Contents

Add normative reference

NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, 2001

11 Security-oriented mechanisms

11.1 Introduction

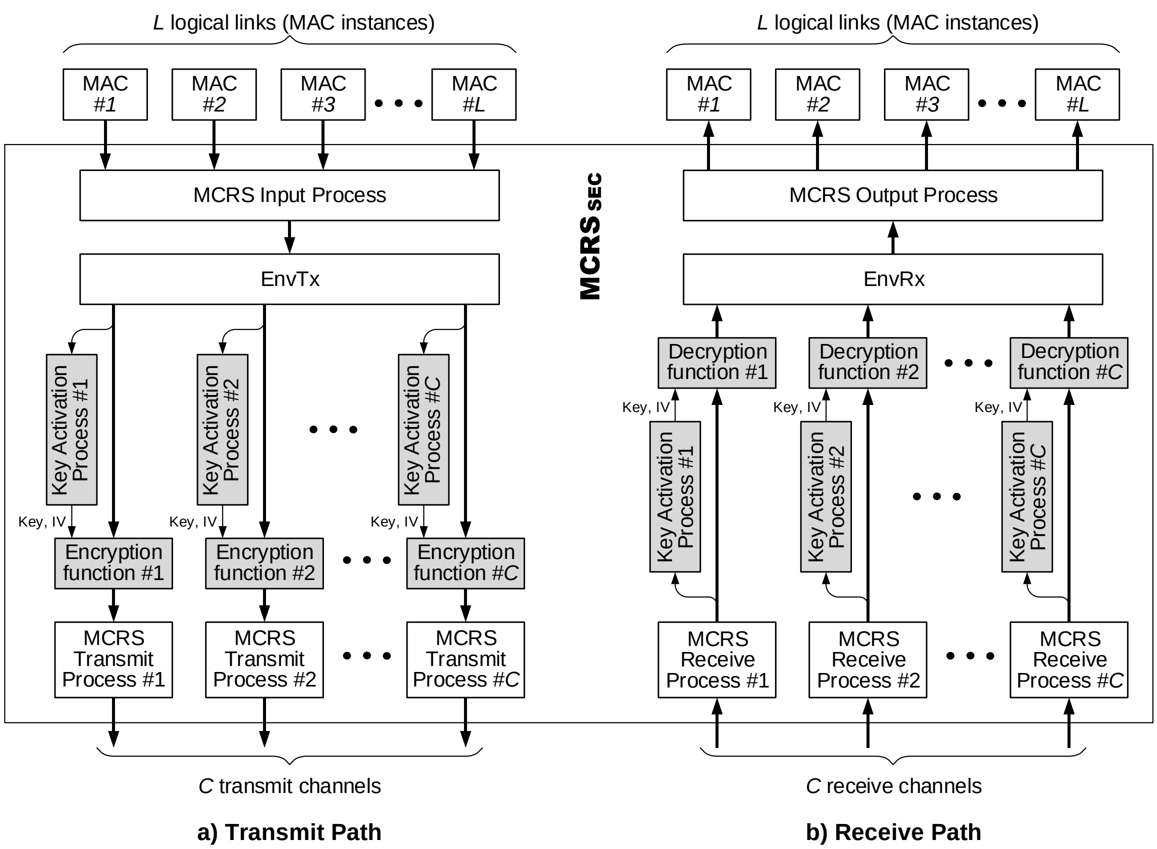
11.2 Overview of SIEPON.4 security architecture

11.2.1 Encryption entity

11.2.1.1 Mapping between the encryption entities and logical links

11.2.2 Location of encryption/decryption functions

The Multi-channel Reconciliation Sublayer (MCRS) reconciles *L* logical links (i.e., MAC instances) above the sublayer with *C* physical layer channels below it. The MCRS is defined in IEEE Std 802.3, Clause 143. When security mechanisms are implemented within the MCRS sublayer, such enhanced sublayer is referred to as Secure MCRS ($MCRS_{SEC}$) sublayer. The encryption function is located in the transmit path of the $MCRS_{SEC}$ sublayer, as illustrated in Figure 11-1(a), and the decryption function is located in the receive path of the $MCRS_{SEC}$ sublayer, as illustrated in Figure 11-1(b).



