hex values.

Input set is `{0 1 2 3 4 5 6 7 8 9}`.
Output set is either the same or the larger set `{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}`.

TDEA 3-BYTE DTP ENCRYPTION/DECRYPTION

```
Key: 0x0123456789abcdef1112131415161718212223242526272 8
Cipher-scrambled data: 0x04070405080205090401090701030500
Rippled data: 0x00060500000202070305060803070804
```

DTP TDEA 3-byte; same output set

```
 ROUND   TDEA INPUT BLOCK   TDEA OUTPUT BLOCK   r   o   c

   1     00000010ffffffff   056ee40032852045   00  45  09
                                               06  20  08
                                               05  85  08
   2     0032852045000605   6ed2c0600904af89   00  89  07
                                               00  af  05
                                               02  04  06
   3     600904af89000002   9af0f8035019395c   02  5c  04
                                               07  39  04
                                               03  19  08
   4     035019395c020703   30edb7541c3c2d80   05  80  03
                                               06  2d  01
                                               08  3c  08
   5     541c3c2d80050608   42a77dbb1fb7213f   03  3f  06
                                               07  21  00
                                               08  b7  01
   6     bb1fb7213f030708   e4239dbe674790de   04  de  06

Encrypted value: 9887564483186016

 ROUND   TDEA INPUT BLOCK   TDEA OUTPUT BLOCK   c   o   r

   1     00000010ffffffff   056ee40032852045   09  45  00
                                               08  20  06
                                               08  85  05
   2     0032852045000605   6ed2c0600904af89   07  89  00
                                               05  af  00
                                               06  04  02
   3     600904af89000002   9af0f8035019395c   04  5c  02
                                               04  39  07
                                               08  19  03
   4     035019395c020703   30edb7541c3c2d80   03  80  05
                                               01  2d  06
                                               08  3c  08
```

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK			
5	541c3c2d80050608	42a77dbb1fb7213f	06	3f	03
			00	21	07
			01	b7	08
6	bb1fb7213f030708	e4239dbe674790de	06	de	04

Decrypted value: 1122334455667788

DTP TDEA 3-byte; different output set

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	r	o	c
1	00000010ffffffff	056ee40032852045	00	45	11
			06	20	0c
			05	85	08
2	0032852045000605	6ed2c0600904af89	00	89	07
			00	af	13
			02	04	06
3	600904af89000002	9af0f8035019395c	02	5c	10
			07	39	0c
			03	19	02
4	035019395c020703	30edb7541c3c2d80	05	80	03
			06	2d	19
			08	3c	10
5	541c3c2d80050608	42a77dbb1fb7213f	03	3f	0e
			07	21	0e
			08	b7	09
6	bb1fb7213f030708	e4239dbe674790de	04	de	12

Encrypted value: RMIHTGQMCDZQOOJS

ROUND	TDEA INPUT BLOCK	TDEA OUTPUT BLOCK	c	o	r
1	00000010ffffffff	056ee40032852045	11	45	00
			0c	20	06
			08	85	05
2	0032852045000605	6ed2c0600904af89	07	89	00
			13	af	00
			06	04	02
3	600904af89000002	9af0f8035019395c	10	5c	02
			0c	39	07
			02	19	03
4	035019395c020703	30edb7541c3c2d80	03	80	05
			19	2d	06
			10	3c	08
5	541c3c2d80050608	42a77dbb1fb7213f	0e	3f	03
			0e	21	07
			09	b7	08
6	bb1fb7213f030708	e4239dbe674790de	12	de	04

Decrypted value: 1122334455667788

AES-256 3-BYTE DTP ENCRYPTION/DECRYPTION

```
Key: 0x0123456789abcdef111213141516171821222324252627283132333435363738
Cipher-scrambled data: 0x01010805030603030507070905010008
Rippled data: 0x04030101060804070702000103000602
```

