Understanding the Advanced Encryption Standard

James Ridgway

February 17, 2013

1 The Advanced Encryption Standard

The Advanced Encryption Standard (AES) is the current de facto encryption algorithm for securing sensitive information, and is used by governments and militaries across the world.

In January 1997 the National Institute of Standards and Technology (NIST), a subdivision of the U.S Department of Commerce, start looking for a replacement to the Data Encryption Standard (DES) which had been the encryption standard for nearly two decades. DES is now considered to be too insecure for many applications.

Candidate algorithms were submitted to NIST for consideration as alternatives to DES. All of the algorithms submitted were subjected to review and analysis by NIST and general public. Rijndael (pronounced rain dahl) was finally chosen as the AES out of the final 15 candidate algorithms. The Rijndael algorithm was developed by Belgian cryptographers, Joan Daemen and Vincent Rijmen.

1.1 AES and Rijndael

Rijndael and AES are often used interchangeably, when they are technically different. Strictly speaking AES is an implementation of Rijndael. Rijndael can have any multiple of 32-bit block size and key size between 128- and 256-bits, AES has a strict block size of 128-bits and keys must be either 128-, 192- or 256- bits in length [DR02, p. 31]. The exact specification of AES is formalised in the Federal Information Processing Standard (FIPS) 197 [oST] which was released by the Secretary of Commerce on 6 December 2001.

1.2 Algorithm Overview

AES is a symmetric key block cipher. Symmetric key algorithms are encryption algorithms that use the same cryptographic key for the encryption and decryption process. Block ciphers and stream ciphers are the types of symmetric key algorithms [Sch96, p. 4]. A stream cipher will encrypt bits one at a time. On the other hand, a block cipher will read in a number of bits (called a block) and encrypt them as a single entity. If the plaintext bits that are being encrypted are less than the block size, the plaintext will be padded so that the plaintext matches the block size. AES always produces encrypted messages that are multiples of 128-bits because of the 128-bit block size. Although the algorithm uses a fixed block size it does accept a variety of key sizes: 128-, 192- and 256- bits. AES is based on a design principle called a substitution-permutation network (SPN). An SPN uses substitution boxes (S-Boxes) and permutation boxes (P-Boxes) to apply layers of substitution and permutation to a block. S- and P- box transformation are often achieved using exclusive or and bitwise rotation [DR02][p. 77] [AT90, FNS75, KD79, O'C95].

During the encryption and decryption process repetitive transformation rounds are applied to a 4x4 column-major order matrix - known as a state matrix. The number of rounds of transformation repetitions that are applied to the state matrix are governed by the key size.

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

Each round of encryption involves several processing steps which are used to transform the plaintext to ciphertext. Conversely, inverse processing steps exist to transform ciphertext back to plaintext during the decryption process.

The AES algorithm can be summaries into the following four major steps:

1. Key Expansion
2. Initial Round
   (a) `AddRoundKey`
3. Iterate Rounds
   (a) `SubBytes`
   (b) `ShiftRows`
   (c) `MixColumns`
   (d) `AddRoundKey`
4. Final Round
   (a) `SubBytes`
   (b) `ShiftRows`
   (c) `AddRoundKey`

2 Algorithm Processes

The definitions of the algorithm processes in the section are derived from the descriptions in [DR02].

2.1 Key Expansion

Key expansion derives round keys for the cipher key using the Rijndael Key Schedule. The Key Schedule expands a short key into a number of separate round keys. The key schedule algorithm uses a number of core operations: Rotate, Rcon and Rijndael S-Box.

2.1.1 Operations

Rotation
The rotation operation takes a 32-bit word and rotates it 8-bits to the left, for example, `1D 2C 3B 4A` becomes `2C 3B 4A 1D`.

Rcon
Rcon (or round constants) are values computed in GF(2^8), whereby:

$$Rcon(i) = x^{i-1} \bmod x^8 + x^4 + x^3 + x + 1 \tag{1}$$

with $x = 2$ and i starting at 1.