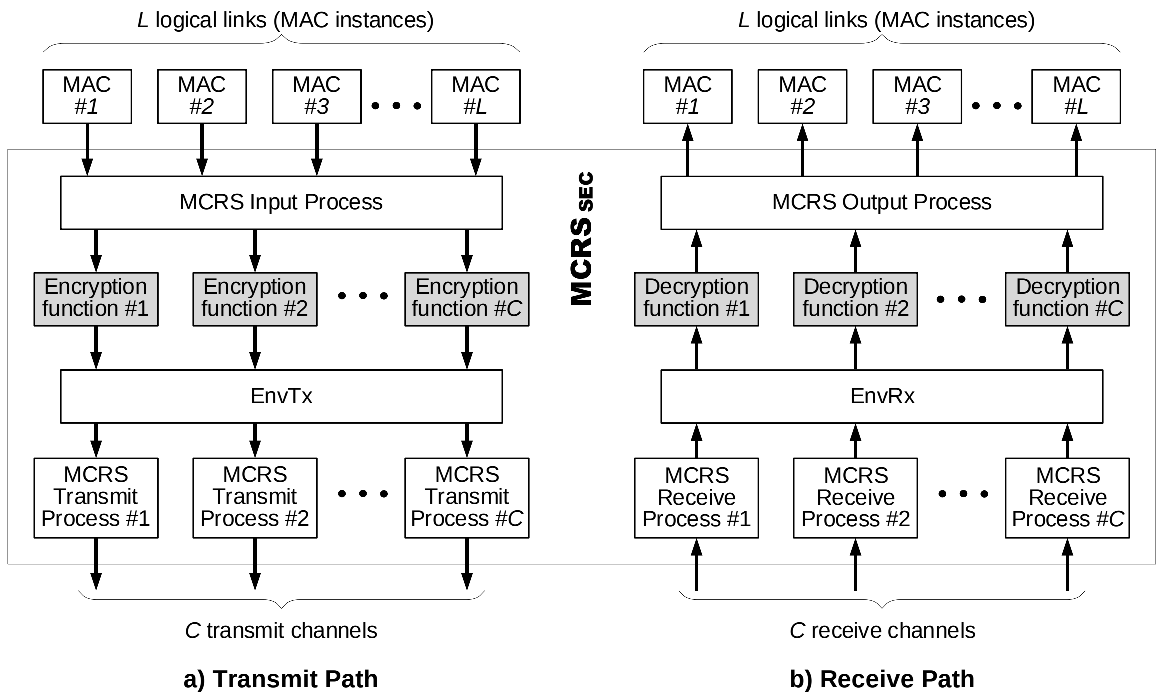


Figure 11-1– Location of encryption/decryption function within $MCRS_{SEC}$

~~A separate instance of the encryption function is located within every transmit channel between the *EnvTx* buffer and the MCRS Transmit Process. The encryption function is driven by the Encryption Key Activation process defined in 11.6.2.~~

~~A separate instance of the decryption function is located within every receive channel between the MCRS Receive Process and the *EnvRx* buffer. The decryption function is driven by the Decryption Key Activation process defined in 11.6.2.~~

In the MCRS transmit data path, a separate instance of the encryption function is located within every channel between the MCRS Input Process and the *EnvTx* buffer. In the MCRS receive data path, a separate instance of the decryption function is located within every channel between the *EnvRx* buffer and the MCRS Output Process.

11.2.3 Encryption function block diagram

Each instance of the encryption function includes the Encryption Key Activation process (see 11.6.2) and the Encryption process (see 11.7.2). The block diagram of the encryption function is illustrated in Figure 11-x2.

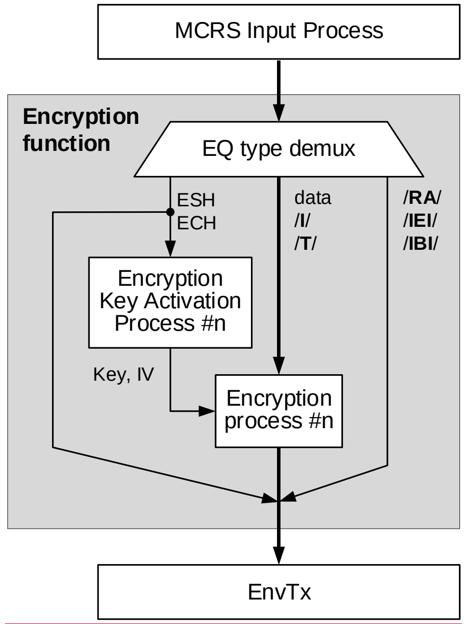


Figure 11-x2 – Encryption function block diagram.

The Encryption Key Activation process detects the transmission of either the envelope start header (ESH) or the envelope continuation header (ECH) and uses the data inside the header to generate an initialization vector IV and the encryption key. These two parameters are passed to the Encryption process which encrypts the given envelope as one cryptographic message.

The encryption function encrypts the envelope payload, which includes data EQs, idle control characters /I/, and termination control characters /T/. The envelope header itself bypasses the Encryption process and is transmitted unencrypted.

The inter-envelope control characters include rate adjustment /RA/, inter-envelope-idle /IEI/, and inter-burst idle /IBI/ (see IEEE Std 802.3, 142.2.1). These control characters bypass the Encryption process and are transmitted unencrypted.

11.2.4 Decryption function block diagram

Each instance of the decryption function includes the Decryption Key Activation process (see 11.6.2) and the Encryption process (see 11.7.3). The block diagram of the encryption function is illustrated in Figure 11-x3.

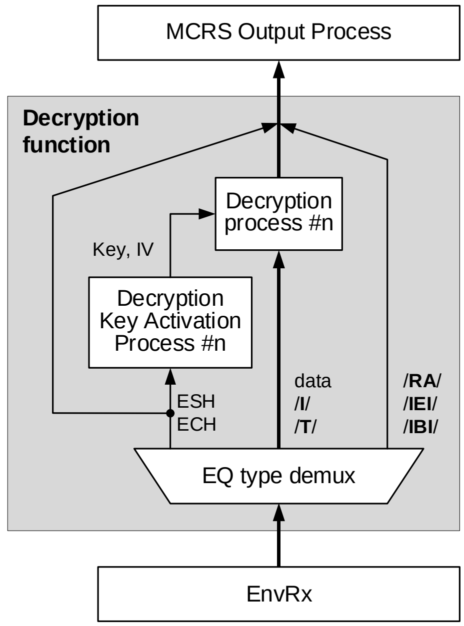


Figure 11-x3 – Encryption function block diagram.

The Decryption Key Activation process detects the reception of either the envelope start header (ESH) or the envelope continuation header (ECH) and uses the data inside the header to generate an initialization vector IV and the decryption key. These two parameters are passed to the Decryption process which decrypts the given envelope as one cryptographic message.

The decryption function decrypts only the envelope payload, which includes data EQs, idle control characters /I/, and termination control characters /T/. The envelope header itself is received unencrypted and bypasses the Decryption process.

The inter-envelope control characters include rate adjustment /RA/, inter-envelope-idle /IEI/, and inter-burst idle /IBI/ (see IEEE Std 802.3, 142.2.1). These control characters are received unencrypted and bypass the Decryption process.

~~11.2.3~~11.2.5 Latency requirements

The latency introduced into the MCRS transmit path by the encryption function (see Figure 11-1(a)) shall remain constant (to within 1 EQT), regardless of whether the encryption is enabled or disabled.

The latency introduced into the MCRS receive path by the decryption function (see Figure 11-1(b)) shall remain constant (to within 1 EQT), regardless of whether the decryption is enabled or disabled.