DTP AES-256 3-byte; same output set

```
 ROUND   AES-256 INPUT/OUTPUT BLOCK           r   o   c

   1     00000010ffffffffffffffffffffffff   04  3a  02
         8f6307734f2bf814ca0c34b609ef3c3a   03  3c  03
                                            01  ef  00
   2     734f2bf814ca0c34b609ef3c3a040301   01  bb  08
         c0bc489674f55297e78c20bc5bbacdbb   06  cd  01
                                            08  ba  04
   3     9674f55297e78c20bc5bbacdbb010608   04  b8  08
         3978d5fee586cb46d88b52c72bba07b8   07  07  04
                                            07  ba  03
   4     fee586cb46d88b52c72bba07b8040707   02  23  07
         9ce673dc70de1f9616fb99b1ea763b23   00  3b  09
                                            01  76  09
   5     dc70de1f9616fb99b1ea763b23020001   03  4a  07
         910939b7c2a6349d6e0ebdbc5151744a   00  74  06
                                            06  51  07
   6     b7c2a6349d6e0ebdbc5151744a030006   02  19  07
         1c0e73aaa98e8301aed26dde800f8419

Encrypted value: 2308148437997677

 ROUND   AES-256 INPUT/OUTPUT BLOCK           c   o   r

   1     00000010ffffffffffffffffffffffff   02  3a  04
         8f6307734f2bf814ca0c34b609ef3c3a   03  3c  03
                                            00  ef  01
   2     734f2bf814ca0c34b609ef3c3a040301   08  bb  01
         c0bc489674f55297e78c20bc5bbacdbb   01  cd  06
                                            04  ba  08
   3     9674f55297e78c20bc5bbacdbb010608   08  b8  04
         3978d5fee586cb46d88b52c72bba07b8   04  07  07
                                            03  ba  07
   4     fee586cb46d88b52c72bba07b8040707   07  23  02
         9ce673dc70de1f9616fb99b1ea763b23   09  3b  00
                                            09  76  01
   5     dc70de1f9616fb99b1ea763b23020001   07  4a  03
         910939b7c2a6349d6e0ebdbc5151744a   06  74  00
                                            07  51  06
   6     b7c2a6349d6e0ebdbc5151744a030006   07  19  02
         1c0e73aaa98e8301aed26dde800f8419

Decrypted value: 1122334455667788
```

DTP AES-256 3-byte; different output set

```
 ROUND   AES-256 INPUT/OUTPUT BLOCK          r   o   c

   1     00000010ffffffffffffffffffffffff   04  3a  0a
         8f6307734f2bf814ca0c34b609ef3c3a   03  3c  0b
                                            01  ef  06
   2     734f2bf814ca0c34b609ef3c3a040301   01  bb  06
         c0bc489674f55297e78c20bc5bbacdbb   06  cd  03
                                            08  ba  0c
   3     9674f55297e78c20bc5bbacdbb010608   04  b8  06
         3978d5fee586cb46d88b52c72bba07b8   07  07  0e
                                            07  ba  0b
   4     fee586cb46d88b52c72bba07b8040707   02  23  0b
         9ce673dc70de1f9616fb99b1ea763b23   00  3b  07
                                            01  76  0f
   5     dc70de1f9616fb99b1ea763b23020001   03  4a  19
         910939b7c2a6349d6e0ebdbc5151744a   00  74  0c
                                            06  51  09
   6     b7c2a6349d6e0ebdbc5151744a030006   02  19  01
         1c0e73aaa98e8301aed26dde800f8419

Encrypted value: KLGGDMGOLLHPZMJB

 ROUND   AES-256 INPUT/OUTPUT BLOCK          c   o   r

   1     00000010ffffffffffffffffffffffff   0a  3a  04
         8f6307734f2bf814ca0c34b609ef3c3a   0b  3c  03
                                            06  ef  01
   2     734f2bf814ca0c34b609ef3c3a040301   06  bb  01
         c0bc489674f55297e78c20bc5bbacdbb   03  cd  06
                                            0c  ba  08
   3     9674f55297e78c20bc5bbacdbb010608   06  b8  04
         3978d5fee586cb46d88b52c72bba07b8   0e  07  07
                                            0b  ba  07
   4     fee586cb46d88b52c72bba07b8040707   0b  23  02
         9ce673dc70de1f9616fb99b1ea763b23   07  3b  00
                                            0f  76  01
   5     dc70de1f9616fb99b1ea763b23020001   19  4a  03
         910939b7c2a6349d6e0ebdbc5151744a   0c  74  00
                                            09  51  06
   6     b7c2a6349d6e0ebdbc5151744a030006   01  19  02
         1c0e73aaa98e8301aed26dde800f8419

Decrypted value: 1122334455667788
```