Note: this operation is performed as a polynomial in the finite field GF(2^8) and not as real integers. The example below shows how to calculate the $Rcon(9)$:

$$\begin{aligned} Rcon(9) &= x^{9-1} \bmod x^8 + x^4 + x^3 + x + 1 && (2)\\ &= x^8 \bmod x^8 + x^4 + x^3 + x + 1 && (3)\\ &= 100000000 \bmod 100011011 && (4)\\ &= 11011 = (27 \text{ in decimal}) && (5) \end{aligned}$$

Rijndael S-Box
The Rijndael S-Box is a lookup table of values (see 2.2), which is used with a `SubWord` function that applies the S-Box to each of the 4-byte of the input word to produce an output word.

2.1.2 Key Schedule Pseudocode

The following pseudocode from FIPS PUB 197 [oST] outlines the operation of the Rijndael Key Schedule algorithm:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp
  i = 0
  while (i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while
  i = Nk
  while (i < Nb * (Nr+1)]
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end
```

2.2 `SubBytes` Step

`SubBytes` consists of applying an S-Box permutation to the bytes of the state matrix. This is the only non-linear transformation in the entire cipher. The S-Box used by AES was designed to minimise the input-output correlation, and difference propagation probability. The S-Box is derived from the multiplicative inverse in GF(2^8), and is defined as:

$$g : a \rightarrow b = a^{-1} \tag{6}$$

2.3 `ShiftRows` Step

The `ShiftRows` step involves applying a left circular shift to each row of the state matrix in turn. The first row remains unchanged, but the first, second and third rows are shift 1, 2 and 3 positions to the left respectively.

$$M = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \quad \texttt{ShiftRows}(M) = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{bmatrix} \tag{7}$$

2.4 `MixColumns` Step

`MixColumns` is a permutation that operates column by column of the state matrix. The `MixColumns` step takes four input bytes and produces four output bytes, whereby each input byte affects all of the output bytes. Each column of the state matrix (represented above by vector a) is treated as a polynomial over GF(2^8) and is multiplied modulo $x^4 + 1$ with a polynomial $p(x)$. The polynomial coefficients have simple values: 0, 1, 2 and 3. Coefficients of 0 and 1 result in no processing, a coefficient of 2 results in a shift to the left and a coefficient of 3 results in a shift to the left and XORing with the unshifted value. Let $b(x) = p(x) \times a(x) \pmod{x^4 + 1}$, then:

$$\begin{bmatrix} b_{0,0} \\ b_{0,1} \\ b_{0,2} \\ b_{0,3} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} a_{0,0} \\ a_{0,1} \\ a_{0,2} \\ a_{0,3} \end{bmatrix} \tag{8}$$

2.5 `AddRoundKey` Step

`AddRoundKey` is a simple XOR of the current round, a, with the subkey, k:

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \tag{9}$$

3 Block Cipher Modes

A block cipher is a simple cryptographic primitive that can covert a fixed-length block of plaintext to a block of ciphertext. *Modes of operation* are used to specify how the cipher encrypts and decrypts blocks. This is especially important for ensuring the confidentiality of long messages [DR02, p. 27].

Using cipher block modes can help to prevent against some attacks such as frequency analysis. Block size can also be an important consideration, as small block sizes can be vulnerable to attacks based on statistical analysis. Most block ciphers use a typical block size of 64-bits, however AES uses a larger 128-bits [MOVR96, p. 225].

The following are popular block cipher modes:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feed back mode (CFB)
- Output Feed Back mode (OFB)

The remainder of this section will explain how each of the above block modes worked. These explanations are based on [Sch96] and [MOVR96].

3.1 Electronic Code Book (ECB)