~~11.2.4~~11.2.6 Establishment of security mechanisms

11.3 ONU authentication

11.4 Initial Security Association Key exchange

11.5 Session key distribution protocol

11.6 Session key activation protocol

11.7 Cryptographic method

11.7.1 Introduction

In SIEPON.4 systems, the OLT and ONUs encrypt data using the AES Counter mode (AES-CTR). The AES-CTR is a confidentiality mode that applies the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are XOR-ed with the plaintext to produce the ciphertext, and vice versa. The AES-CTR mode requires that all counter values be distinct across all of the messages that are encrypted under the given key. For the detailed specification of the AES-CTR refer to NIST SP 800-38A, 6.5.

11.7.1.1 Envelope-based encryption

The concept of transmission envelope is defined in IEEE Std 802.3, 143.2.4.2. An envelope encapsulates continuous transmission by a specific MAC instance (LLID) on one MCRS channel.

In SIEPON.4 cryptographic method, the encryption is based on an envelope structure, i.e., an envelope payload constitutes the plaintext message to be encrypted. The envelope headers themselves are not encrypted. An entire envelope payload is encrypted using the same session key. A new session key may only activate during the reception or transmission of an envelope header (refer to Session Key Activation protocol in 11.6).

The cipher block size is 128 bits. Each block of plaintext includes exactly two EQs. Some plaintext blocks are only partially-encrypted, i.e., the above-mentioned XOR operation is applied to only a portion of the plaintext block. The reason for this is explained in 11.7.2.1.

11.7.1.2 Location of the encryption/decryption functional blocks

The encryption and decryption functional blocks are located within the secure MCRS ($MCRS_{SEC}$) sublayer, as detailed in 11.2.2.

11.7.2 Encryption process

The encryption process applies the forward cipher function to each counter block, and the resulting output blocks are XOR-ed with the corresponding plaintext blocks to produce the ciphertext blocks (see Figure 11-xx1).

The first counter block in a message (Counter 1) is initialized to the value called Initialization Vector (IV). The IV value used for the encryption is calculated by the OLT encryption key activation process (see Figure 11-8) and by the ONU encryption key activation process (see Figure 11-10). Every subsequent counter block associated with the given message is constructed by incrementing the value of the previous counter block by 1.

If the envelope payload length is odd, the last block will only contain one EQ. In such case, the most significant 64 bits of the last output block are used for the XOR operation and the remaining 64 least significant bits of the last output block are discarded.

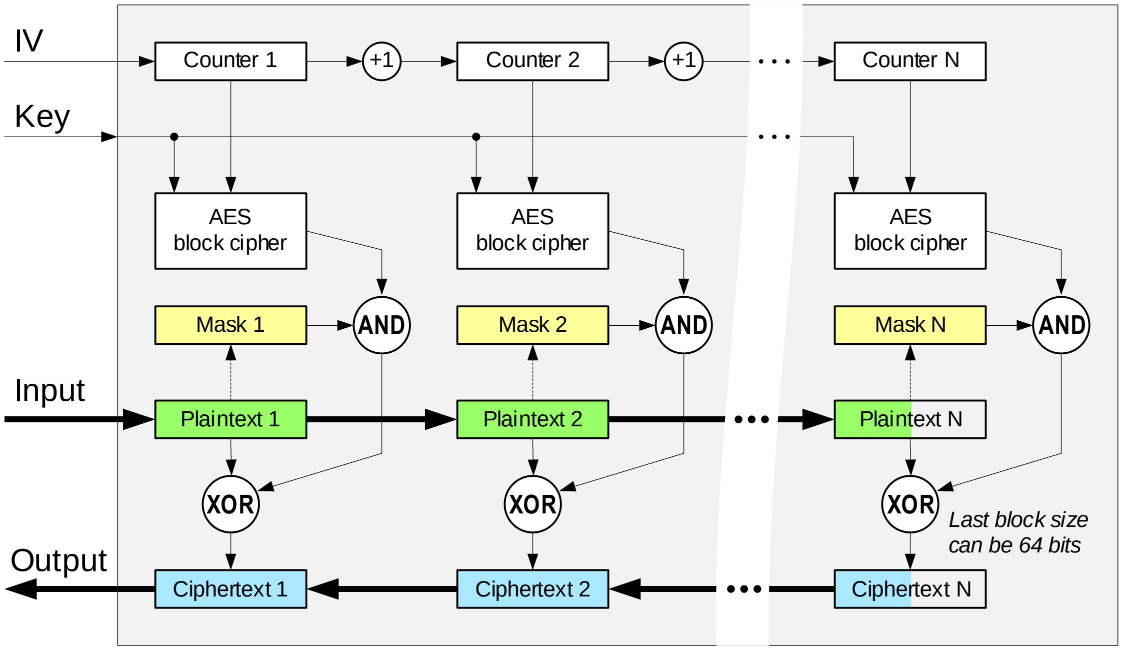


Figure 11-xx – Block diagram of the encryption process

For every block of plaintext, a mask is constructed to block-out the control characters (see 11.7.5.2). This mask is AND-ed with the output of the AES block cipher, resulting in the unencrypted control characters being placed in the ciphertext blocks.

In the encryption process, the forward cipher block operations can be performed in parallel. Moreover, the forward cipher functions can be applied to the counters prior to the availability of the plaintext data, if the corresponding counter block values can be determined.

11.7.3 Decryption process

The decryption process applies the forward cipher function to each counter block, and the resulting output blocks are XOR-ed with the corresponding ciphertext blocks to recover the plaintext blocks (see Figure 11-xx2).

Similarly to that in the encryption process, the first counter block in a message (Counter 1) is initialized to the value called Initialization Vector (IV). The IV value used for the decryption is calculated by the OLT and ONU decryption key activation process (see Figure 11-9). Every subsequent counter block associated with the given message is constructed by incrementing the value of the previous counter block by 1.

For a given encrypted message (i.e., an envelope), the IV and the subsequent counter block values applied by the decryption process match the IV and the counter values that were previously applied by the encryption process to encrypt the same message.

If the envelope payload length is odd, the last block will only contain one EQ. In such case, the most significant 64 bits of the last output block are used for the XOR operation and the remaining 64 least significant bits of the last output block are discarded.