HMAC-SHA1 3-BYTE DTP ENCRYPTION/DECRYPTION

```
Key: 0x0123456789abcdef111213141516171821222324252627283132333435363738
Cipher-scrambled data: 0x06020106080701040307010808040004
Rippled data: 0x04080001060303020709040804020600
```

DTP HMAC-SHA1 3-byte; same output set

```
 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK              r   o   c

   1     00000010ffffffffffffffffffffffffffffffff   04  a2  06
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2   08  3f  01
                                                    00  d6  04
   2     ae2dd2f3caf9f52e27324d4280f3d63fa2040800   01  ab  02
         0fcd0a8e1a2f3610cd9b2aa714195505b040c4ab   06  c4  02
                                                    03  40  07
   3     8e1a2f3610cd9b2aa714195505b040c4ab010603   03  9e  01
         62558aad4029bd2e6dc6e0cc188bdc1a28f0de9e   02  de  04
                                                    07  f0  07
   4     ad4029bd2e6dc6e0cc188bdc1a28f0de9e030207   09  68  03
         9864899a9414d7b5d2165df27a17daa4eaacf468   04  f4  08
                                                    08  ac  00
   5     9a9414d7b5d2165df27a17daa4eaacf468090408   04  13  03
         7de4cc7604a900d117eac1a88c0f9689a371ca13   02  ca  04
                                                    06  71  09
   6     7604a900d117eac1a88c0f9689a371ca13040206   00  08  08
         81112f5df6705568c6cfe481e25b6c76d20d2908
```

```
Encrypted value: 6142271473803498
```

```
 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK              c   o   r

   1     00000010ffffffffffffffffffffffffffffffff   06  a2  04
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2   01  3f  08
                                                    04  d6  00
   2     ae2dd2f3caf9f52e27324d4280f3d63fa2040800   02  ab  01
         0fcd0a8e1a2f3610cd9b2aa714195505b040c4ab   02  c4  06
                                                    07  40  03
   3     8e1a2f3610cd9b2aa714195505b040c4ab010603   01  9e  03
         62558aad4029bd2e6dc6e0cc188bdc1a28f0de9e   04  de  02
                                                    07  f0  07
   4     ad4029bd2e6dc6e0cc188bdc1a28f0de9e030207   03  68  09
         9864899a9414d7b5d2165df27a17daa4eaacf468   08  f4  04
                                                    00  ac  08
   5     9a9414d7b5d2165df27a17daa4eaacf468090408   03  13  04
         7de4cc7604a900d117eac1a88c0f9689a371ca13   04  ca  02
                                                    09  71  06
   6     7604a900d117eac1a88c0f9689a371ca13040206   08  08  00
         81112f5df6705568c6cfe481e25b6c76d20d2908
```

```
Decrypted value: 1122334455667788
```