Electronic Code Book is by far the simplest method of encryption with a block cipher. With ECB each block is encrypted independently, whilst this can present some advantages, it is a method that is heavily undermined by the disadvantages.

The main advantage of ECB is that each block is independent, thus meaning that plaintext does not have to be encrypted or decrypted linearly (from start to finish). In some contexts the ability to randomly and independently access blocks of encrypted data can be useful but this does open up a number of vulnerabilities.

The independent nature of ECB means that two identical plaintext blocks will have identical ciphertext blocks - this is a significant problem. Patterns in plaintext data will still highly correlated between the plaintext and ciphertext blocks. If an attacker has access to several plaintext and ciphertext messages they can start to compile a codebook of known plaintext-ciphertext pairs.

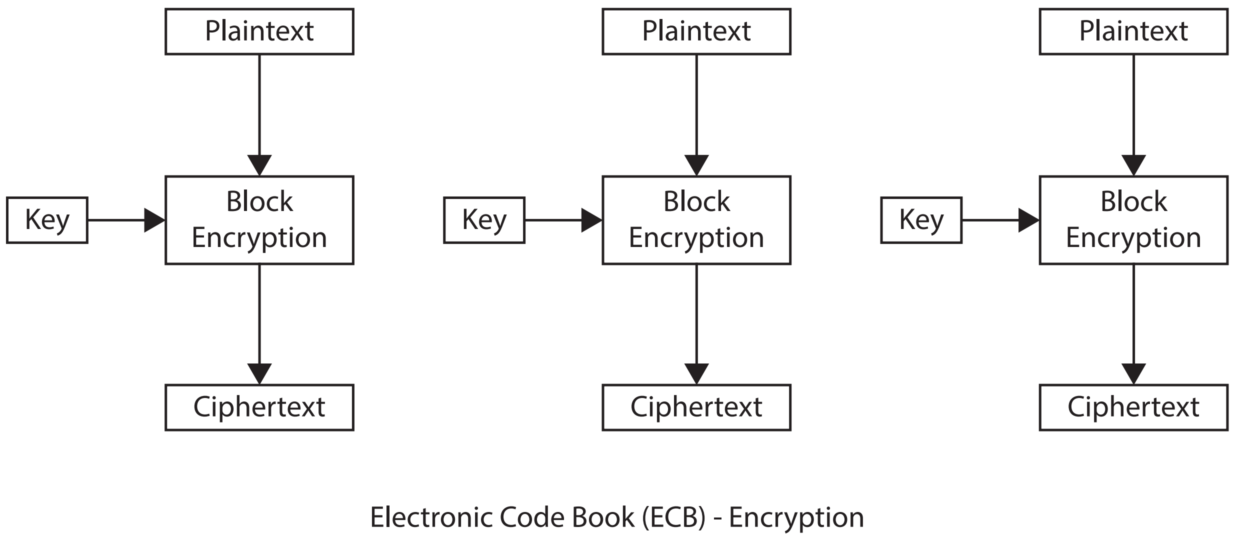


Figure 1: *Electronic Code Book (ECB) - Encryption*

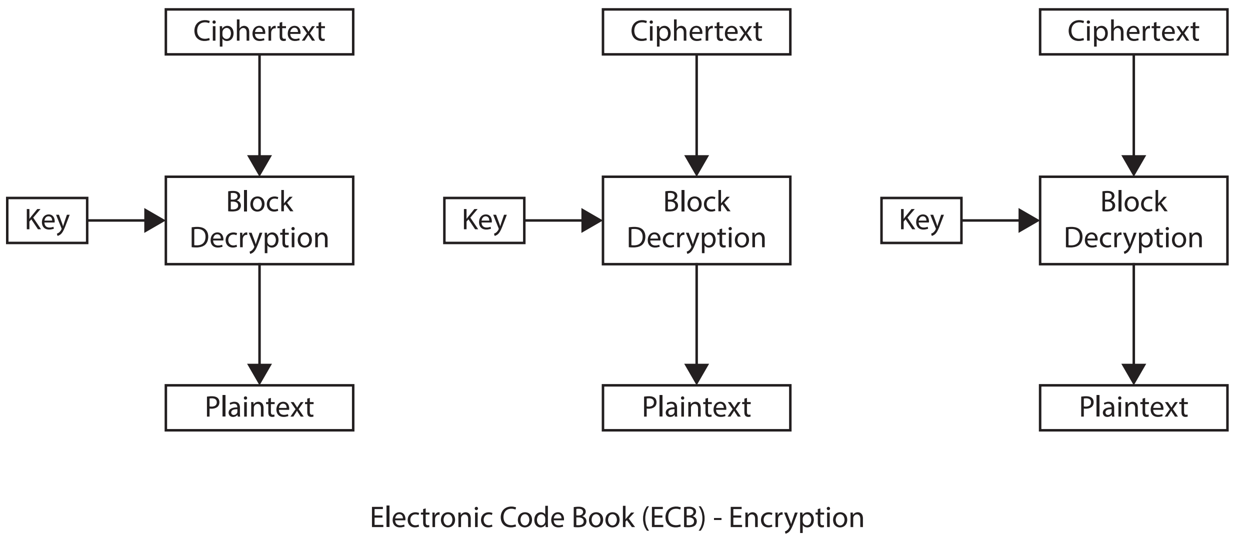


Figure 2: *Electronic Code Book (ECB) - Decryption*

3.2 Cipher Block Chaining (CBC)