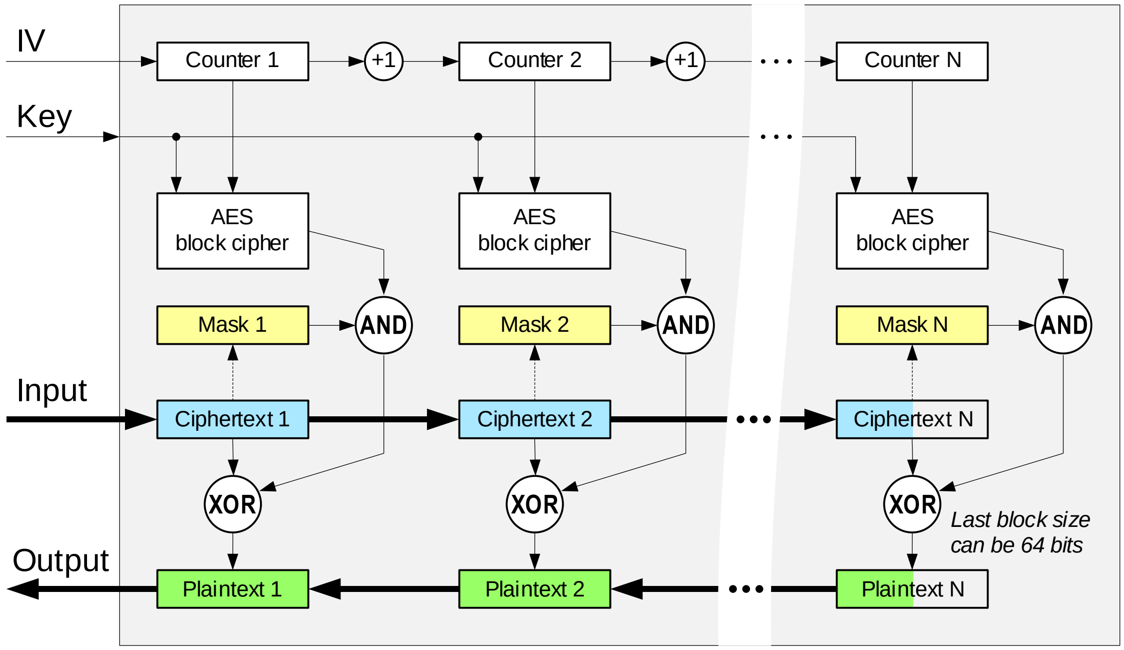


Figure 11-xx – Block diagram of the decryption process

For every block of ciphertext, a mask is constructed to block-out the control characters (see 11.7.5.2). This mask is AND-ed with the output of the AES block cipher, resulting in the unencrypted control characters being placed in the plaintext blocks.

In the decryption process, the forward cipher block operations can be performed in parallel. Moreover, the forward cipher functions can be applied to the counters prior to the availability of the ciphertext data, if the corresponding counter block values can be determined.

11.7.4 Initialization Vector (IV) construction

The sequence of counters must have the property that each block in the sequence is different from every other block. This condition is not restricted to a single message; across all of the messages that are encrypted under the given key, all of the counters must be distinct. This condition is satisfied by ensuring that IV values calculated for every message are distinct for any message (envelope) encrypted with the same key.

To encrypt a message, the IV is calculated by the encryption key activation processes at the OLT (see Figure 11-8) and at the ONU (see Figure 11-10) when an Envelope Start Header (ESH) is observed in the transmit path of the $MCRS_{SEC}$ sublayer.

To decrypt a message, the IV is calculated by the decryption key activation processes at the OLT and the ONU (see Figure 11-9) when an Envelope Start Header (ESH) is observed in the receive path of the $MCRS_{SEC}$ sublayer.

The data within the envelope header together with the index of the channel on which this envelope header was transmitted or received comprise the input parameters to the `CalculateIV(…)` function that derives the IV values in the above mentioned processes (see 11.6.2.2). It is critical that for any given encrypted message (envelope), the IV calculated by the decryption key activation process matched the IV calculated by the encryption key activation process.

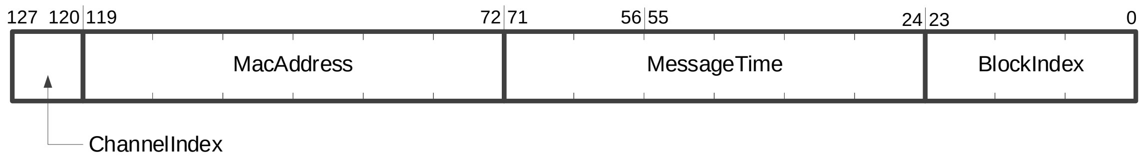


Figure 11-xx – Structure of the Initialization Vector

The structure of the IV is illustrated in Figure 11-xx. The IV consists of the following four fields:

ChannelIndex – Index of the channel on which the encrypted message is being transmitted or received. The most significant bit (bit 127) represents the direction (0 – downstream; 1- upstream), and bits [126:120] represent the channel number. For example, the value 0x01 represents the downstream channel 1 (DC1) and the value 0x80 represents the upstream channel 0 (UC0). The MCRS channels are explained in IEEE 802.3, 143.4.1.1.

The inclusion of this field ensures that in a situation when multiple ESHs to/from the same ONU are transmitted at the same time (i.e., the IVs have the same *MessageTime* filed value) on different channels, their associated IV values would still be distinct.

MacAddress – This is the MAC address of the device that encrypted the given envelope. In the downstream direction, this is the MAC address associated with the PON port of the OLT. In the upstream direction, this is the MAC address associated with the PON port of the transmitting ONU.

To calculate the IV for the decryption, the ONU uses the OLT's MAC address known to it from the MPCP registration step. The OLT also has a prior knowledge of MAC addresses of all connected ONUs, but it needs to determine which specific ONU sourced the given envelope. It does that by first extracting the LLID value from the ESH and then looking up the MAC address associated with this LLID.

MessageTime – This field represents a timestamp of *cipher clock* captured at the moment when the encryption key activation process observes the ESH in the MCRS$_{SEC}$ transmit path or the decryption key activation process observes the ESH in the MCRS$_{SEC}$ receive path. The cipher clock runs synchronously with the MPCP clock, but otherwise is a distinct clock, as explained in 11.7.4.1.