DTP HMAC-SHA1 3-byte; different output set

```
 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK              r   o   c

   1     00000010ffffffffffffffffffffffffffffffff   04  a2  0a
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2   08  3f  13
                                                    00  d6  06
   2     ae2dd2f3caf9f52e27324d4280f3d63fa2040800   01  ab  10
         0fcd0a8e1a2f3610cd9b2aa714195505b040c4ab   06  c4  14
                                                    03  40  0f
   3     8e1a2f3610cd9b2aa714195505b040c4ab010603   03  9e  05
         62558aad4029bd2e6dc6e0cc188bdc1a28f0de9e   02  de  10
                                                    07  f0  0d
   4     ad4029bd2e6dc6e0cc188bdc1a28f0de9e030207   09  68  09
         9864899a9414d7b5d2165df27a17daa4eaacf468   04  f4  0e
                                                    08  ac  18
   5     9a9414d7b5d2165df27a17daa4eaacf468090408   04  13  17
         7de4cc7604a900d117eac1a88c0f9689a371ca13   02  ca  16
                                                    06  71  0f
   6     7604a900d117eac1a88c0f9689a371ca13040206   00  08  08
         81112f5df6705568c6cfe481e25b6c76d20d2908

Encrypted value: KTGQUPFQNJOYXWPI

 ROUND   HMAC-SHA1-32 INPUT/OUTPUT BLOCK              c   o   r

   1     00000010ffffffffffffffffffffffffffffffff   0a  a2  04
         0d0efbae2dd2f3caf9f52e27324d4280f3d63fa2   13  3f  08
                                                    06  d6  00
   2     ae2dd2f3caf9f52e27324d4280f3d63fa2040800   10  ab  01
         0fcd0a8e1a2f3610cd9b2aa714195505b040c4ab   14  c4  06
                                                    0f  40  03
   3     8e1a2f3610cd9b2aa714195505b040c4ab010603   05  9e  03
         62558aad4029bd2e6dc6e0cc188bdc1a28f0de9e   10  de  02
                                                    0d  f0  07
   4     ad4029bd2e6dc6e0cc188bdc1a28f0de9e030207   09  68  09
         9864899a9414d7b5d2165df27a17daa4eaacf468   0e  f4  04
                                                    18  ac  08
   5     9a9414d7b5d2165df27a17daa4eaacf468090408   17  13  04
         7de4cc7604a900d117eac1a88c0f9689a371ca13   16  ca  02
                                                    0f  71  06
   6     7604a900d117eac1a88c0f9689a371ca13040206   08  08  00
         81112f5df6705568c6cfe481e25b6c76d20d2908

Decrypted value: 1122334455667788
```

APPENDIX C: DTP 1-BYTE ENCRYPTION OUTPUT EXAMPLES

Following are examples of encrypted output with DTP in 1-byte mode and similar input.
Input set is `{0 1 2 3 4 5 6 7 8 9}`.

TDEA 1-BYTE DTP ENCRYPTION

```
Key: 0x0123456789abcdef11121314151617182122232425262728

  Cleartext            Ciphertext
  ----------------     ----------------
  1111111111111110     1809690109935843
  1111111111111111     5398962855067735
  1111111111111112     3031728079031549
  1111111111111113     0885488278263060
  1111111111111114     2000065556545565
  1111111111111115     8711989839212461
  1111111111111116     1880051135771472
  1111111111111117     5600386835871348
  1111111111111118     5612876534719790
  1111111111111119     2491334180521499

  0111111111111111     4104367400656918
  1111111111111111     5398962855067735
  2111111111111111     1170950761194311
  3111111111111111     9127778985156060
  4111111111111111     4015167489890771
  5111111111111111     1989386910589229
  6111111111111111     3866964521938059
  7111111111111111     4920914873884385
  8111111111111111     7818340388422549
  9111111111111111     6317980754174959

  1111111111111112     3031728079031549
  1111111111111121     3721419733070755
  1111111111111211     2271235796021817
  1111111111112111     8265064779139896
  1111111111121111     7840734150234750
  1111111111211111     0764478136749240
  1111111112111111     6628822257135163
  1111111121111111     0806905799600880
  1111111211111111     8792607839950194
  1111112111111111     2490159911473520
  1111121111111111     5833673378237520
  1111211111111111     8267804635660439
  1112111111111111     9987205318161101
  1121111111111111     4851344613098245
  1211111111111111     3822864672243032
  2111111111111111     1170950761194311

  1111111111111111     5398962855067735
  111111111111111      190428738020365
```

11111111111111	86198844182676
1111111111111	7685733306027
111111111111	301157891182
11111111111	39233295904
1111111111	1341609634
111111111	178220196
11111111	12594366
1111111	9122011
111111	721360