Invented by IBM in 1976, CBC ensures that each plaintext block is randomised before it is encrypted with the block cipher. CBC XORs each block of plaintext with the previous ciphertext block before encrypting with the block cipher. To ensure randomisation an Initialisation Vector (IV) is used for the first block. The CBC method makes each block of ciphertext dependent on all of the plaintext blocks up to that point. Unlike ECB, CBC must encrypt and decrypt linearly because of this dependancy.

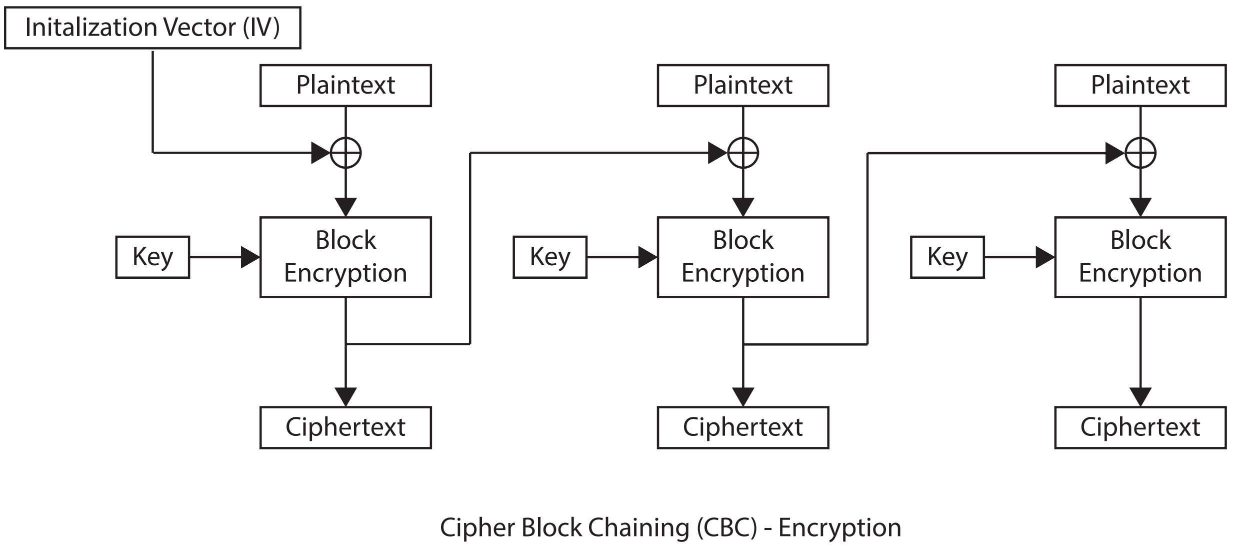


Figure 3: *Cipher Block Chaining (CBC) - Encryption*

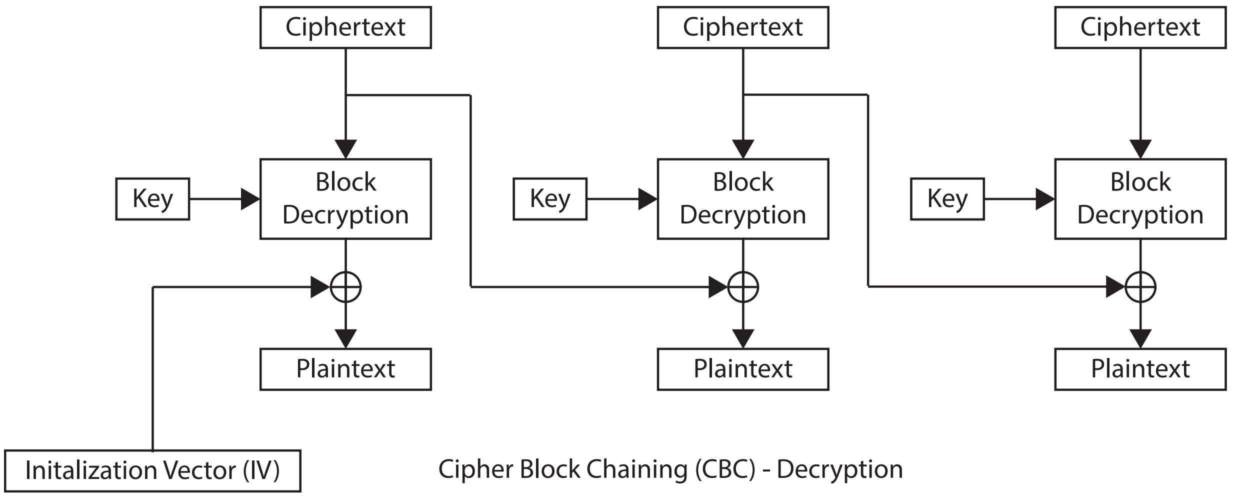
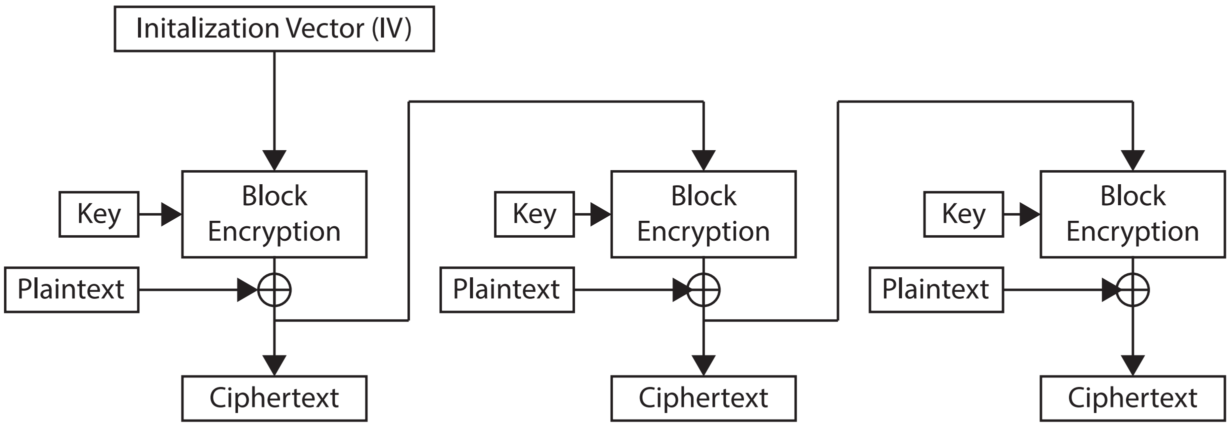


Figure 4: *Cipher Block Chaining (CBC) - Decryption*

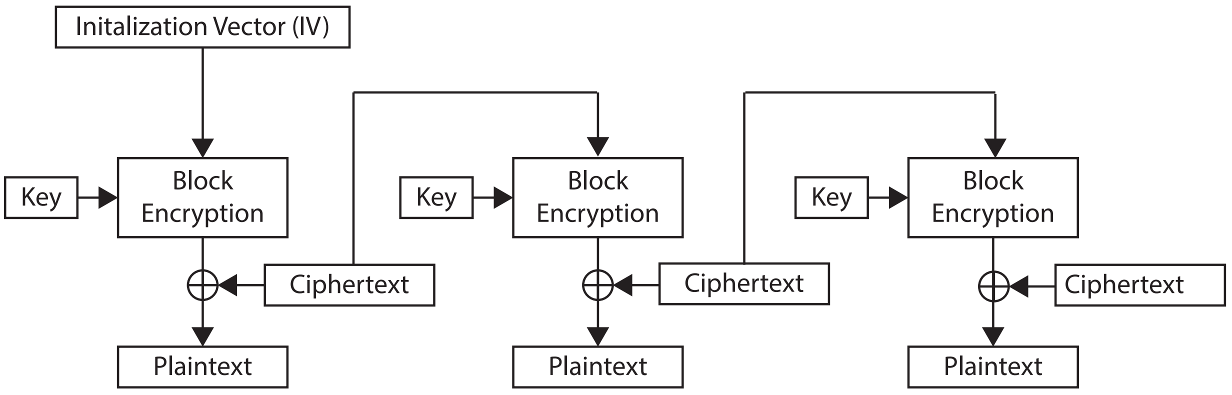
3.3 Cipher Feed Back mode (CFB)

ECB and CBC are modes specifically for block ciphers, but block ciphers can also use stream cipher modes. CFB is a *self-synchronising stream cipher* - using this mode will turn a block cipher into a self-synchronising stream cipher. Self-synchronising stream ciphers use previous ciphertext bits to generate the keystream bits.



Cipher Feed Back (CFB) - Encryption

Figure 5: *Cipher Feed Back (CFB) - Encryption*



Cipher Feed Back (CFB) - Decryption

Figure 6: *Cipher Feed Back (CFB) - Decryption*

3.4 Output Feed Back mode (OFB)