This field ensures that all the counter block values used on the same channel (i.e., the IVs have the same *ChannelIndex* values) and with the same session key are distinct.

BlockIndex – The block index field is set to zero when an envelope header is detected. This field is incremented by 1 for every subsequent counter block in the same envelope.

11.7.4.1 Cipher clock

The cipher clock is a 48-bit counter that runs synchronously with the MPCP clock (*LocalTime*), but is a distinct clock. The OLT and the ONUs contain versions of this clock that is used as a timestamp source for the IV field *MessageTime*. At the OLT, a single clock, referred to as *CipherClock* is used for IV construction in both the encryption and the decryption functions. At the ONU, there are two instances of the cipher clock: the *TxCipherClock* that is used to construct the IV for the encryption function, and the *RxCipherClock* that is used to construct the IV for the decryption function.

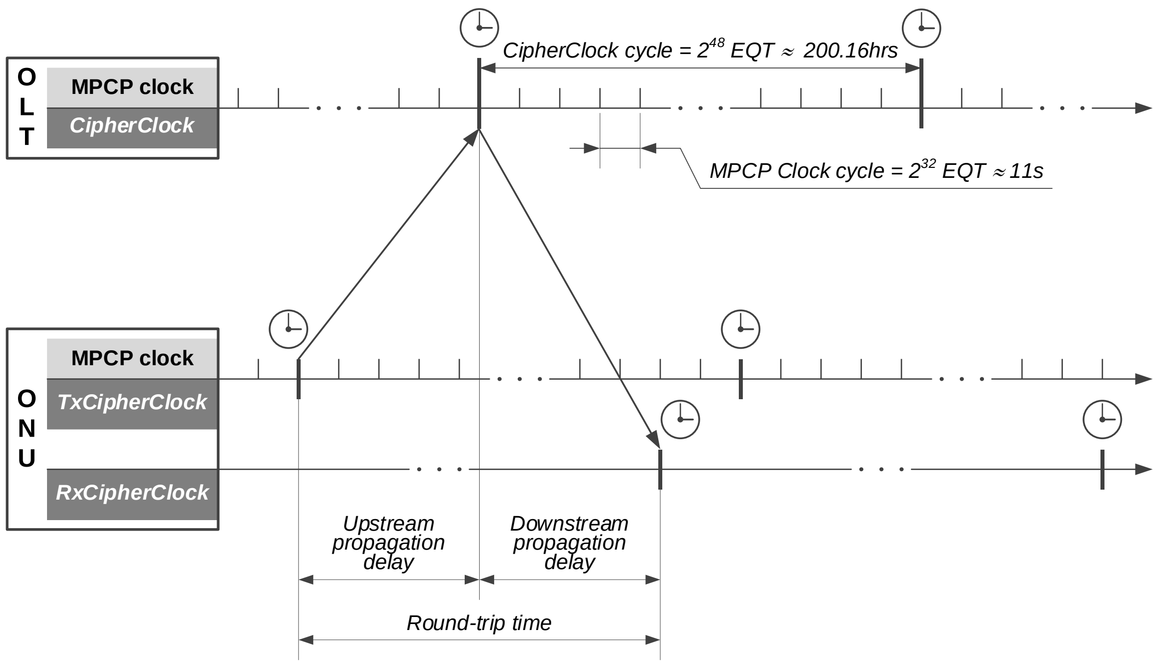


Figure 11-xx – Relationship of OLT's *CipherClock* and ONU's *TxCipherClock* and *RxCipherClock*

The MPCP clock is a 32-bit counter that increments by one every EQT. The initial synchronization of the MPCP clock takes place during ONU's MPCP discovery and registration and is described in IEEE Std 802.3, 144.3.1.1.

The origin point of the MPCP clock (counter) at the ONU is advanced relative to the origin point of the MPCP clock at the OLT by the upstream propagation time. The desired effect of such shift is that an envelope header transmitted by the ONU at its local MPCP time T_i is received by the OLT also at its local MPCP time T_i.

The *CipherClock* in the OLT is an extension of the OLT MPCP clock constructed by prepending 16 most-significant bits to the MPCP clock, i.e., `LocalTime` counter (see IEEE 802.3, 144.2.1.1). The carry-over bit from the `LocalTime` counter increments the first bit of the 16-bit extension portion (i.e., the bit 32 of the 48-bit *CipherClock* counter).

The *TxCipherClock* in the ONU is an extension of the ONU MPCP clock and is constructed in a manner similar to the OLT *CipherClock* construction. Because the OLT *CipherClock* and the ONU *TxCipherClock* extend their respective MPCP clocks, they preserve their relative shift, ensuring that the *MessageTime* value used to construct the IV for the encryption at the OLT matches the *MessageTime* value used to construct the IV for the decryption at the ONU (see Figure 11-xxx).

The *RxCipherClock* at the ONU is a separate 48-bit clock increments synchronously with the ONU MPCP clock, but is not an extension of the MPCP clock (i.e., the low 32 bits of the *RxCipherClock* are not equal to ONU's MPCP `LocalTime` value). The origin point of the *RxCipherClock* at the ONU is delayed relative to the origin point of the *CipherClock* at the OLT by the downstream propagation time (see Figure 11-xxx). The desired effect of such shift is that an envelope header transmitted by the OLT when its *CipherClock* value is T_i is received by the ONU at its *RxCipherClock* value is also T_i.

11.7.4.1.1 Cipher clock alignment in the upstream

The MCRS defined in IEEE Std 802.3, Clause 143 ensures that an envelope header (EH) transmitted by the ONU at a specific local time value is received at that exact local time at the OLT. To achieve that, the ONU sets the EPAM field in the EH to equal 6 least significant bits of its MPCP time (`EH.EPAM = LocalTime[5:0]`).