AES-256 1-BYTE DTP ENCRYPTION

Key: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38

Cleartext	Ciphertext
1111111111111110	0576951358784919
1111111111111111	2609602046056978
1111111111111112	1459946433649857
1111111111111113	5416598199899814
1111111111111114	7632751742462774
1111111111111115	1552453184599914
1111111111111116	1919645473288561
1111111111111117	3568689614928354
1111111111111118	3726475403605484
1111111111111119	5816168042188730
0111111111111111	7159640395393720
1111111111111111	2609602046056978
2111111111111111	0613613614121230
3111111111111111	6800985337971436
4111111111111111	8435813801465539
5111111111111111	0168243394353640
6111111111111111	2820114146431001
7111111111111111	8701814267318874
8111111111111111	0157215721863555
9111111111111111	3360635310725283
1111111111111112	1459946433649857
1111111111111121	5906559126014409
1111111111111211	9342837959299762
1111111111112111	0682029252042910
1111111111121111	2883982482212076
1111111111211111	3608423411567804
1111111112111111	3182642444990687
1111111121111111	4384369006493911
1111111211111111	5910971502547096
1111112111111111	6791789383696374
1111121111111111	8059726692381241
1111211111111111	3389439501312968
1112111111111111	2433783773693442
1121111111111111	3448798356298641
1211111111111111	5587292422822164
2111111111111111	0613613614121230

1111111111111111	2609602046056978
111111111111111	129011644960192
11111111111111	84754950147999
1111111111111	6622471581534
111111111111	696811172263
11111111111	97069632553
1111111111	1530839171
111111111	085812048
11111111	69188952
1111111	7589665
111111	939212

HMAC_SHA1 1-BYTE DTP ENCRYPTION

Key: 0x0123456789abcdef1112131415161718212223242526272831323334353637 38

Cleartext	Ciphertext
1111111111111110	6269509720202858
1111111111111111	3328560629961058
1111111111111112	7914366785286056
1111111111111113	7220684448967092
1111111111111114	2800065136462224
1111111111111115	0008319719816636
1111111111111116	8035318350696780
1111111111111117	4385814814753974
1111111111111118	8070076078302728
1111111111111119	3781708166535370
0111111111111111	8792216927653745
1111111111111111	3328560629961058
2111111111111111	4117624186326712
3111111111111111	0482768704246969
4111111111111111	0526242021747699
5111111111111111	7271370046455841
6111111111111111	9806787033782489
7111111111111111	6577144626321384
8111111111111111	1838059843552039
9111111111111111	0556461275927030
1111111111111112	7914366785286056
1111111111111121	1517603026219833
1111111111111211	1111805328170443
1111111111112111	4542297319686975
1111111111121111	5395191371365950
1111111111211111	7377293073209183
1111111112111111	9070299219798643
1111111121111111	0648378776922603
1111111211111111	6322494272403583
1111112111111111	8075420362829227
1111121111111111	5884482368512231
1111211111111111	9249561243526206
1112111111111111	4794177718251881

1121111111111111	6273735451132837
1211111111111111	4595660723219624
2111111111111111	4117624186326712
1111111111111111	3328560629961058
111111111111111	243965891347340
11111111111111	18298415899826
1111111111111	0490561170222
111111111111	146125729526
11111111111	83957139055
1111111111	1972505580
111111111	189020280
11111111	70719308
1111111	2243321
111111	946958

APPENDIX D: DTP 3-BYTE ENCRYPTION OUTPUT EXAMPLES

Following are examples of encrypted output with DTP in 3-byte mode and similar input.
Input set is `{0 1 2 3 4 5 6 7 8 9}`.

TDEA 3-BYTE DTP ENCRYPTION

```
Key: 0x0123456789abcdef1112131415161718212223242526272 8

  Cleartext           Ciphertext
  ----------------    ----------------
  1111111111111110    1341410114186261
  1111111111111111    5887705576353277
  1111111111111112    3271432875019638
  1111111111111113    0629071471352363
  1111111111111114    2744468179156582
  1111111111111115    8764902233780254
  1111111111111116    1321064139244564
  1111111111111117    5106168367224125
  1111111111111118    5119271366988809
  1111111111111119    2137991887589111

  0111111111111111    4932584624053343
  1111111111111111    5887705576353277
  2111111111111111    1633781198785047
  3111111111111111    9081643761316717
  4111111111111111    4848058095628346
  5111111111111111    1487851041239589
  6111111111111111    3018415449153347
  7111111111111111    4752932105201803
  8111111111111111    7584574502190681
  9111111111111111    6489469482390273

  1111111111111112    3271432875019638
  1111111111111121    3999422757112991
  1111111111111211    2975172836879310
  1111111111112111    8224394121557387
  1111111111121111    7516639630800311
  1111111111211111    0569982553573139
  1111111112111111    6763516478967743
  1111111121111111    0648489088967686
  1111111211111111    8748488869541432
  1111112111111111    2136337137109150
  1111121111111111    5323038140568133
  1111211111111111    8226069928514303
  1112111111111111    9880706796915674
  1121111111111111    4691401919030639
  1211111111111111    3077783564116107
  2111111111111111    1633781198785047

  1111111111111111    5887705576353277
  111111111111111     197186055841237
  11111111111111      83267274346147
```