OFB is a synchronous stream cipher mode whose keystream is independent of the message being encrypted (independent of plaintext and ciphertext). OFB produces ciphertext by

XORing the generated keystream with the plaintext.

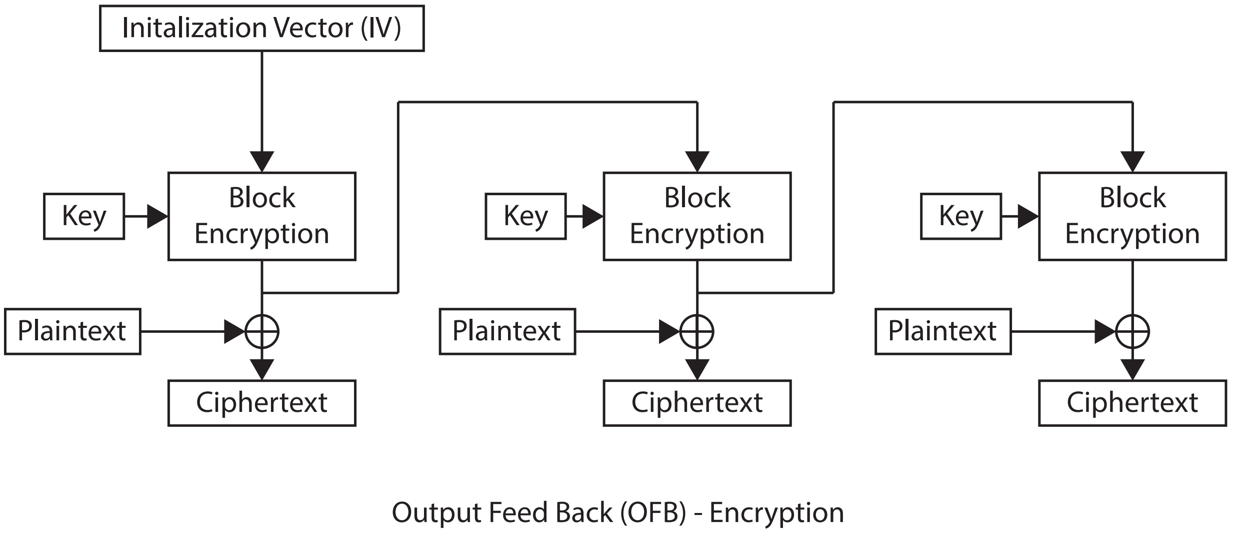


Figure 7: *Output Feed Back (OFB) - Encryption*

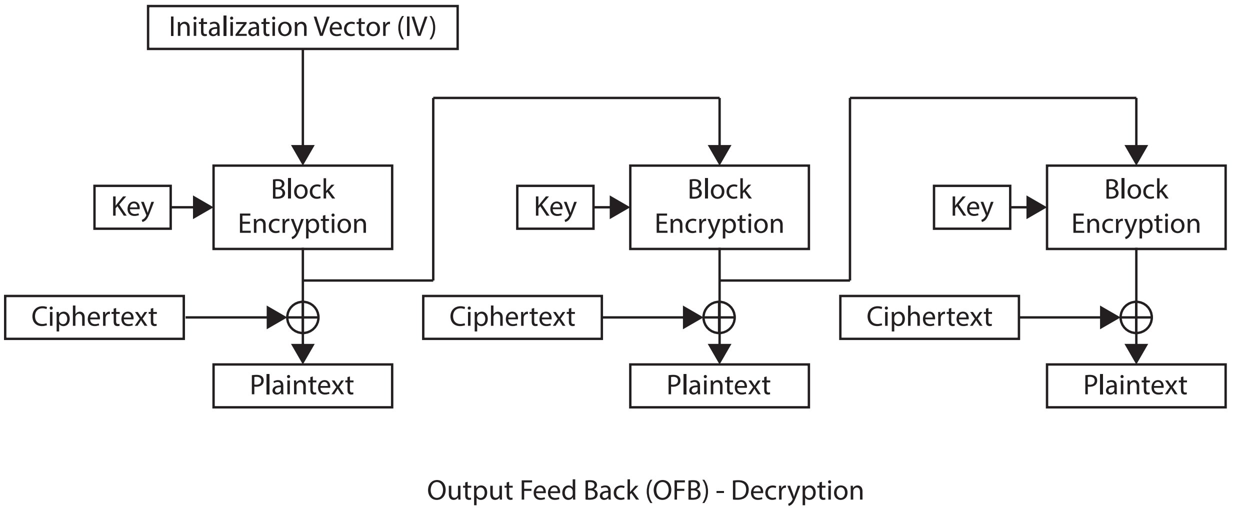


Figure 8: *Output Feed Back (OFB) - Decryption*

References

[AT90] C. M. Adams and S. E. Tavares. The structured design of cryptographically good s-boxes. *Journal of Cryptology*, 3(1):27–42, 1990.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[FNS75] H. Feistel, W. A. Notz, and J. L. Smith. Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545 – 1554, Nov. 1975.

[KD79] J.B. Kam and G.I. Davida. Structured design of substitution-permutation encryption networks. *Computers, IEEE Transactions on Computers*, C-28(10):747–753, Oct. 1979.

[MOVR96] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1996.

[O'C95] L. O'Connor. On the distribution of characteristics in bijective mappings. *Journal of Cryptology*, 8(2):67–86, 1995.

[oST] National Institute of Science and Technology. Federal information processing standards publication 197.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, Inc., 1996.