At the OLT, this EH is received (i.e., is written) into the `EnvRx` buffer into row with index equal to `EH.EPAM`. This EH is then read from the `EnvRx` buffer at the exact time when the OLT's `LocalTime[5:0]` are equal to the row index (i.e., when `LocalTime[5:0] = EH.EPAM`). As the 6 LSB are aligned, so are the entire extended MPCP clock values at the ONU and OLT are equal.

Since the *CipherClock* at the OLT and the *TxCipherClock* at the ONU are the extensions of their respective MPCP clocks, it follows that the value of ONU's *TxCipherClock* latched at the moment when the EH is written into `EnvTx` buffer at the ONU matches the value of OLT's *CipherClock* latched at the moment when the ESH is read from the `EnvRx` buffer at the OLT.

11.7.4.1.2 Cipher clock alignment in the downstream

In the downstream direction, the OLT sets the EPAM field in the EH to equal 6 least significant bits of its MPCP time (`EH.EPAM = LocalTime[5:0]`).

At the ONU, this EH is received (i.e., is written) into the `EnvRx` buffer into row with index equal to `EH.EPAM`. This EH is then read from the `EnvRx` buffer at the exact time when the 6 LSB of the ONU's *RxCipherClock* are equal to the row index. (Note however, that ONU's `LocalTime[5:0] ≠ EH.EPAM` because the ONU's MPCP clock is advanced by the upstream propagation time, see Figure 11-xxx.)

As the 6 LSB are aligned, it follows that the value of OLT's *CipherClock* latched at the moment when the EH was written into `EnvTx` buffer at the OLT matches the value of ONU's *RxCipherClock* latched at the moment when the EH was read from the `EnvRx` buffer at the ONU.

11.7.4.1.3 Initial cipher clock synchronization

TBD

11.7.4.1.4 Implementation options (informative)

At the OLT, the *CipherClock* and MPCP clock can share the same 48-bit variable (register), with the MPCP clock occupying the 32 least-significant bits (i.e., `LocalTime[31:0] =CipherClock[31:0]`).

At the ONU, an observation can be made that the time frame reference of *RxCipherClock* lags behind the time frame reference of the *TxCipherClock* by a fixed interval equal to ONU's round trip time. Thus, it is possible to represent the MPCP clock, the *TxCipherClock* and the *RxCipherClock* by the same 48-bit variable (register). The *TxCipherClock* is represented by the full register value, while the ONU MPCP clock is represented by the 32 least-significant bits (i.e., `LocalTime[31:0] =TxCipherClock[31:0]`). The *RxCipherClock* can be derived by subtracting the round-trip time (a fixed constant) from the value of the *TxCipherClock*: `RxCipherClock[47:0] = TxCipherClock[47:0] – RTT.`

11.7.4.2 CalculateIV(…) function

The function `CalculateIV(ch, eh)` is used by the encryption and decryption key activation processes in the OLT and ONUs. In each of these processes, the behavior of this function is similar at the high level, but differs in specific minor details, as explained below. This function executes within one MPCP clock cycle (in under one EQT), therefore the 6 least-significant bits of the relevant cipher clock counter match the value of the EPAM field of the EH (see IEEE Std 802.3, 143.3.2).

In the OLT encryption key activation process, the function `CalculateIV(ch, eh)` is called at the moment when an envelope header (EH) `eh` is observed in the MCRS transmit path on channel `ch` (see Figure 11-8). The following is an example definition of the function `CalculateIV(ch, eh)`:

```
int128 CalculateIV( ch, eh )
{
       iv.ChannelIndex = ch;                   // Channel index
       iv.MacAddress   = OLT_MAC_ADDRESS;      // Known constant
       iv.MessageTime  = CipherClock;          // Latch OLT’s cipher clock
       iv.BlockIndex   = 0;                    // Reset block index
       return iv;
}
```

In the OLT decryption key activation process, the function `CalculateIV(ch, eh)` is called at the moment when an envelope header (EH) `eh` is observed in the MCRS receive path on channel `ch` (see Figure 11-9). The following is an example definition of the function `CalculateIV(ch, eh)`:

```
int128 CalculateIV( ch, eh )
{
       iv.ChannelIndex = ch;                   // Channel index
       iv.MacAddress   = MacAddr[eh.llid];     // MAC address table lookup
       iv.MessageTime  = CipherClock;          // Latch OLT’s cipher clock
       iv.BlockIndex   = 0;                    // Reset block index
       return iv;
}
```

Note that, in this function, the OLT needs to perform a table lookup to retrieve the MAC address associated with a given LLID value.

In the ONU encryption key activation process, the function `CalculateIV(ch, eh)` is called at the moment when an envelope header (EH) `eh` is observed in the MCRS transmit path on channel `ch` (see Figure 11-10). The following is an example definition of the function `CalculateIV(ch, eh)`:

```
int128 CalculateIV( ch, eh )
{
       iv.ChannelIndex = ch;                   // Channel index
       iv.MacAddress   = ONU_MAC_ADDRESS;      // Known constant
       iv.MessageTime  = TxCipherClock;        // Latch ONU’s tx. cipher clock
       iv.BlockIndex   = 0;                    // Reset block counter
       return iv;
}
```

In the ONU decryption key activation process, the function `CalculateIV(ch, eh)` is called at the moment when an envelope header (EH) `eh` is observed in the MCRS receive path on channel `ch` (see Figure 11-9). The following is an example definition of the function `CalculateIV(ch, eh)`:

```
int128 CalculateIV( ch, eh )
{
       iv.ChannelIndex = ch;                   // Channel index
```

```
        iv.MacAddress   = OLT_MAC_ADDRESS;       // Learned at registration
        iv.MessageTime  = RxCipherClock;         // Latch ONU’s rx. cipher clock
        iv.BlockIndex   = 0;                     // Reset block index
        return iv;
}
```

11.7.5 Encrypted envelope format

To encrypt a message, an envelope payload is divided in 128-bit blocks of plaintext and the encryption operation is performed as described in 11.7.2. To decrypt a message, an envelope payload is divided in 128-bit blocks of ciphertext and the decryption operation is performed as described in 11.7.3.

Exactly two EQs form a plaintext or ciphertext block, except in the case of odd payload length, the last block contains a single EQ (see Figure 11-xx). In every envelope, the first payload block is aligned to the end of envelope header. The envelope header itself is not encrypted.