1111111111111	7378634418405
111111111111	339939022627
11111111111	30441182339
1111111111	1297202968
111111111	100640998
11111111	13314610
1111111	9314687
111111	764067

AES-256 3-BYTE DTP ENCRYPTION

Key: 0x0123456789abcdef111213141516171821222324252627283132333435363738

Cleartext	Ciphertext
1111111111111110	0201306288741847
1111111111111111	2779569070612537
1111111111111112	1480505525375165
1111111111111113	5278907487027472
1111111111111114	7587768528088988
1111111111111115	1500757035259932
1111111111111116	1970203975186609
1111111111111117	3555846673107040
1111111111111118	3765234430530904
1111111111111119	5653019458242499
0111111111111111	7051834235154580
1111111111111111	2779569070612537
2111111111111111	0383022130808477
3111111111111111	6487090052926665
4111111111111111	8871804208240199
5111111111111111	0866964695006836
6111111111111111	2995343405609147
7111111111111111	8193180849442383
8111111111111111	0856948782064765
9111111111111111	3376239158193344
1111111111111112	1480505525375165
1111111111111121	5754441823212872
1111111111111211	9167043878333012
1111111111112111	0352949149196765
1111111111121111	2959994620204973
1111111111211111	3693775760447764
1111111112111111	3198091607158228
1111111121111111	4759597929075368
1111111211111111	5761812979877503
1111112111111111	6365782259316900
1111121111111111	8468036454289997
1111211111111111	3395163298005285
1112111111111111	2587357062463009
1121111111111111	3464586033266149
1211111111111111	5354902720221845
2111111111111111	0383022130808477

```
  1111111111111111     2779569070612537
  111111111111111      167205852563363
  11111111111111       87716217649863
  1111111111111        6044860980622
  111111111111         649888599326
  11111111111          90664094819
  1111111111           1544246946
  111111111            021588390
  11111111             64193411
  1111111              7008651
  111111               980819
```

HMAC-SHA1 3-BYTE DTP ENCRYPTION

```
Key: 0x0123456789abcdef111213141516171821222324252627283132333435363738

  Cleartext            Ciphertext
  ----------------     ----------------
  1111111111111110     6223839375875483
  1111111111111111     3344505725258724
  1111111111111112     7436357568882812
  1111111111111113     7700693478831896
  1111111111111114     2407401823253443
  1111111111111115     0415571536019491
  1111111111111116     8147917310571922
  1111111111111117     4716664885164075
  1111111111111118     8189680859258080
  1111111111111119     3738960742222085

  0111111111111111     8882224990288155
  1111111111111111     3344505725258724
  2111111111111111     4517522863228991
  3111111111111111     0816952507569905
  4111111111111111     0982736135214573
  5111111111111111     7750009468463199
  6111111111111111     9077819571788572
  7111111111111111     6543094658961675
  8111111111111111     1160586789434919
  9111111111111111     0918481928694429

  1111111111111112     7436357568882812
  1111111111111121     1811220762351146
  1111111111111211     1431313241468955
  1111111111112111     4919296964524057
  1111111111121111     5700093519718400
  1111111111211111     7811348163400973
  1111111112111111     9255358958299843
  1111111121111111     0060352383741700
  1111111211111111     6394070906179572
  1111112111111111     8184218957763873
  1111121111111111     5217154955528778
  1111211111111111     9452838568724036
  1112111111111111     4152531244142947
  1121111111111111     6231417419352532
  1211111111111111     4964847638641764
```

2111111111111111	4517522863228991
1111111111111111	3344505725258724
111111111111111	236424304818623
11111111111111	19214101886801
1111111111111	0017975807180
111111111111	101555375734
11111111111	83064256525
1111111111	1562492985
111111111	186610629
11111111	77588932
1111111	2619929
111111	961571