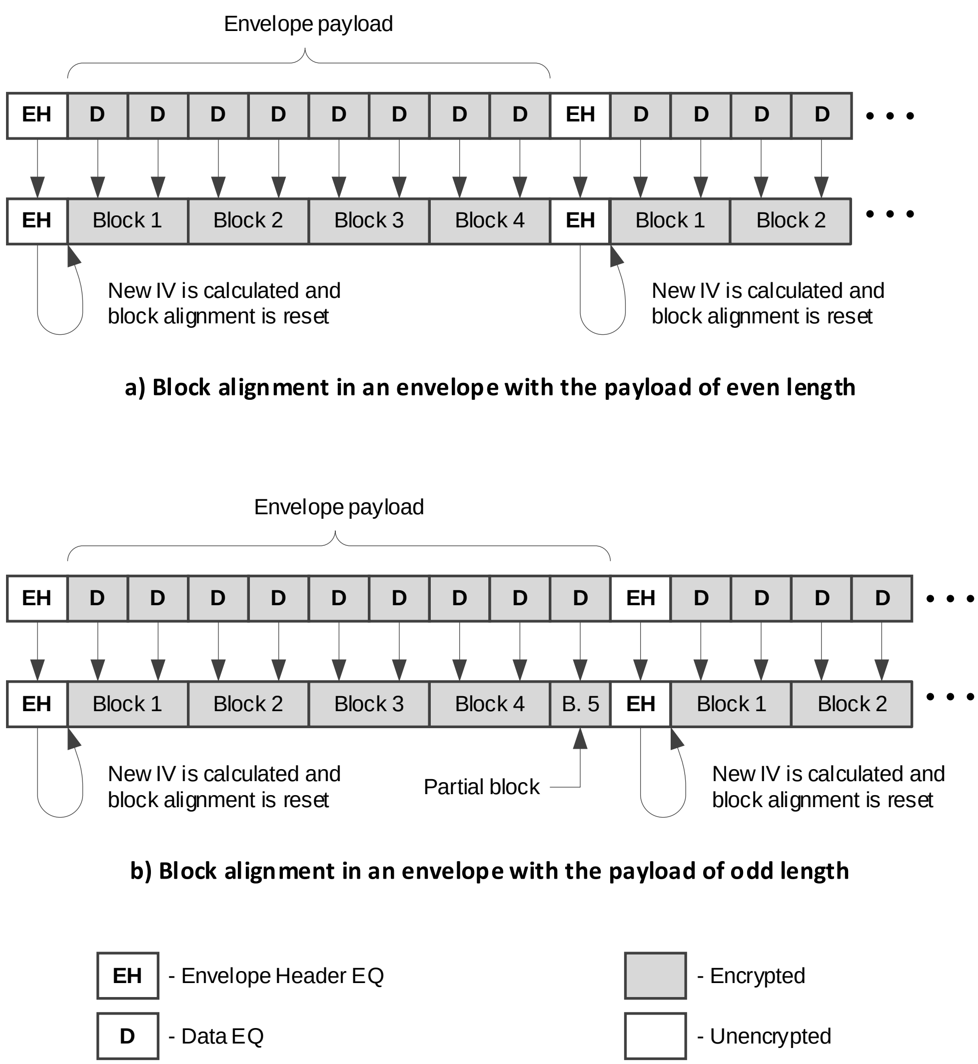


Figure 11-xx – EQ-to-block conversion

11.7.5.1 EQ types that bypass the encryption and decryption processes

As was shown in 11.2.3 and 11.2.4, there are several types of EQs that can appear in the MCRS data path. Some of the EQ types represent control sequences used to signal frame, envelope, or burst delineation (see IEEE Std 802.3, 143.3.3.6.2). These EQs require special treatment by the encryption and the decryption functions, as illustrated in Figure 11-xx and detailed below:

Rate Adjustment (RATE_ADJUST_EQ):

The MCRS ($MCRS_{SEC}$) periodically inserts a series of 33 RATE_ADJUST_EQs to pace the MAC data rate in order to allow the FEC parity data insertion by the PCS. The position of RATE_ADJUST_EQ insertion is determined by the Input process of the MCRS Transmit function and by the Output process of the MCRS Receive function. The RATE_ADJUST_EQ insertion by the Input and the Output processes may happen at different positions within an envelope.

The RATE_ADJUST_EQs are not considered part of envelope (i.e., they are not accounted in envelope length value). As described in 11.2.3 and 11.2.4, these EQs bypass the Encryption/Decryption processes, i.e., they are not encrypted and they do not affect the plaintext or the ciphertext block alignment. The sequence of RATE_ADJUST_EQs may be inserted in the middle of single plaintext or ciphertext block, as illustrated in Figure 11-xxx.

Inter-Envelope Idle (IEI_EQ):

The IEI_EQs are inserted when there is no envelope available for transmission, while the transmission channel itself is active. The IEI_EQs are not part of an envelope. However, unlike the RATE_ADJUST_EQs, they cannot appear in the middle of an envelope. Within the encryption and decryption functions, the IEI_EQs bypass the Encryption/Decryption processes, i.e., they are not encrypted and they do not affect the plaintext or the ciphertext block alignment.

Inter-Burst Idle (IBI_EQ):

The IBI_EQs are inserted when the transmission channel is not active, such as between transmission bursts. The IBI_EQs can only appear in the upstream and are not considered part of an envelope. Within the encryption and decryption functions, the /IBI/ characters bypass the Encryption/Decryption processes, i.e., they are not encrypted and they do not affect the plaintext or the ciphertext block alignment.

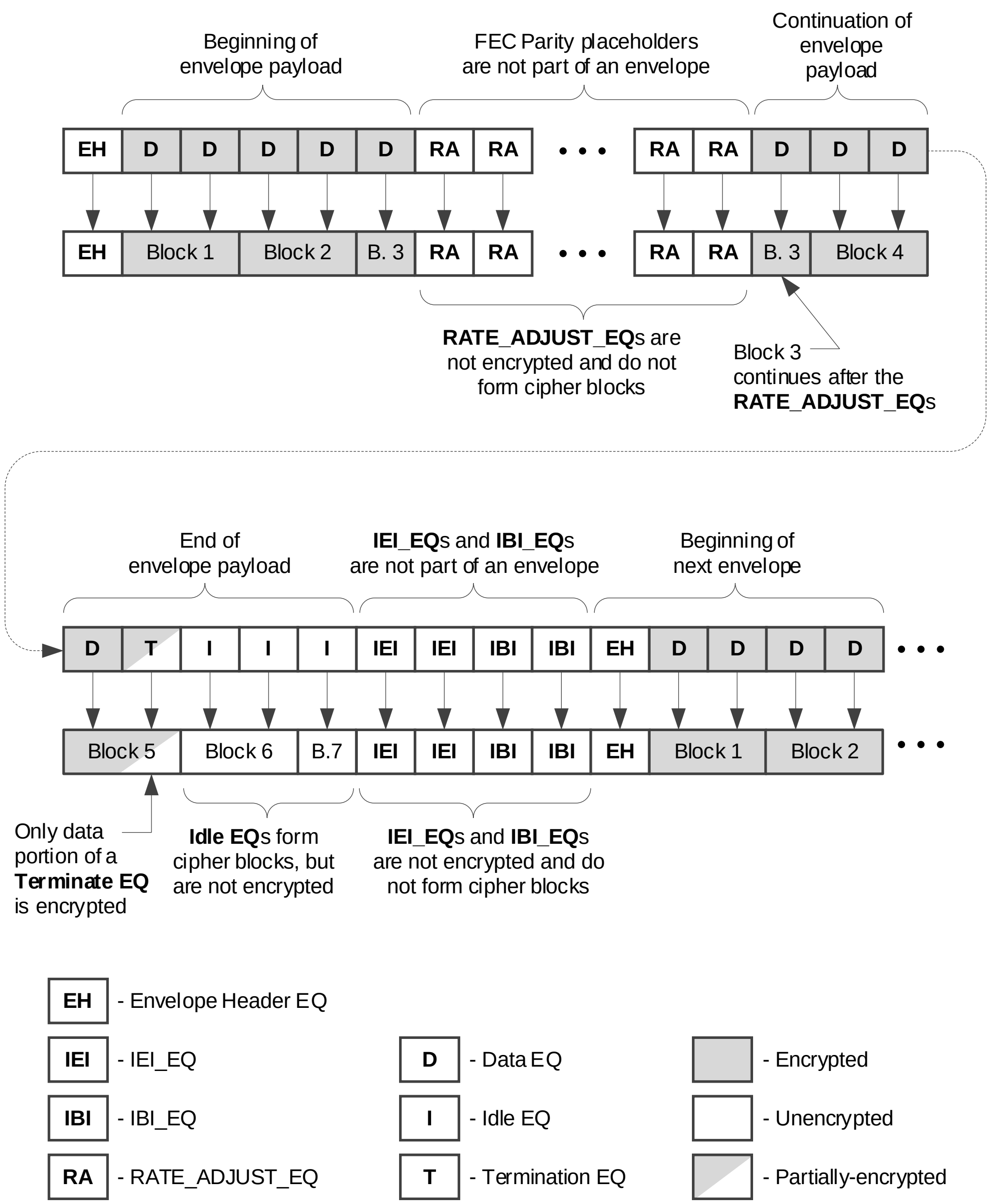


Figure 11-xx – Handling of special EQ types by encryption/decryption function

11.7.5.2 Handling of the control characters in envelope payloads

There are several EQ types that can appear in the payload portion of an envelope: data EQ, Termination EQ, and (regular) Idle EQ. Some of these EQ types may include control characters /T/ and /I/. In order to support 64b/66b encoding in the PCS, these control characters are passed from the input to the output of the encryption or the decryption process unmodified.

As explained in 11.7.2 and 11.7.3, the control characters are left unencrypted by applying a mask to the output of the AES block cipher, before that output is XOR-ed with the plaintext or the ciphertext blocks.

Within the MCRS, an EQ is represented by a 72-bit structure, consisting of 8 control bits `Ctrl[0:7]` and 8 data octets `Data [0:7]` (see IEEE Std 802.3, 143.2.4.1). If the `Ctrl[i]` bit is 1, then the corresponding

Data[i] octet represents a control character, which shall be left unencrypted. Otherwise, the Data[i] is a data octet, which shall be encrypted. The Table 11-x shows all EQ types that may be encountered in the envelope payload and the associated EQ mask. The masks associated with two EQs that form a plaintext or a ciphertext block are combined to form a 128-bit mask that is to be applied to the output of the AES block cipher.

Table 11-x – EQ types and the associated encryption EQ masks

EQ type	Ctrl[0:7] (bin)	Data[0:7] (hex)	Mask (hex)
Data	00000000	xx-xx-xx-xx-xx-xx-xx-xx	FF-FF-FF-FF-FF-FF-FF-FF
Terminate	00000001	xx-xx-xx-xx-xx-xx-xx-FD	FF-FF-FF-FF-FF-FF-FF-00
	00000011	xx-xx-xx-xx-xx-xx-FD-07	FF-FF-FF-FF-FF-FF-00-00
	00000111	xx-xx-xx-xx-xx-FD-07-07	FF-FF-FF-FF-FF-00-00-00
	00001111	xx-xx-xx-xx-FD-07-07-07	FF-FF-FF-FF-00-00-00-00
	00011111	xx-xx-xx-FD-07-07-07-07	FF-FF-FF-00-00-00-00-00
	00111111	xx-xx-FD-07-07-07-07-07	FF-FF-00-00-00-00-00-00
	01111111	xx-FD-07-07-07-07-07-07	FF-00-00-00-00-00-00-00
	11111111	FD-07-07-07-07-07-07-07	00-00-00-00-00-00-00-00
Idle	11111111	07-07-07-07-07-07-07-07	00-00-00-00-00-00-00-00

Note: xx indicates ‘any value’